Nr. 3/86 März
DM 6.50, sfr 6.50, öS 50, Lit 5900, hfl 7.50

peeker

Reelle Zahlen

ProDOS-Subdirectory

Kyan-Aufbaukurs

UCSD-Dateien

Wordstar-Anpassung

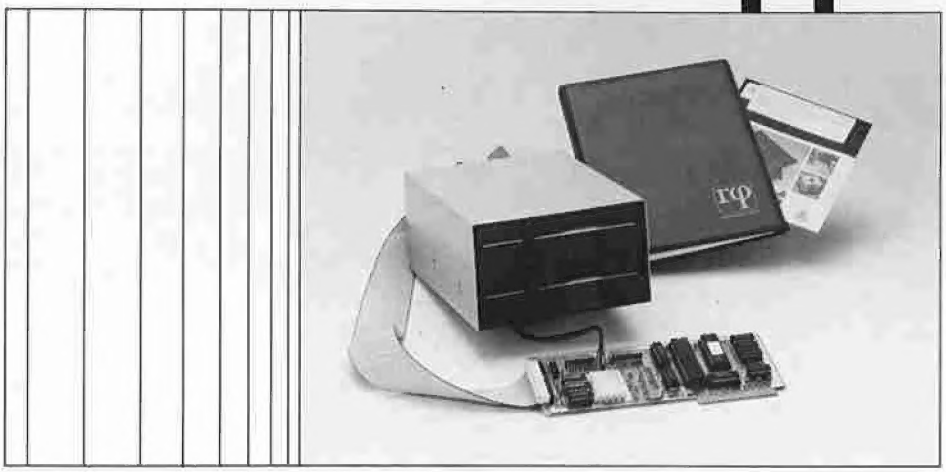
Okidata-Microline

Mockingboard

Großer Sonderbericht
Macintosh Plus

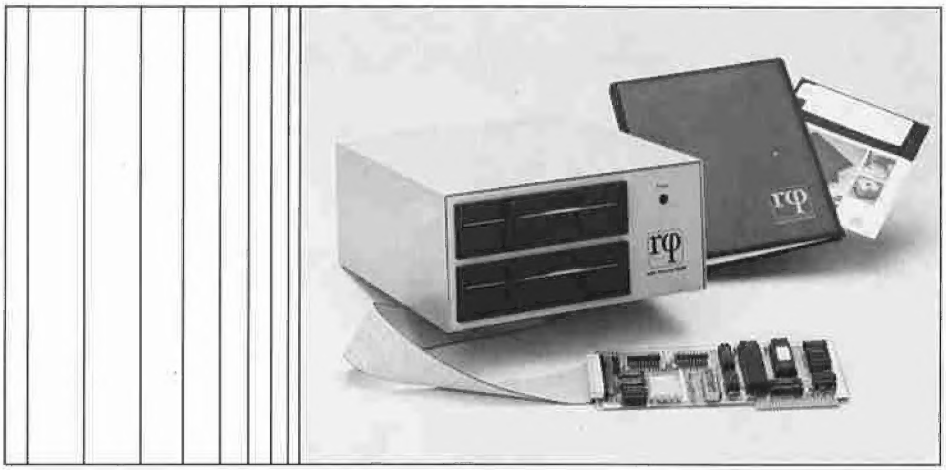
Hüthig
PUBLIKATION

Anschlußfertig für Apple® II, //e...



erphi
Doppellaufwerk DL 280
2 Laufwerke
mit je 640 kB formatiert
im Gehäuse incl.
erphi
Autopatch-Controller AFDC 3
Handbuch und Diskette mit
Dienstprogrammen

Verkaufspreis incl. MwSt.
DM 1.298,–



erphi
Floppy-Subsystem FSS 280
für kommerzielle Anwendungen
2 Laufwerke
mit je 640 kB formatiert
im Gehäuse mit eigener
Stromversorgung incl.
erphi
Autopatch-Controller AFDC 3
Handbuch und Diskette mit
Dienstprogrammen

Verkaufspreis incl. MwSt.
DM 1.698,–

Noch universeller und noch komfortabler...

Betriebssysteme
DOS 3.3
DiversiDOS 2-c, 4-c,
ProDOS 1.0.1, 1.1.0, 1.1.1
Pascal 1.1, 1.2
CP/M 2.20, 2.23, 2.26

SLOT-Unabhängig

Die bisherigen Vorzüge bleiben erhalten, wie

Autopatch-Boot
automatische Erkennung und Erweiterung der Betriebssysteme während des Bootvorgangs (Betriebssystem auf der Boot-Diskette bleibt unverändert)

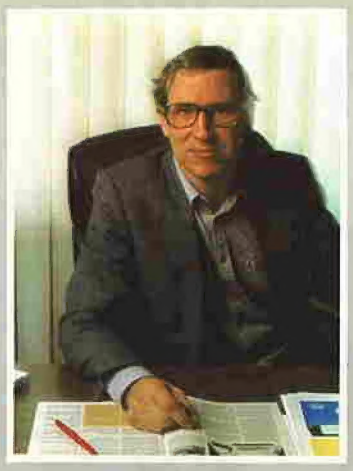
Originalsystem-Boot
Von herkömmlichen Apple®-Disketten (unabhängig vom Laufwerksformat) weiterhin möglich

Problemloses Übertragen
herkömmlicher Apple®-Software auf Disketten höherer Kapazität durch SIM 35-Hilfsprogramm (simuliert 35-Spur-Laufwerk unabhängig vom tatsächlichen Laufwerksformat)

rφ
erphi electronic
GmbH

erphi
electronic
GmbH
Dammweg 3
D-8011 Großhelfendorf
Telefon (0 80 95) 441
Telex 528 021 erphi d

ikw



EDITORIAL

Kyan-Club

Obwohl wir Kyan-Pascal für eine hervorragende Pascal-Implementierung für den Apple II halten, ist die Resonanz doch größer, als wir erwartet haben, denn in Kürze werden wir den 500. Besteller begrüßen dürfen. Dies gibt uns die Möglichkeit, nicht nur den Preis von DM 170,– für die Version 2.0 über den Anmeldeschluß hinaus zu halten, sondern Ihnen zudem einen erweiterten Service in Form eines informellen Kyan-Clubs anzubieten. Was ist darunter zu verstehen?

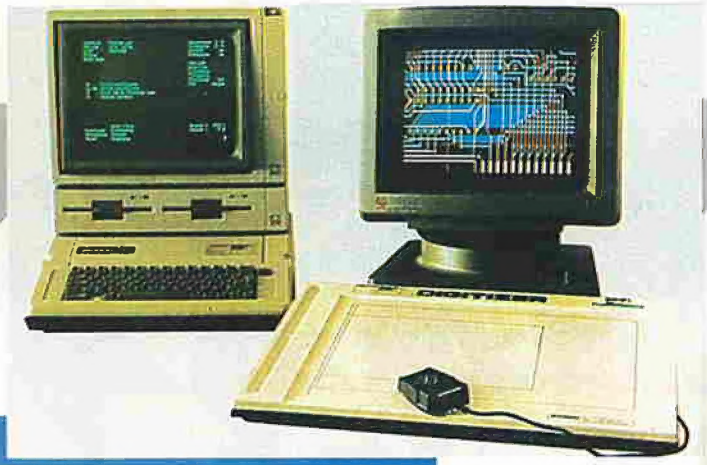
Wer Kyan-Pascal über unseren Software-Service bestellt, wird automatisch Kyan-Club-Mitglied und kann damit bei Bedarf folgende Leistungen in Anspruch nehmen:

1. Wenn Sie Kontakt zu anderen Kyan-Besitzern aufnehmen wollen, so brauchen Sie uns nur Ihre Erlaubnis zur Aufnahme Ihrer Anschrift in unsere Kyan-Liste zu geben. Im Gegenzug erhalten Sie von uns kostenlos ein nach Postleitzahlen sortiertes Verzeichnis aller Kyan-Kontaktsuchenden. Sie können dann beispielsweise örtliche Treffen vereinbaren oder telefonisch Tips und Tricks austauschen.
2. Alle Kyan-Toolkits, die teils von der Firma Kyan-Software und teils von unserem eigenen Software-Service entwickelt werden, können Sie zu Club-Sonderpreisen erwerben, die deutlich unter den jeweiligen Normalpreisen liegen werden.

Weitere Details zu diesen Sonderleistungen entnehmen Sie bitte unserem Rundschreiben, das wir in diesen Tagen verschickt haben. Da Sie sicherlich bereits ungeduldig auf Kyan-Pascal warten, haben wir darin allen Frühbestellern die Altversion 1.2 kostenlos angeboten, denn die beachtlich erweiterte Version 2.0, zu der uns bislang nur die „Beta-Version" vorliegt, wird erst Ende März erscheinen.

Ulrich Stiehl

INHALT



Impressum

Peeker
3. Jahrgang 1986
ISSN 0176-9200

Verleger und Herausgeber:
Dipl.-Kfm. Holger Hüthig
Geschäftsführung Zeitschriften:
Heinz Melcher
Chefredakteur: Ulrich Stiehl (us)

Hüthig
PUBLIKATION

Telefonnummern:
Zentrale: 06221/489-1
Redaktion: 06221/489-352
Anzeigen: 06221/489-206
Abonnement: 06221/489-283
Software: 06221/489-231
Bücher: 06221/489-353
(Bestellungen bitte nur schriftlich)

Abonnement:
Der Abonnent kann seine Bestellung innerhalb von 7 Tagen schriftlich durch Mitteilung an den Dr. Alfred Hüthig Verlag GmbH, Postfach 102869, 6900 Heidelberg 1, widerrufen. Zur Fristwahrung genügt die rechtzeitige Absendung des Widerrufs (Datum des Poststempels). Das Abonnement verlängert sich zu den jeweils gültigen Bedingungen um ein Jahr, wenn es nicht zwei Monate vor Jahresende schriftlich gekündigt wird. Die Abonnementsgelder werden jährlich im voraus in Rechnung gestellt, wobei bei Teilnahme am Lastschriftabbuchungsverfahren über die Postscheckämter und Bankinstitute eine vierteljährliche Abbuchung möglich ist.
Nichterscheinen infolge höherer Gewalt berechtigt nicht zu Ansprüchen gegen den Verlag.

PEEKER

Heft 3/1986

Anschrift:
Dr. Alfred Hüthig Verlag GmbH
Im Weiher 10, Postfach 10 28 69
6900 Heidelberg
Telefon (0 62 21) 4 89-1
Telex 4-6 17 27 hued d
Telefax (0 62 21) 489 279
BTX * 51851 #

Vertrieb:
Erscheinungsweise: 12 Hefte jährlich,
Erscheinungstag jeweils 1 Woche vor Monatsbeginn.
Jahresabonnement DM 72,–, einschließlich MwSt,
im Inland portofrei. Einzelheft DM 6,50
Vertrieb Handel:
MZV – Moderner Zeitschriften Vertrieb GmbH
Breslauer Str. 5, Postfach 1123,
8057 Eching b. München,
Tel. 089/319 10 67, Telex 0 522 656
Vertriebsleitung:
Walter Menzel, Tel. (0 62 21) 48 92 80

Bankverbindungen:
Zahlungen: an den Dr. Alfred Hüthig Verlag GmbH, D-6900 Heidelberg 1: Postgirokonten: BRD: Karlsruhe 485 45-753; Österreich: Wien 7555888; Schweiz: Basel 40-24417; Niederlande: Den Haag 1 457 28; Italien: Mailand 47718; Belgien: Brüssel 7230 26; Dänemark: Kopenhagen 349 69; Norwegen: Oslo 994 24; Schweden: Stockholm 5477 76-5
Bankkonten: Landeszentralbank Heidelberg 67 207 341; BLZ 672 000 00; Deutsche Bank Heidelberg 0 2165 041; BLZ 672 700 03; Bezirkssparkasse Heidelberg 204 51, BLZ 672 500 20.

Herstellung:
Produktionsleitung: Gunter Sokollek
Gestaltung: Rainer Schmitt
Titelbild: Werner Hable
Satz und Druck: Heidelberger Verlagsanstalt
Printed in Germany

Der neue Macintosh

Erste Eindrücke zum Mac Plus

von Ulrich Stiehl

Am 21.1.1986 wurde in München der neue Macintosh präsentiert, der ab Februar lieferbar ist und an die Stelle des alten Macintosh treten soll, der vor exakt zwei Jahren am 25.1.1984 in Frankfurt vorgestellt wurde, aber erst sechs Monate später ab etwa Juni 1984 erhältlich war.
Der neue Mac verhält sich zum alten wie der Apple IIe zum II+, d.h. der neue Macintosh = Macintosh Plus = Mac Plus = Mac+ ist kein völlig neues, sondern nur ein erweitertes und verbessertes Modell, das zum Alt-Modell weitgehend kompatibel ist. Um es vorweg zu sagen: Der Mac+ hinterließ einen durchaus ausgereiften und positiven Eindruck. Allerdings haben wir noch keinen eigenen Mac+ in der Redaktion, so daß ein spezieller Leistungstest noch aussteht. Im übrigen werden wir den Macintosh im Peeker jetzt ausführlicher als bisher behandeln.

1. Produktpalette

Bevor wir auf den Mac+ im Detail eingehen, möchten wir zunächst die langfristige Produktplanung der Firma Apple umreißen, die für jede Produktkategorie ein offenes und ein geschlossenes System vorsieht. Die offenen Systeme („Open Systems") sind erweiterbar (Slot-Konzept usw.), die geschlossenen Systeme („Closed Systems") sind es nicht:
1. Beim Apple II gilt der IIe als offenes und der IIc als geschlossenes System.
2. Beim Macintosh ist der Mac+ ein geschlossenes System. Der offene „Slot-Mac" soll laut dem Apple-Geschäftsführer R.M.Deja in etwa einem Jahr folgen.
3. Als Nachfolgemodell zum Apple II soll zunächst ein offenes System „in 6 bis 18 Monaten" (R.M.Deja) erscheinen. Die Details, die man einige Tage nach der Münchener Pressekonferenz im „Chip" (2/1986, S. 10f.) über den „Apple IIx" lesen konnte, wurden nicht bestätigt. Kein Wunder, denn dem alten (!) Macintosh wurde beispielsweise ein 320K-Laufwerk attestiert, was die Vermutung nahelegt, daß bei dem „exklusiven Blick hinter die Kulissen" aus Versehen ein Fremdgerät erspäht wurde.

2. Der alte Mac

Als wir vor genau einem Jahr detaillierte Leistungstests über den Macintosh veröffentlichten (Heft 3/85, S. 44-51), hagelten ohnmächtige Schimpftiraden aus dem Lager der Firma Apple und ihrer Händler auf den Peeker nieder, ohnmächtig deshalb, weil jeder Mac-Besitzer die Tests exakt nachvollziehen konnte. Die Mängel waren jedoch zu offenkundig, als daß man sie hätte verschweigen können. So wurde beispielsweise gezeigt, daß ein simpler Directory-Befehl in bezug auf einen Datenträger mit 100 Dateien beim alten Mac 500 Sekunden dauerte, während jeder andere Mikrocomputer für denselben Befehl nur etwa 5 Sekunden benötigte. Man bezichtigte uns des Auslotens von Extremsituationen, die in der Praxis selbst bei 20M-Festplatten niemals vorkommen würden. Weit gefehlt! Denn im nachhinein erfährt unsere Mac-Kritik ihre volle Bestätigung just durch die Firma Apple selbst, die mit dem neuen Mac exakt das ausbügelt, was wir am alten Mac kritisiert haben. So hat beispielsweise der neue Mac jetzt nicht mehr Pseudo-Subdirectories, sondern echte Subdirectories, womit die Unzulänglichkeit des alten Directory-Befehls behoben wurde.

3. Technisches zum Mac+

Der neue Mac zeichnet sich durch folgende Erweiterungen aus:
– 1024K statt 128K bzw. 512K RAM
– 128K statt 64K ROM
– Zusätzliche Schnittstelle
– DIN-Tastatur
– 800K- statt 400K-Drives
– Hierarchischer Finder
Über weitere Neuerungen wie „Cache"-RAM-Speicher, Laserwriter Plus mit Original-Schriftarten usw. werden wir zu einem späteren Zeitpunkt berichten.

3.1. 1-Megabyte-RAM

Wie erinnerlich, wurde der alte Mac zunächst mit 128K RAM ausgeliefert. Wenige Monate nach der Mac-Einführung (im Juni 1984 in Deutschland bzw. im Januar 1984 in den USA) sah man sich genötigt, eine 512K-Speichererweiterung anzubie-

ten. Die Tatsache, daß die ICs nicht gesockelt waren und somit die gesamte Platine ausgetauscht werden mußte, läßt den Schluß zu, daß man zunächst keine Speichererweiterung für den 128K-Mac ins Auge gefaßt hatte, denn die Lisa war noch bis zum Frühjahr 1985 lieferbar und somit war eine Produktdifferenzierung zu dem teureren Modell erforderlich. Es zeigte sich jedoch alsbald, daß ein 128K-Mac praktisch unbrauchbar war, weil erstens der für damalige Verhältnisse gigantische Bildschirmspeicher (ca. 22K) einschließlich des Hintergrundspeichers für nichtaktive Fenster sowie zweitens die „auf die schnelle" in Hochsprachen erstellten Anwenderprogramme einen Großteil des RAMs in Beschlag nahmen, so daß für Daten kein Platz mehr übrigblieb.
Zum Vergleich hat der momentane Atari 520+ einen 32K-Bildschirmspeicher, und der zukünftige Atari wird sogar ein 128K-Video-RAM aufweisen. Ob sich dieser Trend zu Grafik-Computern als richtig erweisen wird, sei dahingestellt. Zumindest kann man festhalten, daß der Speicherbedarf beim alten Mac von der Firma Apple völlig falsch eingeschätzt worden war.

3.2. 128K-ROM

Wir haben im Peeker bereits gezeigt (Heft 5/85, S. 41ff. sowie Heft 9/85, S. 67f.), daß der Macintosh zwar über einen 8-MHz-68000-Prozessor verfügt, daß jedoch wegen der Wartezyklen nur eine effektive Taktfrequenz von ca. 5,2 MHz bei RAM-residenten Programmen erreicht werden kann. Des gilt übrigens nicht für den Atari 520+, der es bei unserem Test (5/85, S. 41) auf ca. 7,9 MHz bringt, wie unser Pascal-Autor Dieter Geiß auf seinem Atari nachgeprüft hat. Damit ist der Atari im RAM immerhin 50% schneller als der Macintosh (7,9 - 5,2 = 2,7 oder ca. 50% von 5,2). Da laut dem technischen Leiter der Firma Apple, Herrn Zimmermann, sich bezüglich der Taktfrequenz beim neuen Mac nichts geändert hat, wurde das ROM von 64K auf 128K aufgestockt und enthält jetzt zusätzlich einen (weiteren) Teil des Betriebssystems sowie darüber hinaus u.a. die besonders zeitkritischen Fließkomma-Routinen. Damit können nun endlich wesentliche Betriebssystemfunktionen im ROM mit den vollen 8 MHz ausgeführt werden.

Die grafische Bildschirm-Textausgabe bzw. das Scrollen von Textzeilen wurde von uns in Heft 1/85, S. 76 heftig kritisiert. Angeblich sollen jetzt die der Grafik-Ausgabe zugrunde liegenden sog. Quickdraw-Routinen, die seinerzeit Bill Atkinson entwickelt hatte, laut Apple um 70% verbessert worden sein (was wir jedoch erst noch verifizieren müssen), womit die „Byte" (Heft 2/84, S. 37) mit ihren Äußerungen („Quickdraw is very fast"; „a 24K-byte package of highly optimized 68000 code") völlig daneben lag, denn eine Assembler-Routine, die sich um 70% verbessern läßt, kann wohl kaum „very fast" gewesen sein.

3.3. Schnittstelle SCSI

Als kleiner Vorgeschmack auf den zukünftigen „Slot-Mac" wurde der Mac+ mit einer weiteren Schnittstelle namens SCSI (= Small Computer System Interface) versehen, die sich erstens für den Datentransfer zwischen Mac+ und Großrechner sowie für den Anschluß von Festplatten usw. sowie weiteren Peripheriegeräten im Daisy-Chain-Verfahren eignet. Wie bekannt, verfügte der alte Mac über zwei serielle Schnittstellen mit RS-422-Protokoll, wobei die RS-422- eine Weiterentwicklung der RS-232- bzw. V24-Schnittstelle darstellt. Eine der beiden Schnittstellen wurde regelmäßig durch den Drucker (Imagewriter o.ä.) belegt, und an das andere Interface konnte theoretisch eine Festplatte angeschlossen werden. Während die klassische V24 eine Datenübertragungsrate von maximal 19.200 Bits/s (ca. 2,4K/s) hat, erzielt die Mac-RS-422 eine Rate von normalerweise 230.000 Bits/s (ca. 28K/s) oder mit externem Taktgenerator von maximal 920.000 Bits/s (ca. 112K/s). Für Festplatten ist dies trotzdem viel zuwenig, denn es handelt sich hierbei um theoretische Werte, die durch das jeweilige Betriebssystem – hier durch den Finder – stark relativiert werden. So soll etwa das Öffnen von „MS-Word", das wir nicht besitzen, selbst bei einer RAM-Disk ca. 6,5s dauern. Wenn wir nun annehmen, daß dabei 100K eingeladen werden, so sind wir bereits bei
100 : 6,5 = 15K/s
angelangt. Da eine RAM-Disk stets schneller als jede Hard-Disk ist, kommen wir bei einer Festplatte mit serieller Schnittstelle auf kümmerliche Werte. Die Firma Apple gestattete deshalb bereits vor dem Erscheinen des Mac+, daß ohne Garantieverlust eine Festplatte ins Mac-Gehäuse eingebaut werden durfte („Hyperdrive"). Darüber hinaus stehen nunmehr dem Käufer des Mac+ durch die zusätzliche Schnittstelle, über deren technische Besonderheiten wir mangels aussagekräftiger Unterlagen erst zu einem späteren Zeitpunkt berichten können, erweiterte Anschlußmöglichkeiten offen. Laut dem Geschäftsführer M.Klein der Firma MKV in Mannheim, der manchem Peeker-Leser von seinem Z80-Buch her bekannt sein dürfte, hat die SCSI eine Übertragungsrate von 1 Megabit/s (ca. 128K/s). Außerdem konnte er sehr hohe Lade- und Kopiergeschwindigkeiten in Verbindung mit dem Cache-Speicher messen. So soll etwa Macwrite jetzt in 1s von Diskette (nicht von der RAM-Disk!) geladen werden, wenn sich nach einem Warmstart das Betriebssystem bereits im Speicher befindet. Dies sind Werte, die alles bisher Bekannte in den Schatten stellen. Das „ewige Warten", das viele Mac-Benutzer beklagt haben, dürfte damit endlich vorbei sein.

Hinweis: Die Bezeichnung „Slot-Mac" wird sich möglicherweise als unzutreffend erweisen. Wahrscheinlicher ist, daß beim zukünftigen „offenen Mac" – ähnlich wie beim Atari – eine größere Anzahl von Anschlüssen nach außen geführt wird, so daß es Steckkarten im klassischen Sinne nicht mehr geben wird. Doch ist dies alles bislang noch Spekulation.

3.4. Cursor-Tastatur

Das Fehlen von Cursor-Tasten hat sich beim alten Mac als ein erheblicher Nachteil erwiesen. Vereinfachend kann man sagen, daß die Maus für große Distanzen und die Cursor-Tasten für kurze Distanzen optimal sind. Wer sich bei einem Textverarbeitungsprogramm um ein paar Buchstaben zurückbewegen wollte (Kurzdistanz), mußte bereits zur Maus greifen. Dies ist völlig unergonomisch, weil das Betätigen der Cursor-Tasten viel schneller vonstatten geht, als wenn man die rechte Hand von der Tastatur abhebt, die Maus rührt und klickt und dann wieder die rechte Hand in die DIN-Normposition bringt. Viele dürften sich damit beholfen haben, daß sie mit der Delete-Taste einfach die letzten Buchstaben bis zur gewünschten Cursor-Position gelöscht haben. Ein unmöglicher Zustand für denjenigen, der effizient und schnell schreiben mußte!
Mit der neuen DIN-Normtastatur mit Ziffernblock sind diese Probleme aus dem Weg geräumt, doch müssen jetzt wohl erst alle Altprogramme dahingehend modifiziert werden, daß sie die Cursor-Tasten erkennen.

3.5. 800K-Drives

Die neuen 800K-Drives sind wie die alten 400K-Drives 3,5-Zoll-Sony-Laufwerke, die jedoch jetzt beidseitig (statt einseitig) benutzt werden. Langfristig gesehen sollen alle Apple-Modelle mit 3,5-Zoll-Drives ausgestattet werden, wobei die 800K-Drives für den Apple IIe bereits vorliegen.
Die 800K kommen übrigens so zustande, daß zwar – wie etwa beim normalen 640K-Laufwerk – jede Seite 80 Spuren aufweist, jedoch – im Gegensatz zu den meisten anderen Aufzeichnungsverfahren – die Anzahl der Blöcke von den inneren zu den

äußeren Spuren zunimmt (Näheres siehe Heft 3/85, S. 48).
Ob sich bei den neuen 800K-Disketten eine einzelne Datei über beide Seiten erstrecken kann, konnte bislang nicht verifiziert werden.

3.6. Hierarchischer Finder

Der alte Finder, wie das Mac-DOS genannt wird, war das zugleich schönste und schlechteste Betriebssystem, das die Firma Apple bislang entwickelt hat. Das schönste insofern, als sich viele von dem „Visual Interface" einnehmen ließen. Das schlechteste insofern, als dem Kenner von Betriebssystemen die mangelnde Effizienz zu offenkundig vor Augen geführt wurde. Dies war auch der Firma Apple bewußt, sonst wären nicht innerhalb von weniger als zwei Jahren quasi pausenlos neue Versionen des Finders von ursprünglich 1.0 bis nunmehr 5.1 erschienen. Ohne auf Einzelheiten einzugehen, seien zwei globale Mängel herausgehoben:
1. Die fehlende hierarchische Dateistruktur machte den alten Finder in Verbindung mit größeren Datenträgern wie Festplatten sowie den neuen 800K-Drives schlechthin zur Farce (s.o.). Über die technischen Einzelheiten des neuen Finders 5.1 werden wir einen gesonderten Beitrag veröffentlichen, der auch auf die Beziehungen zu dem hierarchischen ProDOS eingehen wird.
2. Die Verarbeitungsgeschwindigkeit war einfach zu niedrig. Wer beispielsweise noch das alte Visicalc 3.3 für den Apple II kannte, das nach einem Kaltstart Betriebssystem *und* Programm in 8 Sekunden von der Diskette einlud, mußte sich beim Macintosh an ganz andere Zeiten gewöhnen. Als Faustregel mußte man mit 60 Sekunden rechnen, bis man sich nach einem Kaltstart beim Macintosh in irgendeinem Anwenderprogramm befand. Wie oben bereits angedeutet, wurden in dieser Hinsicht erhebliche Verbesserungen vorgenommen.

4. Ökononisches zum Mac+

Nach unseren eigenen, vorsichtigen Schätzungen gibt es in der Bundesrepublik etwa 80.000 Apple-II- und etwa 5.000 Mac-Besitzer. Zum Vergleich hat der Peeker knapp 20.000 Leser, unter denen sich ein paar hundert Mac-Besitzer befinden, insbesondere solche, die neben dem Mac (z.B. im Büro) noch über einen Apple II (z.B. zu Hause) verfügen. Trotz dieses mißlichen Verhältnisse möchten wir dem neuen Mac+ in den nächsten Peeker-Heften durch eine verstärkte Berichterstattung noch einmal eine Chance geben. Wird sich der Mac+ durchsetzen können?

4.1. Mac+ und Atari

Nachdem im Herbst 1985 der 520+ von Atari erschienen und die Amiga von Commodore für das Frühjahr 1986 angekündigt worden ist, wird man in diesem Jahr unter drei Computern wählen können, die hinsichtlich der Konzeption unmittelbar vergleichbar sind. Da die Preise für die Amiga bislang noch nicht festliegen, beschränken wir uns auf einen Vergleich zwischen Mac+ und 520+:

– Beide benutzen den 68000-Prozessor von Motorola in der 8-MHz-Version.
– Beide haben ein großes ROM, beim Mac+ 128K, beim Atari 192K.
– Beide haben ein RAM mit z.Zt. 1 Megabyte, aufrüstbar auf theoretisch bis zu 4 Megabytes (mit zukünftigen ICs).
– Beide haben eine separate DIN-Tastatur mit Cursor-Tasten und verwenden die Maus.
– Beide haben einen großen Grafik-Bildschirmspeicher, beim Mac+ ca. 22K, beim Atari z.Zt. 32K. Der Atari hat damit eine größere Punktauflösung und gestattet außerdem den Anschluß eines Farbmonitors, während der Mac+ bislang ohne Farbe auskommen muß.
– Beide benutzen 3,5-Zoll-Drives, beim Mac+ mit maximal 800K, beim Atari mit maximal 720K (160 Spuren à 9 Blöcken).
– Beide verwenden ein Betriebssystem mit Fenstertechnik, wobei jedoch beim Mac+ eine größere Flexibilität hinsichtlich der Anzahl der geöffneten Fenster besteht.

Aus der *Sicht der Hardware* sind damit beide Geräte praktisch identisch. Erst beim Endabnehmerpreis kommt die Überraschung: Für die Grundausstattung des Mac+ mit eingebautem Monitor und 800K-Drive zahlt man ca. DM 10.000,-, während sich die Grundausstattung des Atari 520+ mit separatem Monitor und 640K-Drive auf nur DM 3.300,- beläuft. Damit ist der Mac+ dreimal so teuer wie der Atari+. In manchen Zeitschriften, so z.B. in „Infowelt", Heft 4/1986 wurde als Grundpreis DM 8750,- genannt, was darauf zurückzuführen ist, daß die Presseunterlagen der Firma Apple die gemäß UWG unzulässigen Nettopreise ohne MwSt aufführten.
Mit der Grundausstattung hat man jedoch noch kein Komplettsystem, das noch unbedingt einen Drucker sowie ein zweites Laufwerk (oder ggf. eine Festplatte) einschließen sollte. Für ein solches Komplettsystem mit Matrixdrucker und Zweitlaufwerk zahlt man dann beim Macintosh ca. DM 13.500,- und beim Atari ca. DM 5.500,-
Aus der *Sicht der Software* ist der erst im Herbst 1985 erschienene Atari noch so nackt wie seinerzeit der 128K-Mac im Herbst 1984. Für den Mac und Mac+ gibt es inzwischen zahlreiche gute Programme, die allerdings zumeist amerikanischer Provenienz und im übrigen oft sehr teuer sind. In der Mikrocomputer-Branche gilt nämlich der Grundsatz:
Der teuren Hardware folgt die teure Software und umgekehrt.
Ein gutes Beispiel hierfür ist der Schneider CPC, der derart preiswert ist, daß man für ihn das Original-dBase für knapp DM 200,- erwerben kann, während es für den Macintosh wahrscheinlich das 8- bis 10fache kosten würde.

4.2. Scimming-Pricing

Die Tatsache, daß die Preise für die 512K-Speichererweiterung für den Macintosh in regelmäßigen Abständen drastisch gesenkt worden sind, läßt sich nicht allein mit den Preisveränderungen im Bauelemente-Sektor erklären, sondern offenbart vielmehr eine neue Salami-Preispolitik der Firma Apple, die in Marketing-Lehrbüchern vornehm als „Abschöpfen" oder weniger vornehm als „Absahnen" („Scimming"; to skim = absahnen) bezeichnet wird: Zunächst kommen die Direktoren im Topmanagement an die Reihe (DM 3750,- für 512K im Herbst 1984), dann die Führungskräfte im mittleren Management (DM 2750,- für 512K im Sommer 1985) und schließlich die Sekretärinnen (ca. DM 1100,- für 1024K ab Februar 1986; exakter Preis inkl. MwSt liegt noch nicht fest).
Wenn man die Preislisten der Firma Apple durchgeht, wird man weitere Beispiele für diese Salami-Taktik finden, jüngst etwa beim Imagewriter II, der im November 1985 für ca. DM 2800,- erschien und bereits im Januar 1986 auf ca. DM 2200,- herabgesetzt wurde.
Peeker-Leser, die den Kauf des Mac+ in Erwägung ziehen, müssen für sich selbst entscheiden, ob sie gleich zugreifen oder noch einige Monate zuwarten wollen, bis sich die Vorstandsvorsitzenden bedient haben.

4.3. Grund- und Zusatznutzen

Daß der Mac+ kein Gerät für „Edel-Freaks" werden wird, dürfte bei einem Grundpreis von DM 10.000,- jedem einleuchten. Andererseits könnte der verbesserte Mac+ eine große Verbreitung in Büros erreichen, wenn das Gerät etwas preiswerter wäre. Leider peilt die Firma Apple in ihrer Werbung vornehmlich die Topmanager an, für die die Ledersessel-Garnitur genauso selbstverständlich ist wie der Porsche Targa als Zweitwagen oder neuerdings eben der Macintosh als

Ein Exemplar der hier gezeigten Bücher

Sie haben die Wahl!

erhalten Sie als "Dankeschön" für einen neuen Abonnenten, den Sie uns vermitteln.

Als »**peeker**«-Leser wissen Sie, wie gut Ihnen dieses Magazin beim Umgang mit Ihrem Apple oder Kompatiblen hilft.

Denn: Wer einen Apple oder einen Kompatiblen hat, der soll auch seinen »**peeker**« haben.

Bestellcoupon

Ich habe den neuen Abonnenten geworben und erhalte kostenlos eines der folgenden Bücher (bitte ankreuzen)

- ☐ Bühler, **Applesoft Basic**
- ☐ Meinhold, **Was ist Elektronik**
- ☐ Schilling, **EDV–kein Geheimnis**
- ☐ Stiehl, **Apple ProDOS Bd. 2**
- ☐ Meinhold, **Was ist Mikroelektronik**

Name, Vorname

Straße, Postfach

PLZ, Ort

X

Datum, Unterschrift

Ich bin der neue Abonnent. Bitte liefern Sie mir bis auf Widerruf, zumindest aber für 1 Jahr, »peeker« zum Jahresbezugspreis von DM 72,– (Ausland plus DM 18,– Porto) an folgende Anschrift:

Name, Vorname

Straße, Postfach

PLZ, Ort

X

Datum, Unterschrift

Gewünschte Zahlungsweise
☐ gegen Rechnung
☐ bargeldlos durch Bankeinzug

Konto-Nr. | Bankleitzahl

Geldinstitut

Vertrauensgarantie:
Diese Bestellung kann ich innerhalb einer Woche bei Dr. Alfred Hüthig Verlag GmbH, Im Weiher 10, 6900 Heidelberg 1 widerrufen. Zur Wahrung der Frist genügt die rechtzeitige Absendung. Ich bestätige die Kenntnisnahme mit meiner Unterschrift:

X

2. Unterschrift

Coupon ausschneiden oder kopieren und einsenden an:

»peeker«
Abonnementservice
Im Weiher 10
6900 Heidelberg 1

Hüthig
PUBLIKATION

Statussymbol auf dem Palisander-Schreibtisch. Ledersessel, Porsche und Teakholz-Schreibtisch werden nicht wegen des Grundnutzens (Sitzen, Fahren, Schreiben), sondern wegen des psychologischen Zusatznutzens gekauft. Dies gilt im übrigen auch für viele Waren des täglichen Lebens wie modische Kleider, Kosmetika usw. Warum sollten also nicht auch bestimmte Mikrocomputer allein wegen des Prestige-Nutzens erworben werden. Hierzu ist der Macintosh bestens geeignet, da er im Gegensatz zu dem recht klobigen IBM-PC keinen gesonderten Schreibtisch erforderlich macht. Wie man weiß, leben Firmen, die sich auf Luxusgüter spezialisiert haben, in der Regel recht gut. Man denke nur an die Porsche AG, deren Vorstandsvorsitzender seit geraumer Zeit in Prospekten, Anzeigen und auf Videobändern die Werbetrommel für die Firma Apple rührt. Unglaubwürdig wird diese Werbung jedoch dann, wenn vom Porsche-Chef in einer Time-Werbebeilage vom 11.11.1985 (Faschingsanfang!) behauptet wird, daß er das Diktaphon abgeschafft habe und nunmehr nicht nur seine private, sondern auch seine Geschäftskorrespondenz auf dem Macintosh selbst tippe. Dies könnte zu einer Revolution der Chef-Sekretärinnen führen, denn diesen verbliebe dann nur noch das rituelle Kaffeekochen sowie als neue Tätigkeit das Massieren der Handgelenke des Chefs zur Vorbeugung gegen Tendovaginitis...
Zurück zur Realität! Es würde der Verbreitung des Mac+ sicher guttun, wenn dieses Gerät mehr über den Preis als über die Prestige-Werbung verkauft werden würde.

4.4. Mac+ und Großrechner

Die 68000-Rechner Mac+, Atari 520+ und Amiga sind weder unter sich noch in bezug auf IBM-PCs vom Betriebssystem her kompatibel. In den nächsten Jahren wird sich ein großer Markt für PCs eröffnen, wenn in Teilbereichen EDV-Terminals durch Mikrocomputer ersetzt werden. Hier haben jedoch nach unserer Beurteilung der Sachlage weder Apple noch Atari noch Commodore irgendeine Chance. Wer eine IBM-Anlage besitzt, wird einen IBM-PC anschließen, und wer eine Siemens-Anlage unterhält, wird zum Siemens-PC greifen. Fremde Mikrocomputer werden hier nur in Spezialfällen zum Tragen kommen. Es wäre an der Zeit, daß die Firma Apple die Hoffnungslosigkeit dieses Unterfangens einsieht.

Betrachten wir hierzu das Service-Problem. Da der Peeker meist als letzte Station und Klagemauer in Sachen Soft- und Hardware-Service fungiert, können wir diesen Komplex wahrscheinlich sogar besser als die Firma Apple selbst beurteilen. Mit der wachsenden Zahl der Nur-Anwender wachsen auch die Service-Probleme der Apple-Händler, denn Service muß heute in einem erheblich erweiterten Sinne verstanden werden:
1. Der sog. reine Hardware-Service bezieht sich auf die Reparatur und/oder den Austausch eines Geräts oder Geräteteils. Beispiel: Auswechseln von defekten Druckköpfen bei Matrixdruckern. Dieser Service wird von allen Apple-Händlern erfüllt.
2. Der sog. reine Software-Service bezieht sich auf die Beseitigung eines Programmfehlers oder Bugs. Beispiel: Änderung des Applewriter-Cursors für Apple-IIc-Besitzer. Dieser Service wird bislang von kaum einem Apple-Händler geboten.
3. Der sog. kombinierte Hard- und Software-Service bezieht sich auf die Anpassung von Apple-Software an fremde Hardware und umgekehrt. Beispiel: Beseitigung von Driver-Problemen eines Apple-Computers in Verbindung mit einem Typenraddrucker. Auch dieser Service wird bislang von kaum einem Apple-Händler geboten.

Beispiel: Ein mir bislang unbekannter Apple-Besitzer, der übrigens kein Peeker-Abonnent war, rief mich vorgestern (30.1.86) zu Hause in aufgelöstem Zustand an: Er sei reiner Anwender, seine Appleworks-Datendiskette, prallvoll mit Daten, sei „defekt", werde nicht mehr von Appleworks akzeptiert, er habe kein Duplikat, sei bei seinem Apple-Vertragshändler, der Firma Rufenach gewesen, das dortige Personal habe einen „Error" festgestellt und ihn dann an den Peeker verwiesen.
Wir haben ihm dann am nächsten Tag in der Redaktion die Datendiskette ausnahmsweise geflickt, obwohl wir natürlich für solche Aufgaben nicht zuständig sind. Lakonischer Kommentar von Herrn Grumser: „Der Mensch war echt erleichtert!"
Der Fall Rufenach ist typisch für viele Apple-Händler. Wohlgemerkt, das Verkaufspersonal der Firma Rufenach ist sehr zuvorkommend und freundlich, aber dies allein genügt heute nicht mehr. Es vergeht bei uns fast kein Tag, an dem sich nicht irgendein Apple-Besitzer über irgendeinen Händler oder über die Firma Apple beschwert. Es ist immer wieder das alte Lied: Weil der Apple-Händler nicht helfen kann, wendet sich der Apple-Besitzer an Apple-München, und Apple-München verweist wieder an den Apple-Händler. Und dann wendet sich der Apple-Besitzer an uns, die wir jedoch meist auch nicht helfen können, weil wir uns nicht eigens für die Beantwortung einer Postkarte die Hardware X oder die Software Y kaufen können.
Hinzu kommt noch, daß die Zahl der Apple-Vertragshändler seit Sommer 1985 drastisch reduziert worden ist, so daß bis Mitte 1986 wahrscheinlich nur noch ca. 160 Händler verbleiben werden, von denen etwa 40 als sog. System-Händler den Macintosh ausliefern werden. Wer dann beispielsweise in Ostfriesland oder im Saarland wohnt, wird möglicherweise 50-100 km fahren müssen, bis er eine „Level-1-Service-Station" findet.
Wir verstehen in gewisser Hinsicht, daß sich Apple-Händler nur für den Hardware-Service im engeren Sinne zuständig fühlen und kein Interesse daran haben, etwa einem Appleworks-Käufer bei einem Spezialproblem zu helfen. Schließlich, so könnte man argumentieren, ist Appleworks nicht von Apple, sondern von einem Mr. Rupert Lissner entwickelt worden, und im übrigen gibt es ja noch diese Freak-Zeitschrift – wie war doch der Name – ach ja diesen Peeker, an den man sich bei technischen Problemen wenden kann. Bei dieser Einstellung darf man sich jedoch dann nicht wundern, wenn kaum jemand bereit ist, den Macintosh in Verbindung mit einer IBM-Anlage einzusetzen, denn gegen den Dschungel der IBM-Datenprotokolle ist das Flicken einer Datendiskette nun wirklich ein kleiner Fisch. Und wenn dann der Macintosh tatsächlich einmal am Tropf einer IBM-Anlage hängen sollte, wird sich der Mac-Besitzer ganz allein helfen müssen. Denn wer bereits bei kleinen Apple-II-Problemen überfordert ist, wird bei IBM-Problemen hoffnungslos resignieren.

5. Fazit

Wir glauben, daß sich der Mac+ zu einem soliden und brauchbaren Mikrocomputer gemausert hat und daß er gute Verkaufschancen hätte, wenn er halb soviel kosten würde. Wir glauben jedoch nicht, daß es dem Mac+ oder irgendeinem anderen IBM-fremden Mikrocomputer gelingen wird, als intelligenter Terminal an einen Mainframe-Computer angeschlossen zu werden. Dieser Markt wird den kombinierten EDV- und PC-Anbietern vorbehalten bleiben. Aber der Einsatz eines Mac+ als Stand-alone-Gerät stellt schließlich auch einen Markt dar. Neben der Verwendung als Manager-Werkzeug könnte er bei angemessenem Preis in den Bürobereich schlechthin eindringen. Außerdem wird er in Verbindung mit dem Laserwriter neue Bereiche in den traditionellen Hausdruckereien von Industriefirmen erschließen können. Aufgrund von Unterlagen des Satzanlagenherstellers Compugraphic wird hierzu von uns ein mehrteiliger Bericht vorbereitet.

Im folgenden wird untersucht, wie reelle Zahlen von Applesoft behandelt werden. Wir werden unter anderem der Frage der Genauigkeit nachgehen und auch zeigen, warum Applesoft, im Gegensatz zu vielen Taschenrechnern, nur Zahlen aus dem Bereich von 10 ↑ (-38) bis 10 ↑ 38 verarbeiten kann.
Da dieser Artikel sowohl für Applesoft-Neulinge als auch für mathematisch wenig bewanderte Leser verständlich bleiben soll, sind einige vorbereitende Bemerkungen unerläßlich.

Wenn Sie wissen möchten, warum PRINT 1.01-1.0 nicht 0.01 ergibt, dann lesen Sie den folgenden Aufsatz!

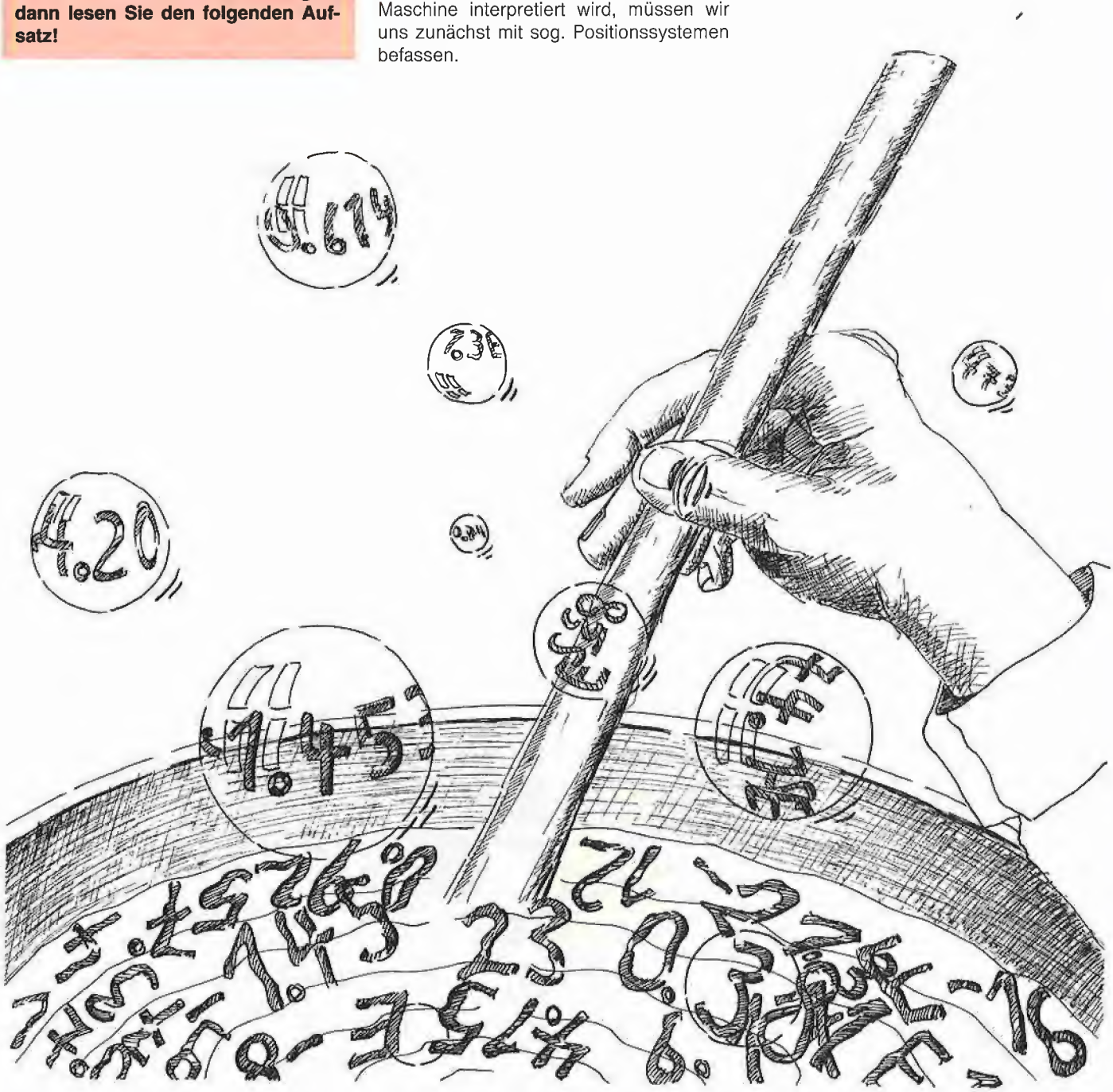
1 Eine exakte Definition des Begriffs der „reellen Zahl" ist nur möglich, wenn entsprechende mathematische Kenntnisse vorausgesetzt werden dürfen. Wir wollen hier unter „reellen Zahlen" einfach Dezimalbrüche jeglicher Art verstehen.

2 Die CPU des Apple ist ein 8-Bit-Prozessor. Sowohl RAM- als auch ROM-Speicher sind daher folgerichtig als 8-Bit-Speicher ausgelegt.

3 Jedes Bit kann einen der beiden Zustände „on" (Wert 1) und „off" (Wert 0) annehmen.

4 Um verstehen zu können, wie der Inhalt eines Speichers (8 Bits = 1 Byte) von der Maschine interpretiert wird, müssen wir uns zunächst mit sog. Positionssystemen befassen.

Wir sind von früher Kindheit an so sehr daran gewöhnt, „dezimal" zu denken, daß uns, wenn wir etwa die Zahl 234 sehen, kaum noch bewußt ist, daß es sich hierbei um eine durch Konvention festgelegte Kurzschreibweise für den Ausdruck 2 ∗ 10 ↑ 2 + 3 ∗ 10 ↑ 1 + 4 ∗ 10 ↑ 0 handelt. Jeder Ziffer ist also neben ihrem Wert noch ein Stellenwert zugeordnet. In dem uns vertrauten Dezimalsystem sind dies die Potenzen der Basis 10.



Ein System, das sich dieser Art der Zahlennotation bedient, wird **Positionssystem** genannt. Ein Gegenbeispiel ist die römische Art der Zahlenschreibweise.
Positionssysteme haben unter anderem den Vorzug, daß man nur eine geringe Anzahl von Ziffern braucht (in unserem Falle die Ziffern von 0 bis 9), um alle denkbaren Zahlen in verhältnismäßig kurzer Form schreiben zu können.
Daß sich gerade die Zahl 10 als Basis

1011011110110110 (47030).
Um diese Zahl abzuspeichern, brauchen wir offensichtlich zwei Speicherplätze. Wir legen das aus den letzten 8 Bits bestehende Byte (182) in einem Speicher A, und das aus den ersten 8 Bits bestehende (183) im darauffolgenden A + 1 ab. Da die Stellenwerte des vorderen Bytes 256mal so groß sind wie die entsprechenden des hinteren, wird die Bedeutung des in BASIC-Programmen häufig auftretenden

Muster zu speichern vermag, festgehalten?
Machen wir zuerst ein kleines Experiment. Wir schalten den Apple bei geöffneter Drive-Tür ein und drücken Reset. Die Maschine ist bereit, aber DOS ist nicht geladen. Wir tippen nun:

AA = 17.35
Mit
CALL -151

Behandlung reeller Zahlen in Applesoft

von Konrad Steinmeier

etabliert hat, beruht aller Wahrscheinlichkeit nach auf der Tatsache, daß wir 10 Finger besitzen und der Mensch von früh auf beim Zählen die Finger benutzte. Hätte uns die Natur mit 8 Fingern an jeder Hand ausgestattet, dann wäre unser Denken vermutlich „hexadezimal" ausgerichtet, was dem Computer-Anwender manches Kopfzerbrechen ersparen würde.
Zurück zu Bit und Byte! Da ein Bit nur einen von zwei Zuständen annehmen kann, stehen dem Computer in gewissem Sinne nur zwei „Ziffern" (0 und 1) zur Verfügung. Wir sind somit gezwungen, das uns vertraute Dezimalsystem zu verlassen und uns einem Positionssystem zuzuwenden, das mit eben diesen beiden Ziffern auskommt.
Es dürfte klar sein, daß die Basis des neuen Systems die Zahl 2 sein muß. Damit befinden wir uns im Dual- oder Binärsystem.
Was bedeutet nun beispielsweise ein Speicherinhalt der Form 10110110? Wenn wir die einzelnen Bits (Ziffern) mit ihren Stellenwerten (Potenzen von 2) versehen, kommen wir zu dem in **Tabelle 1** dargestellten Ergebnis.

5 Noch ein paar Worte zur Art der Abspeicherung natürlicher Zahlen (insbesondere Speicheradressen). Die größte durch ein Byte darstellbare Zahl ist (binär) 11111111, dezimal also 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255. Will man eine Zahl abspeichern, die größer ist, dann braucht man folglich mehr als ein Byte.
Nehmen wir als Beispiel

Ausdrucks PEEK (A) + 256 * PEEK (A + 1) klar. In unserem Beispiel ergäbe das 182 + 256 * 183 = 47030.
Die größte Zahl, die in dieser Weise gespeichert werden kann, ist 255 + 256 * 255 = 257 * 255 = 65535, und dies ist somit auch die größte Speicheradresse. Wie die Adressen werden z. B. auch die Zeilennummern eines Applesoft-Programms in 2-Byte-Form gespeichert.

Zum Schluß dieses Abschnitts noch ein Wort zu der zunächst sinnwidrig erscheinenden Konvention, die beiden Bytes in umgekehrter Reihenfolge abzuspeichern. Der Prozessor des Apple führt im Grunde als einzige Rechenoperation die Addition aus. Subtraktionen werden durch Addition des sog. Basiskomplements des Subtrahenden simuliert (nach einer Idee übrigens, die von Gottfried Wilhelm Leibniz [1646-1716] stammt!).
Will man nun zwei 2-Byte-Zahlen addieren, so hat man zuerst die beiden hinteren Bytes zu addieren, und danach die beiden vorderen zusammen mit dem sich aus der ersten Addition ergebenden Übertrag. Durch die Art der Abspeicherung findet die CPU die zu verarbeitenden Bytes in der Reihenfolge vor, wie sie zu verarbeiten sind.

1. Ein kleines Experiment

Wenden wir uns nun unserem eigentlichen Thema zu. In welcher Form wird eine reelle Zahl von Applesoft, das doch nur natürliche Zahlen von 0 bis 255 als Bit-

gehen wir danach in den Monitor und tippen:
69.6A
Auf dem Schirm erscheint:

```
0069- 03 08
```

Die beiden Bytes hinter dem Strich sagen uns, daß die Speicherung der numerischen Variablen bei $0803 (LOMEM) beginnt. Nun tippen wir:
803.809
und sehen:

```
0803- 41 41 85 0A CC
0808- CC CD
```

Die ersten beiden Bytes stellen den Namen der Variablen AA dar, wobei 41 der ASCII-Code des Buchstaben A in hexadezimaler Form ist. Es folgen ab Speicher $0805 fünf Bytes, die den der Variablen AA zugewiesenen Wert repräsentieren.
Die fünf Bytes, die die Zahl 17.35 repräsentieren (im folgenden wird der der englischen Notation entsprechende Dezimalpunkt anstelle des Kommas benutzt), befinden sich in dem Speicher $0805 bis $0809. Um sie in die uns vertraute dezimale Form umzuwandeln, sehen wir erst einmal nach, wie die Speicheradressen in dezimaler Form lauten. Hier hilft uns eine Routine des Applesoft-Interpreters (LINPRT), die diese Umwandlung für uns erledigt. Wir tippen:
45:08 05 NED24G
Was da im einzelnen vor sich geht, soll hier nicht weiter besprochen werden. Begnügen wir uns mit dem angezeigten Ergebnis 2053.

Wir kehren mit Ctrl-C Return in den BASIC-Editor zurück und finden schließlich die gesuchten fünf Zahlen, indem wir folgende Zeile eingeben:
FOR I = 2053 TO 2057: ? PEEK (I)" ";: NEXT
Auf dem Schirm erscheint:

133 10 204 204 205

Und das endlich ist das Ergebnis unseres Experiments. Die Zahl 17.35 wird also von Applesoft in das Quintupel aus den Zahlen 133, 10, 204, 204 und 205 umgewandelt.

2. Die interne Darstellung

Zwei Fragen bleiben noch zu klären: Wie wird die Umwandlung einer reellen Zahl in ein Quintupel vollzogen, und wie wird aus einem Quintupel wieder eine reelle Zahl generiert? Beide Prozesse beanspruchen jeweils sechs Schritte, wobei natürlich im zweiten Falle die gleichen Schritte vollzogen werden wie im ersten, nur in umgekehrter Reihenfolge. Die einzelnen Schritte sind:

I. Umwandlung der reellen Zahl in einen Binärbruch,
II. Normierung, d.h. Aufspaltung in Mantisse und Exponent,
III. Voranstellen des um 128 vergrößerten Exponenten,
IV. Aufteilung der Mantisse in Achtergruppen,
V. Setzen des Vorzeichen-Bits,
VI. Umwandlung der Achtergruppen in natürliche Zahlen.

2.1. Dezimal nach Binär

Wir wollen die sechs Schritte der Umwandlung am Beispiel der Zahl 17.35 in aller Ausführlichkeit vollziehen und erklären. Da es aber dazu offensichtlich notwendig ist, Dezimalzahlen in Binärzahlen und Binärzahlen in Dezimalzahlen umzuwandeln, sollen zuerst Algorithmen entwickelt werden, die zur Lösung dieser Aufgaben geeignet sind.

Schritt I: Sehen wir uns noch einmal die uns schon bekannte binäre Darstellung der Zahl 183 an. Wenn wir die rechte Seite sukzessive durch die Basis 2 dividieren, dann erhalten wir das in **Tabelle 2** dargestellte Schema.
Es ist sicher nicht zu übersehen, daß es sich bei den auftretenden Resten genau um die Ziffern der dualen Darstellung handelt, die allerdings in umgekehrter Reihenfolge auftreten. Man erkennt, wie sie durch die Divisionen der Reihe nach freigelegt werden.
Soweit wir wissen, stammt das Verfahren von dem griechischen Mathematiker Euklid und ist daher als euklidischer Algorithmus bekannt. Seine Umsetzung in ein BASIC-Programm finden wir in **REELL.1**. Wenden wir es doch noch einmal auf eine andere Zahl an! Diesmal in etwas vereinfachter Schreibweise und mit der Basis 3:

```
217 = 3 * 72 + 1 (Stellenwert:  1)
 72 = 3 * 24 + 0 (Stellenwert:  3)
 24 = 3 *  8 + 0 (Stellenwert:  9)
  8 = 3 *  2 + 2 (Stellenwert: 27)
  2 = 3 *  0 + 2 (Stellenwert: 81)
```

In der Tat ist 2 * 81 + 2 * 27 + 0 * 9 + 0 * 3 + 1 * 1 = 217. Der euklidische Algorithmus funktioniert also in gleicher Weise für jede beliebige Basis (**REELL.2**). Ist allerdings die Basis größer als 10, dann kommen wir mit den Ziffern 0 bis 9 nicht mehr aus (**REELL.3**).
In den Programmen REELL.1 bis REELL.3 wird die Applesoft-Darstellung reeller Zahlen verwendet. Die Ergebnisse sind dadurch bezüglich ihrer Exaktheit gewissen Beschränkungen unterworfen. Das gilt nicht für **REELL.1A**. Hier werden Zahlen mit bis zu 10 Ziffern in voller Genauigkeit in Binärzahlen umgewandelt. Dafür müssen allerdings größerer Programmieraufwand und längere Laufzeit in Kauf genommen werden. Die 10-Ziffern-Begrenzung läßt sich selbstverständlich durch entsprechende Dimensionierung in Zeile 160 nach Belieben abändern.
Zur Umwandlung einer in einem gewissen Positionssystem vorliegenden Zahl in dezimale Notation sehen wir uns **REELL.4** an. Nachdem Basis und Zahl eingegeben worden sind (Zeile 150-160), werden die Ziffern mit ihren Stellenwerten (Potenzen der Basis) multipliziert und die so entstehenden Produkte addiert (Zeile 180-220). Das Ergebnis ist die Zahl in dezimaler Form.
Das Problem der Darstellung einer natürlichen Zahl in einem beliebigen Positionssystem ist damit gelöst. Da wir es aber mit Dezimalbrüchen zu tun haben, müssen wir uns jetzt mit der Frage ihrer Umwandlung in Dualbrüche befassen. Sie wird durch **REELL.5** beantwortet.
In Zeile 170 wird zunächst das Vorzeichen festgehalten. In den Zeilen 200 und 210 wird dann die unserer Zahl benachbarte Potenz von 2 bestimmt. Ist unsere Zahl kleiner als diese Potenz, dann notieren wir die Ziffer 0 (Zeile 220); ist sie nicht kleiner, dann ist die zu notierende Ziffer 1, und die Zahl wird um die Zweierpotenz vermindert (Zeile 230). Nun wird der Exponent um 1 vermindert und der Vorgang solange wiederholt, bis wir die für unsere Zwecke erforderliche Anzahl von Ziffern besitzen (240-250). Die Position des Binärpunktes wird in der Variablen BP festgehalten.
Zum besseren Verständnis wollen wir die Ergebnisse der ersten Schritte für unser Beispiel 17.35 aufschreiben (**Tabelle 3**).
In dieser Weise erhalten wir den folgenden Binärbruch:

10001.0101100110011001100110100.

Damit ist Schritt I erledigt.

Schritt II: Wir erinnern uns, daß die Multiplikation eines Dezimalbruchs mit 10 bzw. seine Division durch 10 sich in einer entsprechenden Verschiebung des Dezimalpunkts ausdrückt. Es ist etwa 123.456 = 1.23456 * 10 ↑ 2 = 1234.56 * 10 ↑ (-1). Entsprechendes gilt für Dualzahlen, bei Multiplikation mit 2 bzw. Division durch 2.
Unter Normierung versteht man die Darstellung des Binärbruchs in der Form
0.1XXXXXXXXX * 10 ↑ Y,
die durch eine entsprechende Verschiebung des Binärpunktes immer erreichbar ist, es sei denn, der Bruch enthalte keine einzige Ziffer 1. Mit Ausnahme der Null, die wie so oft eine Sonderstellung einnimmt (sie wird durch den sonst nicht auftretenden Exponenten 0 repräsentiert), kann man somit jede Zahl normieren. 1XXXXXXXXX heißt Mantisse, Y Exponent der normierten Form.
Unsere Zahl hat jetzt die Gestalt

0.10001010110011001100110011001101 00
* 10 ↑ 00000101

(10 ↑ 0000101 ist in dezimaler Notation natürlich 2 ↑ 5!).

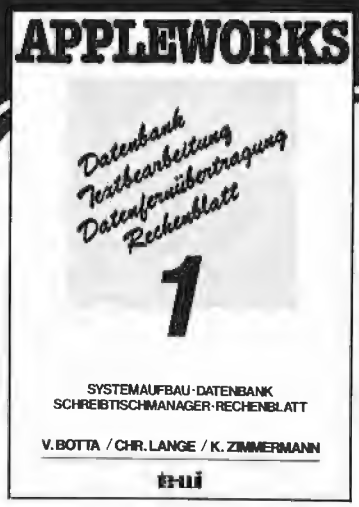
Schritt III: Offensichtlich wird der Exponent um 128 vergrößert, wenn man sein erstes Bit von 0 in 1 umwandelt. Stellen wir ihn an den Anfang, so erhalten wir

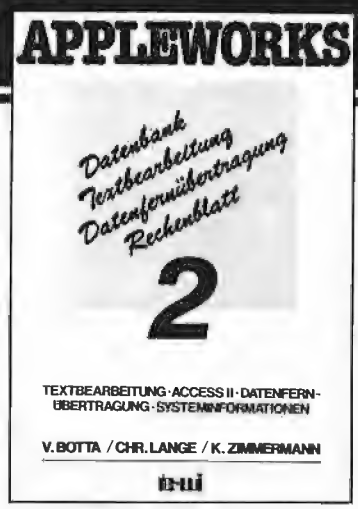
10000101

10001010110011001100110011001101 00.

Die Vergrößerung um 128 wird natürlich vorgenommen, um negative Exponenten, die ja keine natürlichen Zahlen sind, auszuschließen. Das ist aber nur dann erreichbar, wenn der Exponent vor der Vergrößerung nicht kleiner war als -127. Aus dem Wert -128 ergäbe sich nämlich nach der Addition 0, wodurch aber, wie oben erwähnt, die Zahl Null dargestellt wird. Andererseits darf der Exponent +127 nicht überschreiten, da sich sonst nach der Vergrößerung eine Zahl ergäbe, die nicht mehr in einem Byte unterzubringen ist.
Es ist jetzt nicht mehr schwer, die größte in dieser Weise darstellbare Zahl ausfindig zu machen. Wir kehren dazu zu unserem Experiment zurück. Wir tippen:
FOR I = 2053 TO 2057: POKE I,255: NEXT
womit wir den der Variablen AA zugeordneten Zahlen den größtmöglichen Wert zuweisen. Nach PRINT A erscheint auf dem Schirm

-1.70141183E+38





Band 1:
1. Einleitung
2. Was Sie benötigen
3. Starten von APPLE WORKS
4. Der Schreibtischmanager
5. Datenbank
6. Rechenblatt

A1 Anschluß der Festplatte ProFile
A2 APPLE II Easy Pieces Referenz
A3 Druckeranpassungen
A4 DOS 3.3 Konvertierungen
A5 APPLE WORKS Disketten sichern/kopieren
A6 Hilfsfunktionen nach Programmteilen

Band 2:
1. Einleitung
2. Was Sie benötigen
3. Starten von APPLE WORKS
4. Der Schreibtischmanager
5. Textbearbeitung
6. Datenfernübertragung

A1... A6 wie Band 1, dazu:
A7 Modemkabel für APPLE IIe, IIc
A8 SuperSerialCard: Einstellung
A9 ASCII-Textdateien aus anderen Dateiformaten für ACCESS II
A10 DATEX-P20 F Verzeichnis
A11 Deutsche/Englische Menübilder von ACCESS II

APPLE WORKS auf APPLE II, IIe, IIc:
verwandelt APPLE-II-Computer in einen Elektronischen Schreibtischmanager mit:
Texterstellung ...Edition, Briefarchiv, Ausdruck etc.
Datenarchivierung ...Kontoführung, Buchhaltung etc. Formblattkalkulation ...Bilanzen, VisiCalc-Dateien etc. Datenfernübertragung ...Mailbox, Rechnerkopplung etc.

- ist ein erfolgreicheres Integrationspaket als LOTUS auf IBM PC!
- ist auf 1 MByte Speichererweiterungen Ihres APPLE II vorbereitet!
- erschließt Ihnen die Zukunftstechnik MAILBOX!
- ist ebenso einfach zu bedienen wie APPLE WRITER:
  Kein Befehlsstudium ...Einfachste Menüführung ... Sofortige Anwendbarkeit

te-wi's APPLE WORKS SYSTEMBÜCHER 1+2 zeigen Ihnen:
- Sämtliche APPLE WORKS Funktionen an Beispielen aus der Wirtschaft
- Das Wechseln zwischen Text/Rechenblatt/Datenarchiv/Dfü
- Umfassende Systeminformationen zu Dateikonvertierung, Druckeranpassung etc.

Von Botta/Lange/Zimmermann je 264 Seiten, Softcover, je DM 49,–

te-wi Verlag GmbH
Theo-Prosel-Weg 1
8000 München 40

Weitere te-wi-Bücher

Das APPLE II/II+/IIe/IIc-Handbuch NEU
(L. Poole)
Erst mit Hilfe dieses Leitfadens werden Sie Ihren Apple II erfolgreich einsetzen, denn Text und Bildmaterial gehen weit über das hinaus, was herstellerseitig an Literatur angeboten wird.
Neu überarbeitet und jetzt um die spezifischen Eigenheiten der Modelle **II e** und **II c** erweitert. 472 Seiten, Softcover, DM 66,–

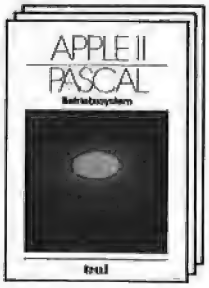
LOGO – Jeder kann programmieren
(Daniel Watt)
Buch des Jahres in den U SA. Für die Computer APPLE II, C-64, IBM PC, ATARI bis 520 ST, TI-99 und Schneider CPCs.
Hochwertiges Textbuch für Logo-Kurse zu Hause und im Lehrbereich.
384 Seiten, A4, DM 59,–

NEU

APPLE II – Bewegte 3D-Graphik
(Phil Cohen)
Selbstentworfene Graphiken und Diagramme – animiert oder als Standbilder – eben oder räumlich: alle erforderlichen BASIC-Programme mit Erklärung finden Sie in diesem Buch.
200 Seiten, Softcover, DM 49,–

Apple Maschinensprache
Für BASIC-Programmierer der einfachste Zugang zur Muttersprache des APPLE. Wesentlich schnellere Maschinenprogramme, direkte Manipulation des Mikroprozessors 6502 im APPLE – als Brücke dorthin benötigt dieses Buch nur die drei BASIC-Befehle POKE, CALL, PEEK. D. Inman/K. Inman, DM 49,–

Reparaturanleitung Computer: Apple II, IIplus NEU
Einzigartige Serviceunterlage für Reparaturen und Entwicklungsarbeiten am Apple II. Enthält Schaltpläne, Bauteile- und Vergleichstypenliste; Prüfpunkte mit Oszillogrammen der Signalformen, Logiktabellen, Spannungsangaben; schnelle Servicetests; Anleitung zur systematischen Fehlersuche. In A4-Mappe, DM 29,80



Erstes deutsches Referenzwerk sämtlicher Befehle und Systemroutinen von Apple II, IIplus, IIe
APPLE II PASCAL NEU
Betriebssystem, 272 S., DM 49,–
Sprache, 216 S., DM 39,–
Pascal 1.2 Addendum, 112 S., DM 36,–

Grundlagenbuch, Bestseller
APPLE II PASCAL,
Eine praktische Anleitung,
544 S., DM 59,–

tm 4422

Noch im Programm:
Computer für Kinder, APPLE II, DM 29,80
6502 Programmieren in Assembler, DM 59,–
Umweltdynamik (Prospekt anfordern), DM 66,– (NEU)

Macintosh Programmierhandbuch mit MSBASIC 2.0 (Ende '85), DM 59,–
Einführung in die Mikrocomputer-Technik, DM 66,–
M68000-Familie, 2 Bände, DM 79,– und DM 69,–

(Stört Sie das Minuszeichen? Mit POKE 2054, 127 kann es entfernt werden.) Die obere Grenze für von Applesoft verarbeitbare Zahlen ist also nicht 10 ↑ 38, sondern 1.70141183 * 10 ↑ 38. In entsprechender Weise findet man die untere Grenze 2.93873588 * 10 ↑ (-39).

Schritt IV: Die Aufteilung in Achtergruppen liefert uns die Darstellung:

10000101

10001010 11001100 11001100 11001101.

Schritt V: Der Normierungsprozeß bringt es mit sich, daß das erste Bit des zweiten Bytes immer gesetzt (= 1) ist. Es bedeutet also keinen Verlust an Information, wenn man dieses Bit nicht kennt. Man kann es daher für andere Zwecke verwenden und tut dies auch, indem man ihm die Rolle des Vorzeichen-Bits zuweist. Nach Vereinbarung beläßt man ihm den Wert 1, falls eine negative Zahl vorliegt. Ist sein Wert 0, dann handelt es sich um eine positive Zahl.

Aus unserer Zahl wird dann

10000101

00001010 11001100 11001100 11001101.

Schritt VI: Zum guten Ende erhalten wir mit Hilfe des Programms REELL.4 die gesuchten Zahlen 133, 10, 204, 204, 205.

Durch **REELL.6** wird ein Programm vorgelegt, das den gesamten Umwandlungsprozeß für beliebige positive und negative reelle Zahlen vollzieht und die Zwischenergebnisse auf den Bildschirm bringt. Das Programm setzt das Vorhandensein einer 80-Zeichenkarte voraus, da anderenfalls nicht alles auf dem Schirm unterzubringen ist.

Es soll nicht verschwiegen werden, daß das Programm seine Grenzen hat. Da nur 73 Binärziffern berechnet werden (Zeile 360), wird BP nur gefunden, wenn die umzuwandelnde Zahl kleiner ist als 2 ↑ 73 (ungefähr 10 ↑ 22). Anderenfalls wird dadurch ein falscher Exponent berechnet.

Zur korrekten Umwandlung braucht man 32 gültige Ziffern (d.h. ohne führende Nullen). Diese werden bis zur Größenordnung 10 ↑ (-12) gefunden, und somit werden richtige Ergebnisse geliefert. Bei 10 ↑ (-13) ist das nicht mehr der Fall, wodurch das letzte Byte ungenau wird.

Eingaben, die kleiner als 10 ↑ (-22) sind, würden zu „BAD SUBSCRIPT IN 690" führen, da I + 31 größer als 100 wird (Zeile 740). Ist schließlich die Eingabe größer als 10 ↑ 37, dann folgt „OVERFLOW IN 310". Die ONERR-GOTO-Anweisung in Zeile 180 verhindert das Abbrechen des Programms in diesen Fällen und fordert zu erneuter Eingabe auf.

2.2. Binär nach Dezimal

Es bleibt die Frage zu klären, wie die Umwandlung eines Quintupels in eine reelle Zahl vollzogen wird. Wie oben gesagt, sind an sich die sechs Schritte in umgekehrter Reihenfolge zu vollziehen. Da wir aber nun die Bedeutung der einzelnen Glieder des Quintupels kennen, können wir mit weit geringerem Programmieraufwand auskommen. Sehen wir uns **REELL.7** an!

Zunächst werden die fünf Zahlen eingegeben, wobei unzulässige Eingaben zurückgewiesen werden (160-190). Dann wird das Vorzeichen bestimmt und das zweite Byte vom Vorzeichen-Bit befreit (210-220). Anschließend addieren wir die mit ihren Stellenwerten versehenen vier letzten Bytes (240-260) und multiplizieren das

Ergebnis mit der durch das Exponent-Byte bestimmten Zweierpotenz (280-290). Man könnte wohl meinen, die beiden Zeilen seien durch D = D * 2 ↑ (Z(1) - 128) ersetzbar. Dies würde aber bei Z(1) = 255 zur OVERFLOW-Meldung führen. Zeile 270 sichert die richtige Behandlung von Zahlen, deren Exponent-Byte den Wert 1 besitzt.

Als kleine Anwendung ist **REELL.8** gedacht, wo sich der Algorithmus aus REELL.7 in den Zeilen 220-250 wiederfindet.
Zur Beschleunigung des Programmablaufs sind im ROM des Apple eine Reihe von oft gebrauchten Zahlen fest gespeichert. Zum Beispiel steht in dem Speicherbereich 61552-61556 das Quintupel für die Zahl 1/4. Das Programm REELL.8 berechnet diese Applesoft-Konstanten und bringt Ergebnis und zugehörigen Adreßbereich auf den Bildschirm.

Tabelle 1

```
  1*2↑7 + 0*2↑6 + 1*2↑5 + 1*2↑4 + 0*2↑3 + 1*2↑2 + 1*2↑1 + 0*2↑0
= 1*128 + 0* 64 + 1* 32 + 1* 16 + 0*  8 + 1*  4 + 1*  2 + 0*  1 = 182.
```

Tabelle 2

```
10110111 = 2*2*2*2*2*2*2 + 2*2*2*2*2*2 + 2*2*2*2*2 + 2*2*2 + 2 + 1
           2*2*2*2*2*2   + 2*2*2*2*2   + 2*2*2*2   + 2   + 1     Rest: 1
           2*2*2*2*2     + 2*2*2*2     + 2*2*2     + 1         Rest: 1
           2*2*2*2       + 2*2*2       + 2*2       + 1         Rest: 1
           2*2*2         + 2*2         + 2                     Rest: 0
           2*2*2         + 2           + 1                     Rest: 1
           2*2           + 1                                   Rest: 1
           2                                                   Rest: 0
           1                                                   Rest: 1
           0
```

Tabelle 3

```
17.35         > = 2↑4            17.35 - 2↑4     = 1.35          Ziffer: 1
 1.35         <   2↑3                                            Ziffer: 0
 1.35         <   2↑2                                            Ziffer: 0
 1.35         <   2↑1                                            Ziffer: 0
 1.35         > = 2↑0             1.35 - 2↑0     = 0.35          Ziffer: 1
 0.35         <   2↑(-1)                                         Ziffer: 0
 0.35         > = 2↑(-2)          0.35 - 2↑(-2) = 0.100000001    Ziffer: 1
 0.100000001 <   2↑(-3)                                         Ziffer: 0
 0.100000001 > = 2↑(-4)   0.100000001 - 2↑(-4) = 0.0375000015  Ziffer: 1
```

3. Die Genauigkeit

Zum Schluß noch einige Bemerkungen über die Genauigkeit, mit der reelle Zahlen in Applesoft behandelt werden. Die Ergebnisse unserer bisherigen Untersuchungen machen es uns leicht, eine grobe Abschätzung vorzunehmen.
Zur Darstellung der Mantisse werden 32 Bits benötigt. Die größte darstellbare Mantisse ist also

11111111 11111111 11111111 11111111.

Fassen wir sie für den Augenblick als natürliche Zahl auf, dann ist ihr Wert in dezimaler Form 2 ↑ 32 - 1, also ungefähr 4.3 * 10 ↑ 9. Das ist eine zehnstellige Zahl. Dies bedeutet aber, daß Zahlen mit mehr als zehn Stellen nicht mehr in voller Genauigkeit dargestellt werden können.

Um die Darstellung von Dezimalbrüchen zu untersuchen, müssen wir zuerst ihre Entstehung als Ergebnis einer Division zweier natürlicher Zahlen betrachten.
Hat der Divisor von 2 und 5 (Faktoren der Basis 10) verschiedene Primfaktoren, dann erhalten wir stets einen periodischen Dezimalbruch, der also mit endlich vielen Ziffern nicht exakt darstellbar ist. Um das einzusehen, betrachten wir z.B. den Bruch 13/40. Durch Erweiterung mit 25 wird daraus 325/1000 = 0.325. Eine solche Erweiterung ist aber etwa bei 13/60 nicht mehr möglich, da der Nenner den Faktor 3 enthält, der in keiner Potenz von 10 vorkommt.
Entsprechendes gilt für Brüche aller Positionssysteme. Da im Dualsystem nur 2 als Primfaktor der Basis auftritt, erhalten wir fast in allen Fällen periodische Ergebnisse. Abbrechende Binärbrüche entstehen nur, wenn der Divisor eine Potenz von 2 ist. Dies macht sich zwar nicht immer bemerkbar, heißt aber doch, daß „fast kein" Dezimalbruch im Applesoft-Format exakt darstellbar ist. Lassen Sie doch einmal das folgende Programm für verschiedene Schrittlängen laufen!

```
10 HOME : INPUT "SCHRITTLAENGE: ";J :
   REM kleiner als 1
20 FOR I = 0 TO 99 STEP J
30 IF PEEK (1801 + (J < 1 / 64)) < >
   160 THEN STOP
40 PRINT I: NEXT
```

Zeile 30 hält das Programm an, sobald in der siebten Bildschirmzeile ein ungenaues Ergebnis steht. Sie werden feststellen, daß das Programm fast immer nach wenigen Schritten anhält. Nur bei den Schrittlängen 0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625 und 0.0078125 ist dies nicht der Fall. Warum? Es sind die Stammbrüche mit Zweierpotenzen als Nenner.
Die geschilderte Tatsache ist auch der Grund für das gelegentliche Versagen primitiver Trainingsprogramme nach Art von **REELL.9**. Bei den in den REM-Zeilen angegebenen Beispielen versagt das Programm, obwohl berechneter und geratener Wert als (scheinbar) gleich ausgegeben werden. (0.7 z.B. wird durch das Quintupel 128, 51, 51, 51, 51, falls es aber als X + Y berechnet wird, durch 128, 51, 51, 51, 52 dargestellt.)
Das Versagen des Programms beruht auf Ungenauigkeiten in der Bestimmung der letzten Bits von Byte 5. Durch Entfernung des REM-Tokens in Zeile 180 (so daß die angegebene Berechnung ausgeführt wird) kann der Fehler korrigiert werden.
Man könnte natürlich voraussagen, für welche Zahlenkombinationen dieser für den Unkundigen verwirrende Umstand eintritt. Dies bedürfte aber einer umfassenden Analyse, die wir uns hier aus Platzgründen versagen müssen. Der interessierte Leser kann mit Hilfe von **REELL.10** weiterforschen.
REELL.10 ist eine Weiterentwicklung von REELL.7. Nachdem die fünf Zahlen eingegeben sind, erscheint in der Mitte des Bildschirms Byte 5 in binärer Darstellung. Unten wird der durch die fünf Zahlen repräsentierte Dezimalbruch angezeigt.
Nach den Anweisungen am oberen Bildschirmrand kann nun jedes Bit von Byte 5 nach Belieben geändert und die Auswirkung der Änderung auf die dargestellte Zahl beobachtet werden.
Versuchen Sie es doch einmal mit dem Quintupel 80, 0, 0, 0, 0! Eine Änderung von Bit 3 wird hier bereits nicht mehr wahrgenommen. Bei 40, 0, 0, 0, 0 beeinflußt selbst eine Änderung von Bit 4 die Darstellung nicht!

Demo-Programme

Die Peeker-Sammeldisk enthält die Demos REELL.1, REELL.1A, REELL.2 bis REELL.10 als Applesoft-Programme, die jeweils mit RUN gestartet werden und unter DOS 3.3 und ProDOS lauffähig sind.
Mit Ausnahme von REELL.6 sollte die 40-Z/Z-Darstellung gewählt werden.

REELL.1

```
100 REM +--------------------+
110 REM !                    !
120 REM ! Euklid für Basis 2 !
130 REM !                    !
140 REM +--------------------+
150 DIM S(39): HOME
160 INPUT "WELCHE ZAHL? ";Z: REM Höchstens 12 Ziffern
170 S = INT (Z / 2)
180 S(I) = Z - 2 * S: REM Rest
190 Z = S: IF Z = 0 THEN 210
200 I = I + 1: GOTO 170
210 PRINT : FOR J = I TO 0 STEP - 1
220 PRINT S(J);: NEXT
```

REELL.1A

```
100 REM +--------------------+
110 REM !                    !
120 REM ! Euklid für B = 2   !
130 REM !      (exakt)       !
140 REM !                    !
150 REM +--------------------+
160 D1 = 11:D2 = 34: DIM S(D1),T(D2): REM Maximal 10 Ziffern
170 HOME : INPUT "WELCHE ZAHL? ";Z$:L = LEN (Z$): IF L > D1
    - 1 THEN 170
180 FOR I = 1 TO L:S(I) = VAL ( MID$ (Z$,I,1)): NEXT
190 K = K + 1: FOR I = 1 + P TO L:T = S(I)
200 S(I) = INT (S(I) / 2)
210 IF T > 2 * S(I) THEN S(I + 1) = S(I + 1) + 10
220 NEXT : IF S(L + 1) > 0 THEN T(K) = 1:S(L + 1) = 0
230 IF S(1 + P) = 0 THEN P = P + 1
240 IF L > P THEN 190
250 PRINT : PRINT Z$: PRINT "= %";:I = D2
260 IF T(I) = 0 THEN I = I - 1: GOTO 260
270 FOR J = I TO 1 STEP - 1
280 PRINT T(J);: NEXT
```

REELL.2

```
100 REM +--------------------+
110 REM !                    !
120 REM ! Euklid für B < 11  !
130 REM !                    !
140 REM +--------------------+
150 DIM S(30): ONERR GOTO 160
160 I = 0: HOME : INPUT "NEUE BASIS: ";B
170 INPUT "ZAHL: ";Z: REM Zulässige Ziffernzahl hängt von
    B ab.
180 S = INT (Z / B)
190 S(I) = Z - B * S
200 Z = S: IF Z = 0 THEN 220
210 I = I + 1: GOTO 180
220 PRINT : FOR J = I TO 0 STEP - 1
230 PRINT S(J);: NEXT
```

REELL.3

```
100 REM +--------------------+
110 REM !                    !
120 REM ! Euklid für B < 37  !
130 REM !                    !
140 REM +--------------------+
150 REM 26 Buchstaben + 10 Ziffern
160 DIM S$(30):0 = 55: ONERR GOTO 170: REM 65 = 10 + 55 =
    ASC("A")
170 HOME :I = 0: INPUT "NEUE BASIS? ";B
180 INPUT "WELCHE ZAHL? ";Z
190 S = INT (Z / B)
200 T = Z - B * S
210 S$(I) = STR$ (T)
220 IF T > 9 THEN S$(I) = CHR$ (T + 0): REM A = 10, B = 11
    usw.
230 Z = S: IF Z = 0 THEN 250
240 I = I + 1: GOTO 190
250 PRINT : FOR J = I TO 0 STEP - 1
260 PRINT S$(J);: NEXT
```

REELL.4

```
100 REM +--------------------+
110 REM !                    !
120 REM ! Basis B -> Basis 10 !
130 REM !                    !
140 REM +--------------------+
141 REM Nur für B < 11
142 REM Keine Ziffer darf größer sein als B-1
150 HOME : INPUT "WELCHE BASIS? ";B
160 INPUT "WELCHE ZAHL? ";Z$
170 L = LEN (Z$)
180 FOR I = L TO 1 STEP - 1
190 Z = VAL ( MID$ (Z$,I,1)): REM Ziffer
200 S = B ↑ (L - I): REM Stellenwert
210 D = D + Z * S
220 NEXT
230 PRINT : PRINT D
```

REELL.5

```
100 REM +--------------+
110 REM !              !
120 REM ! Dualbrüche   !
130 REM !              !
140 REM +--------------+
150 CLEAR : DIM S(33): TEXT : HOME : ONERR GOTO 150
160 INPUT "WELCHE ZAHL? ";S$: REM 2 ↑ 32 > Betrag > 2 ↑ (-34)
170 V$ = "+": IF LEFT$ (S$,1) = "-" THEN V$ = "-":S$ =
    RIGHT$ (S$, LEN (S$) - 1)
180 IF LEFT$ (S$,1) = "." THEN S$ = "0" + S$
190 S = VAL (S$): HOME : PRINT : PRINT V$S$: PRINT : PRINT
    "= "V$"%";: IF S < 1 THEN I = - 1
200 IF S > 2 ↑ EX THEN EX = EX + 1: GOTO 200
210 IF S < 2 ↑ EX THEN EX = EX - 1:I = I + (S < 1): GOTO 210
220 IF S < 2 ↑ EX THEN S(I) = 0
230 IF S > = 2 ↑ EX THEN S(I) = 1:S = S - 2 ↑ EX
240 EX = EX - 1:I = I + 1: IF EX = - 1 THEN BP = I
250 IF I < 32 THEN 220
260 IF BP = 0 THEN PRINT "0";
270 FOR I = 0 TO 33: IF I = BP THEN PRINT ".";
280 PRINT S(I);: NEXT
```

REELL.6 (nur mit 80 Z/Z)

```
100 REM +--------------------+
110 REM !                    !
120 REM ! Zahl -> Quintupel  !
130 REM !                    !
140 REM +--------------------+
150 PRINT CHR$ (4)"PR#3": PRINT CHR$ (12)
160 CALL - 998: INVERSE : PRINT "!" SPC( 74)"!"
170 PRINT "!" SPC( 23)"REELLE VARIABLE IN APPLESOFT" SPC(
    23)"!"
180 PRINT "!" SPC( 74)"!": NORMAL : POKE 34,3: GOSUB 740:
    ONERR GOTO 190
190 S$ = "": VTAB 4: POKE 36,1: PRINT SPC( 59)"<RETURN>
    beendet": VTAB 4: PRINT "X = ";
200 GET X$: PRINT X$;:X = ASC (X$): IF X = 8 THEN PRINT :
    GOTO 190:
210 IF X < > 13 THEN S$ = S$ + X$: GOTO 200
220 IF LEFT$ (S$,1) = "." THEN S$ = "0" + S$
230 L = LEN (S$): IF L = 0 THEN TEXT : PRINT CHR$ (12):
    GOTO 620
240 FOR I = 1 TO L:X = ASC ( MID$ (S$,I,1))
250 IF (X < 48 OR X > 57) AND X < > 46 AND X < > 45 AND X
    < > 43 OR (X = 45 AND X < > 43) AND I > 1 THEN PRINT
    CHR$ (7): GOTO 190
260 NEXT :S = VAL (S$):V = S: FOR I = 1 TO L: POKE 773 + I,
    ASC ( MID$ (S$,I,1)): NEXT : POKE 773,L
270 CALL - 958: GOSUB 730
280 PRINT "Erster Schritt: Darstellung als Binärbruch":
    PRINT "Bitte etwas Geduld!";
290 EX = 0:I = 0: ON S = 0 GOTO 370: IF ABS (S) < 1 THEN I =
    - 1
300 V$ = "+": IF LEFT$ (S$,1) = "-" THEN V$ = "-":S$ = MID$
    (S$,2):S = - S
310 IF S > 2 ↑ EX THEN EX = EX + 1: GOTO 310
320 IF S < 2 ↑ EX THEN EX = EX - 1:I = I + (S < 1): GOTO 320
330 IF S < 2 ↑ EX THEN S(I) = 0
340 IF S > = 2 ↑ EX THEN S(I) = 1:S = S - 2 ↑ EX
350 EX = EX - 1:I = I + 1: IF EX = - 1 THEN BP = I
360 IF I < 72 THEN 330
370 PRINT : VTAB 7: PRINT V$;: IF BP = 0 THEN PRINT "0";
380 FOR I = 0 TO 72: IF I = BP THEN PRINT ".";
390 PRINT S(I);: NEXT : PRINT : GOSUB 730
400 PRINT "Zweiter Schritt: Normierung"
410 ON V = 0 GOTO 700:I = 0:EX = 0: IF S(0) < > 0 THEN EX =
    BP: GOTO 430
420 IF S(I) = 0 THEN I = I + 1:EX = EX - 1: GOTO 420
430 POKE 768,EX + 128:E$ = "+":J = 0: IF EX < 0 THEN EX =
    - EX:E$ = "-"
440 E(J) = EX - 2 * INT (EX / 2)
450 EX = INT (EX / 2): IF EX > 0 THEN J = J + 1: GOTO 440
460 PRINT V$"0.";: GOSUB 690
470 PRINT " * 10↑("E$;: GOSUB 680: PRINT ")": GOSUB 730
480 PRINT "Dritter Schritt: Voranstellen des um 128
    vergrößerten Exponenten"
490 INVERSE : PRINT V$;:E(7) = 1: GOSUB 680: NORMAL
```

If E$ = "-" then for J = 1 to 7: E(J) = Abs(E(J) - 1): Next: If E(0) = 0 then for J = 1 to 7: E(J) = Abs(E(J) - 1): If E(J) < > 1 Then Next

```
500 PRINT " ";: GOSUB 690
510 PRINT : GOSUB 730: PRINT "Vierter Schritt: Aufteilung
    der Mantisse in Achtergruppen"
520 GOSUB 680: FOR J = I TO I + 31: IF INT ((J - I) / 8) =
    (J - I) / 8 THEN PRINT " ";
530 PRINT S(J);: NEXT : PRINT : GOSUB 730
540 PRINT "Fünfter Schritt: Erstes Mantissenbit durch
    Vorzeichenbit ersetzen"
550 S(I) = V$ = "-": GOSUB 680: PRINT " ";: INVERSE : PRINT
    S(I);: NORMAL
560 FOR J = I + 1 TO I + 31:K = (J - I) / 8: IF K = INT (K)
    THEN PRINT " ";
570 PRINT S(J);: NEXT : PRINT
580 GOSUB 730: PRINT "Letzter Schritt: Umwandlung der
    Achtergruppen in Dezimalzahlen"
590 FOR K = 0 TO 3: FOR J = 0 TO 7:Z(K) = Z(K) + S(I + J + 8
    * K) * 2 ↑ (7 - J): NEXT : POKE 769 + K,Z(K): NEXT
600 INVERSE : VTAB 22: PRINT PEEK (768);: FOR I = 0 TO 3:
    NORMAL : PRINT "  ";: INVERSE : PRINT Z(I);: NEXT :
    NORMAL : PRINT : GOSUB 730
610 PRINT "Diese fünf Zahlen werden im Anschluß an den
    Variablennamen gespeichert.";: GOTO 720
620 FOR I = 1 TO PEEK (773):R$ = R$ + CHR$ ( PEEK (773 +
    I)): NEXT
630 PRINT "Machen Sie die folgende Probe:": PRINT : PRINT
    "Tippen Sie zuerst 'NEW' und dann 'A = "R$"'."
640 PRINT : PRINT "Danach finden Sie die vorhin gefundenen
    Zahlen: ";: FOR I = 0 TO 4: PRINT "  " PEEK (768 + I);:
    NEXT
650 PRINT : PRINT : PRINT "durch 'FOR I = 2054 TO 2058 :
    PRINT PEEK ( I )"Q$" "Q$";: NEXT'."
660 PRINT : PRINT "ÜNEW": PRINT : PRINT "ÜA = "R$: PRINT :
    PRINT "ÜFOR I = 2054 TO 2058 : PRINT PEEK ( I )"Q$"  "
    Q$";: NEXT"
670 VTAB 20: INVERSE : PRINT "Benutzen Sie die
    Rechtspfeiltaste!": NORMAL : VTAB 9: END
680 FOR J = 7 TO 0 STEP - 1: PRINT E(J);: NEXT : RETURN
690 FOR J = I TO I + 31: PRINT S(J);: NEXT : RETURN
700 VTAB 15: PRINT "Die Zahl Null muß eine andere Behandlung
    erfahren, da sie nicht normiert werden"
710 PRINT "kann, Sie wird durch den Exponenten 0
    repräsentiert."
720 CLEAR : GOSUB 740: GOTO 190
730 FOR X = 1 TO 76: PRINT "-";: NEXT : PRINT : RETURN
740 DIM S(100):Q$ = CHR$ (34): RETURN
```

REELL.7

```
100 REM +-------------------+
110 REM !                   !
120 REM ! Quintupel -> Zahl !
130 REM !                   !
140 REM +-------------------+
150 REM Eingabe
160 HOME : FOR I = 1 TO 5: PRINT I;
170 INPUT ". ZAHL: ";Z$:Z(I) = VAL (Z$)
180 IF Z$ = "" OR Z(I) < 0 OR Z(I) > 255 THEN I = I - 1:
    VTAB PEEK (37): CALL - 868
190 NEXT
200 REM Vorzeichen
210 V$ = "+": IF Z(2) > 127 THEN V$ = "-"
220 IF Z(2) < 128 THEN Z(2) = Z(2) + 128
230 REM Umwandlung in reelle Zahl
240 FOR I = 1 TO 4
250 D = D + Z(I + 1) * 256 ↑ - I
260 NEXT : PRINT :K = Z(1)
270 IF K = 1 THEN D = D * 2 ↑ (K - 128): GOTO 300
280 D = D * 2 ↑ (K - 129)
290 D = D + D
300 PRINT "DIE ZAHL HEISST "V$;: IF D < 1 THEN PRINT "0";
310 PRINT D
```

REELL.8

```
100 REM +----------------------+
110 REM !                      !
120 REM ! Applesoft-Konstanten !
130 REM !                      !
140 REM +----------------------+
150 TEXT : HOME : HTAB 10: PRINT "APPLESOFT-KONSTANTEN":
    PRINT : PRINT
160 PRINT "NAME" SPC( 11)"WERT" SPC( 8)"SPEICHER": GOSUB 300
170 FOR K = 1 TO 13: READ A$,A
180 GOSUB 200: NEXT : GOSUB 300
190 PRINT : PRINT "MIN.INT IST UNGENAU, WEIL BYTE 5 FEHLT.":
    END
200 D = 0: FOR I = 1 TO 5
210 Z(I) = PEEK (A + I - 1): NEXT
220 V$ = "+": IF Z(2) > 127 THEN V$ = "-"
230 IF Z(2) < 128 THEN Z(2) = Z(2) + 128
240 FOR I = 1 TO 4:D = D + Z(I + 1) * 256 ↑ - I: NEXT
250 D = D * 2 ↑ (Z(1) - 128)
260 PRINT A$;: HTAB 11: PRINT V$;:D$ = STR$ (D): IF INT (D)
    = D THEN D$ = D$ + "."
270 IF D < 1 THEN D$ = "0" + D$
280 IF LEN (D$) < 11 THEN D$ = D$ + "0": GOTO 280
290 PRINT D$;: HTAB 25: PRINT A" BIS "A + 4: RETURN
300 FOR I = 1 TO 39: PRINT "-";: NEXT : PRINT : RETURN
310 DATA MIN.INT,57598,1,59667
320 DATA SQR(1/2),59693,SQR(2),59698
330 DATA -1/2,59703,LN(2),59708
340 DATA 10,59984,MILLIARDE,60692
350 DATA 1/2,61028,2LG(E),61147
360 DATA PI/2,61542,2*PI,61547,1/4,61552
```

REELL.9

```
100 REM +---------------+
110 REM !               !
120 REM ! Additionstest !
130 REM !               !
140 REM +---------------+
150 HOME : INPUT "1. SUMMAND: ";X
160 INPUT "2. SUMMAND: ";Y
170 Z = X + Y
180 REM Z=INT(1E9*Z+1E-9)/1E9
190 INPUT "RATE ERGEBNIS! ";G
200 PRINT : PRINT Z" = "G" (";
210 IF G = Z THEN PRINT "WAHR)"
220 IF G < > Z THEN PRINT "FALSCH)" CHR$ (7)
230 REM Beispiele für falsches "FALSCH":
240 REM 0.1+0.6,0.1+0.8,0.3+0.4
250 REM 0.3+0.6,0.4+0.5,0.6+0.7
```

REELL.10

```
100 REM +--------------------+
110 REM !                    !
120 REM ! Änderung von Byte 5 !
130 REM !                    !
140 REM +--------------------+
150 HOME : PRINT " <- UND -> BEWEGEN ↑  <RET> AENDERT BIT"
160 PRINT " <SPACE> = NEUE ZAHL  <ESC> = ENDE": FOR I = 1 TO
    40: PRINT "-";: NEXT : POKE 34,5
170 CLEAR :X$ = " ↑ ":B$ = CHR$ (8): HOME : FOR I = 1 TO 5:
    HTAB 14: PRINT I;
180 INPUT ". ZAHL: ";Z$:Z(I) = VAL (Z$)
190 IF Z$ = "" OR Z(I) < 0 OR Z(I) > 255 THEN I = I - 1:
    VTAB PEEK (37): CALL - 868
200 NEXT :ZW = Z(2): GOSUB 360
210 J = J + 1:S = INT (Z(5) / 2)
220 B(J) = Z(5) - 2 * S:Z(5) = S
230 HT = 19: IF S > 0 THEN 210
240 VTAB 14: HTAB 12: PRINT "NUMMER: ";: FOR I = 7 TO 0 STEP
    - 1: PRINT I;: NEXT
250 VTAB 15: HTAB 12: PRINT "BYTE 5: ";: INVERSE : FOR I = 8
    TO 1 STEP - 1
260 PRINT B(I);: NEXT : NORMAL
270 VTAB 16: HTAB HT: CALL - 868: PRINT X$B$B$;
280 WAIT - 16384,128:A = PEEK ( - 16384): POKE - 16368,0
290 IF A = 149 AND PEEK (36) < 26 THEN PRINT X$B$B$;:HT = HT
    + 1
300 IF A = 136 AND PEEK (36) > 19 THEN PRINT B$B$X$B$B$;:HT
    = HT - 1
310 ON A = 155 GOTO 480: ON A = 160 GOTO 170: ON A < > 141
    GOTO 280
320 B(27 - HT) = 1 - B(27 - HT): HTAB HT + 1: VTAB 15: PRINT
    B(27 - HT)
330 FOR I = 1 TO 5:T = PEEK ( - 16336): NEXT :J = 0: FOR I =
    1 TO 8
340 J = J + B(I) * 2 ↑ (8 - I): NEXT :Z(5) = J
350 GOSUB 450:Z(2) = ZW: GOSUB 360: GOTO 240
360 V$ = "+": IF Z(2) > 127 THEN V$ = "-"
370 IF Z(2) < 128 THEN Z(2) = Z(2) + 128
380 D = 0: FOR I = 1 TO 4:D = D + Z(I + 1) * 256 ↑ - I:
    NEXT : PRINT :K = Z(1)
390 IF K = 1 THEN D = D * 2 ↑ (K - 128): GOTO 410
400 D = D * 2 ↑ (K - 129):D = D + D
410 VTAB 22: CALL - 958: HTAB 12 - LEN ( STR$ (D)) / 2
420 PRINT "DIE ZAHL HEISST "V$;
430 IF D < 1 AND D > 1E - 3 THEN PRINT "0";
440 PRINT D: RETURN
450 S = 0: FOR I = 8 TO 1 STEP - 1
460 S = S + B(I) * 2 ↑ (I - 1): NEXT :Z(5) = S
470 INVERSE : VTAB 10: HTAB 23: CALL - 868: PRINT S:
    NORMAL : RETURN
480 TEXT : HOME : END
```

MAKES

Erstellung zusammenhängender

SUB

Subdirectories

von Arne Schäpers

Das hier aufgegriffene Problem ist seiner Natur nach so esoterisch, daß es, bevor wir dazu eine Lösung präsentieren, erst einmal einer Erklärung bedarf, worum es überhaupt geht:
Unter ProDOS hat ein Volume-Directory immer eine festgelegte Anzahl von Blocks und ist in seiner Anordnung auf dem Volume konsekutiv, d.h. die einzelnen Blocks, aus denen dieses Directory besteht, folgen direkt aufeinander. Ein Volume-Directory, das aus nicht konsekutiven Blocks besteht (z.B. mit den Blocknummern 4, 5, 17, 23) wäre zwar durch die Verkettung zwischen den einzelnen Blocks (Forward und Backward Reference) prinzipiell möglich, wird aber nicht genutzt.

1. Subdirectories unter ProDOS

1.1. Wie werden Subdirectories verwaltet?

Ein Subdirectory wird anders als ein Volume-Directory verwaltet: Bei der Erstellung dieses Directories mit dem CREATE-Befehl des BASIC.SYSTEMs, das in diesem Fall mit dem CREATE-Befehl des MLI identisch ist, wird ein Subdirectory erstellt, das aus einem einzigen Block besteht, der insgesamt 12 Dateieinträge aufnehmen kann; der erste Dateieintrag wird vom Header des Subdirectories belegt.
Erst wenn versucht wird, einen dreizehnten Dateieintrag in dieses Subdirectory aufzunehmen, wird es vom MLI durch Anhängen eines weiteren Blocks entsprechend vergrößert. Dieser Prozeß setzt sich bei weiteren Dateieinträgen fort: Für die Dateieinträge Nummer 26, 39, 52... wird jeweils ein weiterer Block an das Subdirectory angehängt. Die Verkettung zwischen den einzelnen Blocks, die aus jeweils zwei 16-Bit-Pointern besteht (Backward Reference: der vorherige Directory-Block hat die Blocknummer xxx – Forward Reference: der nächste Directory-Block hat die Blocknummer yyy) ist hier unabdingbar.

Um das zu demonstrieren, nehmen wir eine frisch formatierte Diskette, bei der nur die Blocks 0 bis 6 belegt sind, und CREATEn darauf das Subdirectory SUB. Der erste (und einzige) Block von SUB ist danach der Block Nummer 7.
Wenn man jetzt dieses Subdirectory durch einen „Einzeiler" mit Dateieinträgen füllt:

```
10 FOR X = 1 TO 20: PRINT CHR$(4)"BSAVE
   SUB/TEST"X",A$ 2000, L100": NEXT
```

so zeigt ein CATALOG nach Ablauf dieses Programms, daß SUB jetzt zwei Blocks umfaßt. Wo liegt dieser zweite Block?

ProDOS-Editor 1.0

Applesoft-Editor unter ProDOS-Betriebssystem

von U. Stiehl

1984, Diskette und Manual, DM 98,–
ISBN 3-7785-1024-X

Mit diesem neuen Editor – übrigens der bislang einzige deutsche ProDOS-Editor – wird dem Applesoft-Programmierer ein Werkzeug zur effektiven Programmierung unter dem Betriebssystem ProDOS gegeben, denn die früheren Editoren sind allesamt unter ProDOS nicht mehr lauffähig.

Unter anderem sind folgende Features implementiert worden:

- Zeilenorientierter Editor mit jedem erdenklichen Redigierkomfort (Insert, Delete, Tab, Restore, freie Cursorbewegung in allen vier Richtungen, Eingabe von Ctrl-Buchstaben in Applesoft-Zeilen usw.)
- Renumber (Zeilen-Umnumerierung)
- Xreference (sortierte Variablenliste)
- Suchen von Tokens, Strings und Variablen
- dezimale und hexadezimale Umrechnungen
- Ausführung von Monitorbefehlen aus dem Editor heraus
- Listen des Applesoft-Programms in speicherinterner Form als Hex-Dump
- Suchen von Hex-Folgen, Adressen oder Speicherstellen im gesamten RAM-Bereich einschließlich der Language-Card
- frei definierbare Tastatur-Macrobefehle

Der Applesoft-Editor liegt in einem von ProDOS geschützten Bereich und läßt sich per Tastendruck vorübergehend abschalten und ebenso einfach wieder aktivieren.

Gerätevoraussetzung: Apple II+, IIe oder IIc, 40 Zeichen/Zeile

Hüthig Software Service,
Postfach 10 28 69,
D-6900 Heidelberg

Vorausgesetzt, daß die Diskette außer SUB vorher keinen weiteren File enthalten hat, belegt TEST1 den Block 8, TEST2 belegt den Block 9 und so weiter, bis zum File TEST12, der den Block 20 belegt. Vor dem Erstellen von TEST13 wurde eine Erweiterung von SUB nötig: Der zweite Block von SUB hat damit die Blocknummer 21. Die weiteren Files TEST13 bis TEST20 belegen danach die Blocknummern 22 bis 29.

1.2. Nachteil der „normalen" Subdirectories

Die Folge sollte damit klar sein: Bei einer Suche nach dem File SUB/TEST20 (oder nur TEST20 nach einem PREFIX SUB) liest das MLI zuerst den Block 7 ein (den ersten Block von SUB) und stellt dabei fest, daß TEST20 nicht in diesem Block eingetragen ist. Da dieser erste Block von SUB eine Forward Reference enthält („Subdirectory noch nicht zu Ende"), wird danach über diese Referenz der zweite Block von SUB gelesen und ebenfalls nach dem Eintrag TEST20 abgesucht.
Zwischen dem ersten und dem zweiten Block liegen im Minimalfall 12 Datenblocks, nämlich dann, wenn alle 12 im ersten Directory-Block eingetragenen Files jeweils nur einen Datenblock belegen.
Generell: Zwischen zwei Blocks eines Subdirectories liegen sämtliche Files, die im „vorderen" der beiden Directory-Blocks verzeichnet sind.
Durch ein trickreiches Belegen der Diskette kann man den Abstand auch bei größeren Files auf die erwähnten 12 bzw. 13 Blocks herunterdrücken, aber diese Grenze ist auf Benutzerebene nicht zu unterschreiten.

1.3. Vorteil der mit MAKESUB erstellten Subdirectories

Die Auswirkungen auf die Zugriffsgeschwindigkeit werden immer deutlicher, je länger das Subdirectory ist und je weiter „hinten" der gesuchte File steht. Mit
CREATE/NORMSUB/SUB
wurde ein Subdirectory auf „normale" Weise auf dem Volume /NORMSUB erzeugt, auf einem zweiten Volume /TEST-SUB wurde ebenfalls eine Subdirectory SUB erzeugt, aber mit dem Programm **MAKESUB** (um das es hier letztendlich geht). Danach wurden beide Subdirectories nach Setzen des jeweiligen Präfixes mit dem folgenden Einzeiler beschrieben:

```
10 FOR X = 1 TO 240: PRINT CHR$(4) "BSAVE
   BITS"X",A$2000, L$50": NEXT
```

Schon beim Beschreiben werden die Unterschiede recht deutlich – wir erhalten hier eine „mittlere" Zugriffsgeschwindigkeit, weil der Reihe nach auf den ersten, zweiten... bis 240. Dateieintrag zugegriffen wird.
Noch eine Anmerkung: Tun Sie Ihren Laufwerken diesen Test nicht an – ich habe dabei die ganze Zeit ängstlich auf Rauchzeichen gewartet.

Tabelle 1 gibt Aufschluß über die Zeiten, die zum Erstellen und Einlesen der Files BITS1 bis BITS240 benötigt werden.
Wie man sieht, lohnt sich ein zusammenhängendes Subdirectory um so mehr, je weiter „hinten" innerhalb dieses Subdirectories zugegriffen werden muß.
Bleibt nur noch die Frage zu klären, woher der Unterschied von 0,04 Sekunden pro Zugriff auf BITS1 herrührt:
Bei einem „normalen" Subdirectory steht dieser File direkt als nächster Block nach dem ersten Block von SUB, bei einem mit Hilfe von MAKESUB erstellten Subdirectory liegt das Subdirectory selbst mit einer größeren Anzahl von Blocks (bei 240 Files sind es immerhin 19 Blocks) dazwischen.

1.4. Die Theorie von MAKESUB

Nachdem die Vorteile eines zusammenhängenden Subdirectories damit klargestellt sind, erörtern wir das Programm MAKESUB, mit dem dieses Subdirectory erstellt wurde.

Die erste Möglichkeit wäre ein MLI-Patch gewesen; sie scheidet aus mehreren Gründen aus:
a) Patches sollten nur erstellt werden, wenn es absolut unumgänglich ist.
b) Der Patch müßte für die Versionen 1.0.x und 1.1.1 verschieden ausfallen.
c) Ich besitze rund ein Dutzend Versionen von DOS 3.3 – manche laden schneller, andere zeigen bei einem CATALOG noch die Startadressen an, dafür funktioniert der INIT-Befehl nicht mehr – das ist mir eine Lehre.

Der Erstellungs- bzw. Erweiterungsprozeß eines konsekutiven Subdirectories besteht, wenn er auf legale Weise ausgeführt werden soll, aus erstaunlich vielen Schritten:

– Anforderung des Dateinamens und der Zahl der gewünschten Blocks vom Benutzer.
– GET FILE INFO für dieses Subdirectory. Existiert es bereits?
– Bestimmung des freien Platzes auf dem Volume. Ist genügend vorhanden?
– Wenn das Subdirectory noch nicht existiert, folgt ein CREATE.
– Auffinden des Dateieintrags für das Subdirectory und Eintrag der neuen Blockanzahl sowie des erhöhten EOF.
– Zurückschreiben des Dateieintrags und Laden des Subdirectories selbst.
– Anhängen der erforderlichen Anzahl von Blocks, die gleichzeitig natürlich auf dem Volume belegt werden müssen.

Auf diese Weise ergeben sich folgende Eigenschaften von MAKESUB:
– Es funktioniert mit allen Versionen von ProDOS.
– Es funktioniert mit jeder Art von Volume, d.h. mit 35-, 40-, 80- und 160-Spur-Laufwerken, Festplatten, RAM-Disks usw.
– Sowohl die Bearbeitung existierender Subdirectories (Verlängerung) als auch deren Neuerstellung ist möglich.
Darüber hinaus kann das Volume bereits eine beliebige Anzahl von Files enthalten, denn MAKESUB versucht, das Subdirectory mit fortlaufenden Blocknummern zu erstellen. Falls Sie vorher einige kurze Files gelöscht haben, werden die Blocknummern des Subdirectories u.U. nicht direkt fortlaufend, sondern nur so dicht zusammen wie möglich liegen.

2. Die Realisierung von MAKESUB

2.1. Allgemeines

Ein großer Teil der Unterprogramme ist den innerhalb des MLI verwendeten Funktionen sehr ähnlich – mit einem Unterschied:
Das Programm „weiß", daß der Eintrag des Subdirectories existiert, weil er vorher per GET FILE INFO gesucht und entweder gefunden oder CREATEd wurde, sowie daß genügend Platz auf dem Volume vorhanden ist. Damit ersparen wir uns sämtliche Error-Checks innerhalb der Directory-Suchroutine sowie bei der Belegung von Blocks in der VBM (Volume-Bit-Map).
Aufgrund der hohen Zahl der logischen Einzelschritte habe ich versucht, das Hauptprogramm auf deren Aufrufe zu beschränken. Dabei wurden einige Programmteile zu Subroutinen, die eigentlich nur einmal benutzt werden.

2.2. Spezielle Probleme

Schwierigkeiten bereitet eigentlich nur das BASIC.SYSTEM bei der Eingabe des Dateinamens und der Blockanzahl. Es versucht immer wieder einmal „dazwischenzuspuken" und wird deshalb beim Programmstart abgehängt. Das Programm endet grundsätzlich mit RESTART, einem Sprung zum BASIC-Kaltstart via $03D3, wo sich das BASIC.SYSTEM dann selber wieder anhängt.

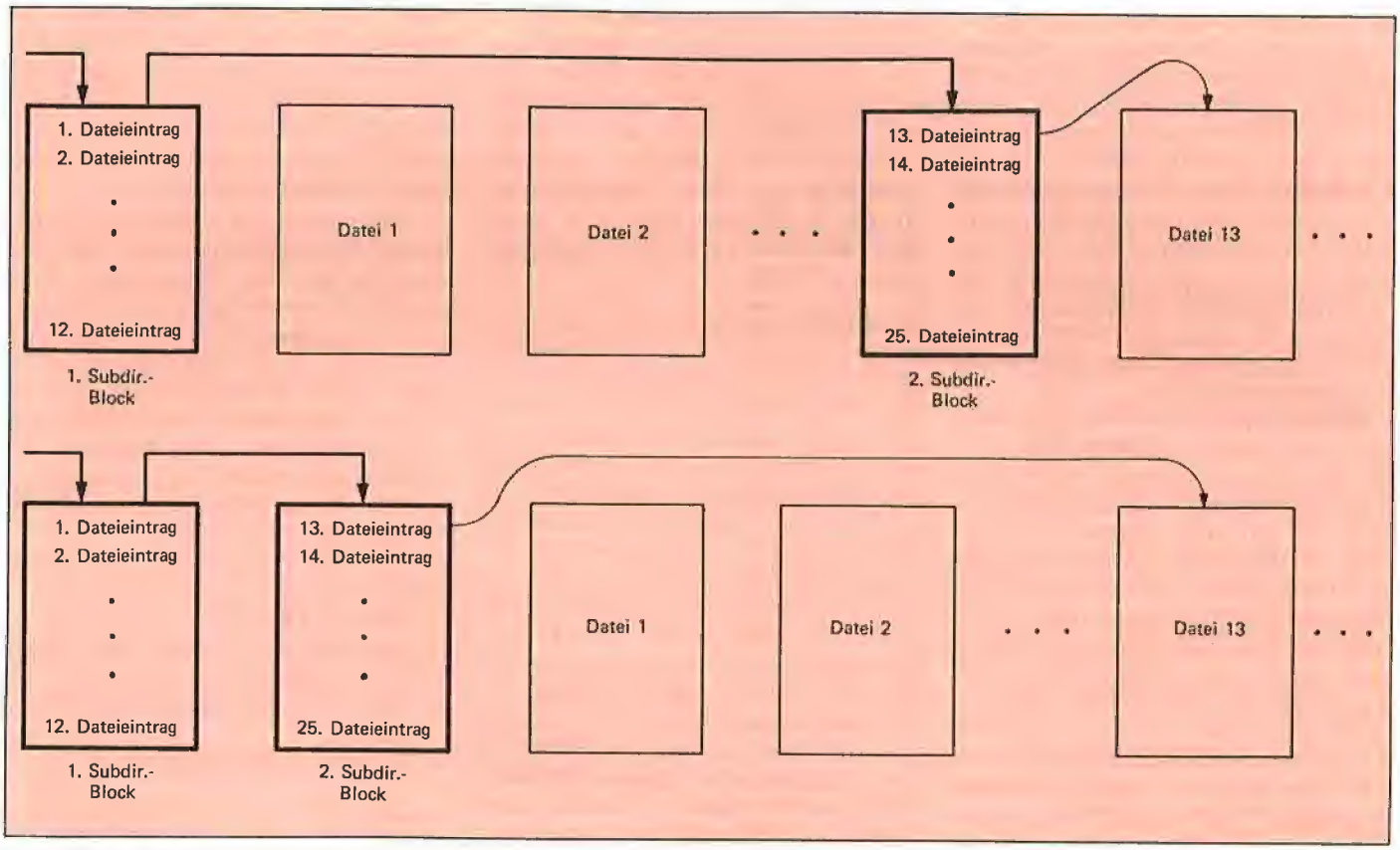


Abb.: Blockverteilung vor und nach MAKESUB

2.3. Die einzelnen Routinen
(in der Reihenfolge des Programmablaufs)

– **DISCONN** kopiert die Adressen nach $0036 bis $0039, die das BASIC.SYSTEM erst dann einsetzt, wenn es intern auf DOS-Befehl geprüft hat. Im Normalfall sind das die Startadressen der Ein/Ausgaberoutinen des Monitors. Damit ist das BASIC.SYSTEM „abgehängt".
Nach Ausdruck des Programmtitels und „Eingabe Filename: " wird die Monitorroutine GETLN benutzt, um den Path-Name abzufragen.
– Der anschließende Aufruf von **SETPATH** bewirkt folgendes: Wenn der Path-Name mit einem „/" beginnt, wird er direkt vom Eingabepuffer des Monitors ($0200) nach PATH kopiert, ansonsten wird vorher ein *GET PREFIX* durchgeführt und das erhaltene PREFIX steht dann vor dem eingegebenen Dateinamen in PATH. Das MLI liefert bei GET PREFIX immer Großbuchstaben zurück; die Benutzer-Eingabe muß gegebenenfalls in Großbuchstaben übersetzt werden.
– **GETNUM** liefert „Block-Anzahl (Hex): " und erwartet die entsprechende Eingabe, die dann via **SETNUM** von ASCII in eine Hexzahl umgewandelt und als BLOCKS_WANTED gespeichert wird. Wenn SETNUM irgendwelche illegalen Zeichen in der Hex-Eingabe festgestellt hat, kommt es mit gesetztem Carry zurück, und die Aufforderung wird wiederholt.

– **GETSTART** ist dann die letzte Möglichkeit (außer einem Reset natürlich), das Programm abzubrechen. Es wartet entweder auf ein <RETURN> oder ein <ESC>, wobei letzteres zum Abbruch von MAKESUB führt.
– Bei **GOTSTART** wird es dann etwas komplizierter: Das Programm nimmt erst einmal an, daß das Subdirectory noch nicht existiert und speichert BLOCKS_WANTED in BLOCKS_NEEDED (tatsächlich benötigte Blocks) um. Danach wird ein *GET FILE INFO* für das neue Subdirectory durchgeführt. Hier sind nur zwei Bedingungen erlaubt, entweder „File nicht gefunden" (**TSTFILE** gibt in diesem Fall ein „EQ" zurück) oder „File gefunden". Bei „Volume und/oder Directory nicht gefunden" folgt Abbruch mit „PATH NOT FOUND".

– Wenn das Subdirectory schon existiert, dann muß die Anzahl der bereits belegten Blocks von BLOCKS_NEEDED abgezogen werden. Des weiteren muß dann geprüft werden, ob dieser gefundene File tatsächlich ein Subdirectory ist – falls nicht, verabschiedet sich das Programm mit „FILE TYPE MISMATCH".
– Sollte sich bei einem existierenden Subdirectory ergeben, daß die Anzahl „Blocks gewünscht" gleich oder gar kleiner der bereits belegten Blockanzahl ist, endet MAKESUB entweder mit „-Fertig" oder „Verkürzung nicht möglich!".
– **TSTROOM** holt als nächstes via GET FILE INFO für das Volume-Directory des Volumes die Anzahl der freien Blocks, vergleicht mit BLOCKS_NEEDED und verabschiedet sich mit „VOLUME FULL", wenn nicht genügend Platz vorhanden ist.
– Wenn das Subdirectory noch nicht existiert hat, wird es jetzt CREATEd und ein Block von BLOCKS_NEEDED abgezogen. Bleibt dann nichts mehr übrig, d.h. wurde BLOCKS_WANTED mit 1 angegeben, ist das Programm damit beendet.
– Ansonsten geht es weiter mit **SETENTRY**. Hier wird via **GETFILE** (s.u.) stufenweise von dem Volume-Directory bis zum Dateieintrag des neuen Subdirectories gesucht und PTR auf diesen Eintrag gesetzt. Der entsprechende Directory-Block, in dem das Subdirectory als Eintrag steht, wird dabei geladen. Die gewünschte Anzahl von Blocks (BLOCKS_WANTED) wird im Dateieintrag gesetzt, genauso wie der neue EOF des Subdirectories. Danach wird dieser Directory-Block zurückgeschrieben.
– **GETDEND** sucht zuerst aus dem noch geladenen Dateieintrag die Nummer des ersten Blocks des Subdirectories heraus und liest dann das Subdirectory Block für

Block, bis das momentane Ende erreicht ist (die Forward Reference zum nächsten Subdirectory-Block hat dann den Wert 00 00). Bei einem Subdirectory, das gerade erst via CREATE erstellt wurde, ist das bereits beim ersten Block der Fall.
– **APPEND**: Wenn wir hier angekommen sind, befindet sich der momentan letzte Block des Subdirectories im Speicher. Der erste VBM-Block des Volumes wird mit **READVBM** gelesen. Danach folgt ein Sprung zu SETLINK, denn wir haben mindestens noch einen weiteren Block an das Subdirectory anzuhängen.
– **SETLINK** holt die Nummer des nächsten freien Blocks über **GNBLOCK** (s.u.) aus der VBM und trägt sie in die Forward Reference des momentanen Subdirectory-Blocks ein. Der Subdirectory-Block wird zurückgeschrieben. Wenn der gerade geschriebene Block nicht der letzte anzuhängende Block war (beim ersten Aufruf immer der Fall, der gerade geschriebene Block war der „alte" letzte Block des Subdirectories!), dann wird es kompliziert:

Wir haben gerade die Nummer des nächsten freien Blocks als Forward Reference eingetragen. Diese Forward Reference ist also der nächste Block, der bereits belegt ist und im folgenden geschrieben werden soll. Diese Blocknummer wird nach RWBLOCK kopiert. Gleichzeitig braucht der neue Subdirectory-Block eine Backward Reference, also einen Zeiger auf den vorherigen Block. Dieser Block wurde gerade geschrieben. Folge: RWBLOCK wird zuerst als Backward Reference in den nächsten Block kopiert, bevor es den Wert des nächsten zu schreibenden Blocks bekommt. Danach wird der neue Block bis auf die eben eingetragene Backward Reference gelöscht.
BLOCKS_NEEDED wird um eins heruntergesetzt: Bleiben dabei keine weiteren Blocks mehr übrig, dann ist der eben im Speicher vorbereitete Block der letzte und erhält keine neue Forward Reference (sie bleibt auf dem Wert 00 00). Ansonsten wird GNBLOCK wieder aufgerufen und liefert die Nummer des nächsten freien Blocks, der hier als Forward Reference eingetragen wird.
Die Schleife geht solange, bis BLOCKS_NEEDED den Wert 0 erreicht und der letzte Dir-Block mit einer Forward Reference von 00 00 geschrieben wurde.
Als letztes wird die VBM auf das Volume zurückgeschrieben, bevor das Programm mit **DONE** („- Fertig.") endet.

Von den benutzten Subroutinen seien hier nur die beiden komplizierteren beschrieben:

– **GETFILE** verwandelt zuerst den angegebenen Path-Name (+ PREFIX) in dieselbe Form, wie es auch die Routine „Copy and Verify Path-Name" innerhalb des MLI tut: Alle „/" im Pfad werden durch Längenangaben für die darauf folgenden Namen ersetzt. Danach wird – abweichend von der MLI-Prozedur – gleich der erste Name des Pfades (Volume-Name) übersprungen; dieser Name ist ja bereits durch das GET FILE INFO für das Subdirectory verifiziert.

SCANDIRS beginnt mit dem Laden des ersten Blocks des Volume-Directories (Block 00 02) und holt noch die Blocknummer des ersten VBM-Blocks für das Volume aus dem Header des Directories, bevor der Dateieintrag für den nächsten (hier zweiten) Namen des Pfades gesucht wird. Die folgenden Schleifen halten sich wieder ziemlich dicht an die Original-MLI-Prozedur: Ein Directory-Block wird geladen und nach dem entsprechenden Namen im Pfad abgesucht, gegebenenfalls durch Nachladen weiterer Blocks des momentanen Directories über die Forward Reference.
Wenn der Name als Eintrag gefunden wird, wird der Index innerhalb des Pfades auf den nächsten Namen gesetzt (**NXTDIR**). Wenn der Pfad dabei zu Ende ist, haben wir den gesuchten File, ansonsten geht der Prozeß mit dem nächsten Subdirectory weiter. Die Verifizierung, daß neu geladene Subdirectories auch wirklich Subdirectories sind und einen Test auf „Directory-Ende, Name nicht gefunden" können wir uns hier aus den erwähnten Gründen sparen. GETFILE kehrt mit einem gesetzten PTR zurück, der auf den gefundenen Dateieintrag zeigt.

– **GNBLOCK** sucht in der VBM den nächsten freien Block, belegt ihn und gibt die Blocknummer im Akku und Y-Register zurück. Auch hier entfallen Tests auf „kein freier Block gefunden"; es ist vorher bestimmt worden, ob genügend Platz auf dem Volume vorhanden ist.
Hier ergeben sich notwendigerweise einige Bit-Tricksereien:
Ein Byte innerhalb eines VBM-Blocks entspricht 8 Blocks, 0 steht für „belegt", 1 steht für „frei".
Ein VBM-Block umfaßt $200 Bytes und damit $200 * 8, also $1000 Blocks.
$1000 Blocks mit jeweils 512 Bytes entsprechen 2 Megabytes, d.h. wirklich genug für eine DISK II, aber um einiges zuwenig für eine ProFile mit 5 MBytes.
Deshalb gibt es in diesem Fall mehrere VBM-Blocks, die allerdings immer konsekutiv auf dem Volume angeordnet sind.
Die Speicherstelle EXTENT im Programm MAKESUB hat eine direkte Entsprechung in der MLI-Routine *ALLOCATE NEXT BLOCK* und steht für „x-ter" VBM-Block (die Zählung beginnt mit 0).

GNBLOCK geht von einem geladenen VBM-Block aus und sucht diesen Block in aufsteigender Reihenfolge solange ab, bis entweder der Block zu Ende ist (dann wird der EXTENT erhöht und der nächste Block geladen) oder ein Byte mit einem Wert ungleich $00 gefunden wird.

Innerhalb dieses Bytes steht das höchstwertige Bit (MSB) für die jeweils niedrigste Blocknummer. Zum Belegen des nächsten Blocks werden also solange Bits nach links herausgeschoben, bis dabei eine Eins abfällt. Durch Mitzählen der Shifts ergibt sich ein Index im Bereich von 0 bis 7. Das abgefallene Bit wird gelöscht, und die restlichen Bits werden wieder auf die ursprüngliche Position zurückgeschoben. Danach wird das veränderte Byte wieder in den VBM-Block geschrieben.
Die Nummer des so ermittelten Blocks ergibt sich aus:
(EXTENT * $1000)
+ (Speicherseite innerhalb des VBM-Blocks * $800)
+ (Index innerhalb dieser Speicherseite * $08)
+ (Index innerhalb des Bytes * 1).

Kurzhinweise

1. Zweck:
Erstellung eines zusammenhängenden Subdirectories unter ProDOS
2. Konfiguration:
II+ mit G/K und LC, IIe oder IIc;
ProDOS, alle Versionen
3. Test:
BRUN MAKESUB
4. Sammeldisk:
T.MAKESUB
(Big-Mac-Quelltext)
MAKESUB
(Maschinenprogramm)

Tabelle 1

```
1. Erstellung von BITS1 bis BITS240 in dem
   Subdirectory SUB:

   1 Stunde, 13 Minuten <-> 41 Minuten

2. FOR X = 1 TO 10: PRINT CHR$(4) "BLOAD
   BITS1": NEXT

   7 Sekunden <-> 7,4 Sekunden

3. FOR X = 1 TO 10: PRINT CHR$(4) "BLOAD
   BITS100": NEXT

   38 Sekunden <-> 17 Sekunden

4. FOR X = 1 TO 10: PRINT CHR$(4) "BLOAD
   BITS240": NEXT

   1 Minute, 30 Sekunden <-> 37 Sekunden
```

(Die links angegebenen Zeiten beziehen sich auf das „normal" erstellte Subdirectory, die rechts angegebenen Zeiten auf das mit MAKESUB erstellte Subdirectory.)

MAKESUB

BSAVE MAKESUB, A$0803, L1283

```
                        1  *-------------------------------*
                        2  *           MAKESUB             *
                        3  *   von Arne Schäpers, 1985     *
                        4  *-------------------------------*
                        5  *
                        6  PTR      EQU  $06        ($07)
                        7  PROMPT   EQU  $33        für GETLIN
                        8  COUTV    EQU  $36        I/O-Vektoren
                        9  *
                       10  PATH     EQU  $280       interner Path-Name
                       11  *
                       12  RESTART  EQU  $3D3       (Pro)DOS-Warmstart
                       13  *
                       14  * Monitor-Routinen:
                       15  *
                       16  HOME     EQU  $FC58
                       17  GETKEY   EQU  $FD0C      GET ein Zeichen
                       18  GETLIN   EQU  $FD6F      GET eine Zeile
                       19  PRBYTE   EQU  $FDDA      Print Akku in Hex
                       20  COUT     EQU  $FDED      PRINT
                       21  *
                       22           ORG  $803
                       23  *
0803: A2 03            24           LDX  #03
0805: BD 30 BE         25  DISCONN  LDA  $BE30,X    BASIC.SYSTEM ab!
0808: 95 36            26           STA  COUTV,X
080A: CA               27           DEX
080B: 10 F8            28           BPL  DISCONN
                       29  *
080D: 20 58 FC         30           JSR  HOME
0810: A2 00            31           LDX  #0
0812: 20 18 0C         32           JSR  PRT1       "Erstellung von..."
0815: E8               33           INX
0816: 20 10 0C         34           JSR  PRTMSG     "Eingabe Filename: "
0819: A9 BA            35           LDA  #$BA
081B: 85 33            36           STA  PROMPT     ":"
081D: 20 6F FD         37           JSR  GETLIN
0820: 20 46 09         38           JSR  SETPATH    Path + PREFIX => $280
0823: A2 51            39  GETNUM   LDX  #GETLEN-MSGS
0825: 20 10 0C         40           JSR  PRTMSG     "Block-Anzahl: "
0828: 20 6F FD         41           JSR  GETLIN
082B: 20 7B 09         42           JSR  SETNUM     ASCII => Hex
082E: B0 F3            43           BCS  GETNUM     illegale Zeichen: Loop
0830: A2 66            44           LDX  #START-MSGS
0832: 20 10 0C         45           JSR  PRTMSG     "<CR>=Start, <ESC>..."
0835: 20 0C FD         46  GETSTART JSR  GETKEY
0838: C9 8D            47           CMP  #$8D       <CR>?
083A: F0 07            48           BEQ  GOTSTART
083C: C9 9B            49           CMP  #$9B       <ESC>?
083E: D0 F5            50           BNE  GETSTART
0840: 4C DB 0B         51           JMP  EXIT       Abbruch
                       52  *
0843: AD 43 09         53  GOTSTART LDA  BLWANT+1   Anzahl gewünschte
0846: AC 42 09         54           LDY  BLWANT     Blocks =>
0849: 8D 45 09         55           STA  BLNEED+1   Anzahl benötigte
084C: 8C 44 09         56           STY  BLNEED     Blocks
084F: 4E 41 09         57           LSR  EXISTS     Flag zurück
0852: 20 BF 09         58           JSR  TSTFILE    sucht SubDIR via MLI
0855: F0 36            59           BEQ  TSTROOM    nicht gefunden
0857: C9 00            60           CMP  #$00       SubDIR existiert?
0859: F0 03            61           BEQ  TSTTYPE
085B: 4C FF 0B         62           JMP  NOVOL      nein, PATH NOT FOUND
085E: A9 80            63  TSTTYPE  LDA  #$80
0860: 8D 41 09         64           STA  EXISTS     Flag setzen
0863: AD 1B 0A         65           LDA  STYPE      aus TSTFILE, FILE INFO
0866: C9 0D            66           CMP  #$0D       ist SubDIR?
0868: F0 03            67           BEQ  SETNEED
086A: 4C F7 0B         68           JMP  BADTYPE    "FILE TYPE MISMATCH"
086D: AD 44 09         69  SETNEED  LDA  BLNEED     SubDIR existiert:
0870: 38               70           SEC             die Anzahl der
0871: ED 1C 0A         71           SBC  BLUSED     benötigten Blocks
0874: 8D 44 09         72           STA  BLNEED     wird entsprechend
0877: AD 45 09         73           LDA  BLNEED+1   heruntergesetzt
087A: ED 1D 0A         74           SBC  BLUSED+1
087D: 8D 45 09         75           STA  BLNEED+1
0880: 90 08            76           BCC  TOOLONG    USED > NEEDED!
0882: 0D 44 09         77           ORA  BLNEED     zusätzliche Blocks
0885: D0 06            78           BNE  TSTROOM    = 00 00?
0887: 4C 24 09         79           JMP  DONE       nichts zu tun...
088A: 4C FB 0B         80  TOOLONG  JMP  BADLEN     "SET EOF?"
                       81  *
088D: 20 DB 09         82  TSTROOM  JSR  GETROOM    GET F'INFO VolDIR
0890: B0 03            83           BCS  ROOMOK     und Test BLOCKS_FREE
0892: 4C 03 0C         84           JMP  NOROOM     "VOLUME FULL"
                       85  *
0895: AD 30 BF         86  ROOMOK   LDA  $BF30      "last used Unit"
0898: 8D 22 0B         87           STA  RWUNIT     für BLOCKREAD/WRITE
089B: 2C 41 09         88           BIT  EXISTS
089E: 30 0B            89           BMI  SETENTRY
08A0: 20 26 0A         90           JSR  MAKEDIR    neues DIR CREATE
08A3: 20 2C 09         91           JSR  DECNEED    BLOCKS_NEEDED-1
08A6: D0 03            92           BNE  SETENTRY   bleiben noch Blocks
08A8: 4C 24 09         93           JMP  DONE       fertig
                       94  *
08AB: 20 3E 0A         95  SETENTRY JSR  GETFILE    sucht SubDIR und setzt
08AE: A0 13            96           LDY  #$13       PTR auf Dateieintrag
08B0: AD 42 09         97           LDA  BLWANT     für das SubDIR
08B3: 91 06            98           STA  (PTR),Y
08B5: C8               99           INY
                      100           LDA  BLWANT+1   BLOCKS_USED im
08B9: 91 06           101           STA  (PTR),Y    File-Eintrag
08BB: C8              102           INY
08BC: C8              103           INY
08BD: AD 42 09        104           LDA  BLWANT     EOF des SubDIR
08C0: 0A              105           ASL             => BLOCKS * $200
08C1: 91 06           106           STA  (PTR),Y    und wird gesetzt
08C3: C8              107           INY
08C4: AD 43 09        108           LDA  BLWANT+1
08C7: 2A              109           ROL
08C8: 91 06           110           STA  (PTR),Y    EOF, höchstes Byte
08CA: 20 27 0B        111           JSR  MLIWRITE   schreibt DIR zurück
                      112  *
08CD: 20 00 0B        113           JSR  SETBLOCK1  setzt 1.Bl. aus Eintr.
08D0: 20 10 0B        114  GETDEND  JSR  RDBLOCK    liest ersten/nächsten
08D3: AD 02 10        115           LDA  DIRBUF+2   Block des SubDIR
08D6: A8              116           TAY             nach DIRBUF
08D7: 0D 03 10        117           ORA  DIRBUF+3   FORWARD REF = 00 00?
08DA: F0 06           118           BEQ  APPEND     ja, SubDIR jetzt Ende
08DC: AD 03 10        119           LDA  DIRBUF+3
08DF: 4C D0 08        120           JMP  GETDEND    geht noch weiter
                      121  *
08E2: 20 33 0B        122  APPEND   JSR  READVBM    1.VBM-Block n. VBMBUF
08E5: F0 29           123           BEQ  SETLINK    "always"
                      124  *
08E7: AD 26 0B        125  NEWBLOCK LDA  RWBLOCK+1  Vorbereiten eines
08EA: AC 25 0B        126           LDY  RWBLOCK    neuen SubDIR-Blocks:
08ED: 8D 01 10        127           STA  DIRBUF+1   die Nummer des zuletzt
08F0: 8C 00 10        128           STY  DIRBUF+0   gelesenen Blocks wird
08F3: AD 03 10        129           LDA  DIRBUF+3   als BACKWARD REF,
08F6: AC 02 10        130           LDY  DIRBUF+2   die alte FORWARD REF
08F9: 8D 26 0B        131           STA  RWBLOCK+1  als nächste Blocknr.
08FC: 8C 25 0B        132           STY  RWBLOCK    für WRITE eingetragen
                      133  *
08FF: A2 00           134           LDX  #0         danach wird der
0901: 8A              135           TXA             neue Block gelöscht,
0902: 9D 02 10        136  CLRBLOCK STA  DIRBUF+2,X
0905: 9D 00 11        137           STA  DIRBUF+$100,X
0908: E8              138           INX             bis auf die Bytes 00
0909: D0 F7           139           BNE  CLRBLOCK   und 01 (BACKWARD REF)
                      140  *
090B: 20 2C 09        141  NXTLINK  JSR  DECNEED    BLNEED-1
090E: F0 09           142           BEQ  LINK00     => letzter Block
0910: 20 7E 0B        143  SETLINK  JSR  GNBLOCK    nein, nächster Block
0913: 8D 03 10        144           STA  DIRBUF+3   wird belegt und als
0916: 8C 02 10        145           STY  DIRBUF+2   FOWARD REF eingetragen
0919: 20 27 0B        146  LINK00   JSR  MLIWRITE   schreibt den Block
091C: 20 3A 09        147           JSR  TSTNEED    noch weitere Blocks?
091F: D0 C6           148           BNE  NEWBLOCK
0921: 20 5E 0B        149           JSR  WRITEVBM   VBM zurückschreiben
                      150  *
0924: A2 00           151  DONE     LDX  #DONMSG-MSGS
0926: 20 10 0C        152           JSR  PRTMSG     "- Fertig."
0929: 4C DB 0B        153           JMP  EXIT
                      154  *
092C: AD 44 09        155  DECNEED  LDA  BLNEED     setzt BLOCKS_NEEDED
092F: 38              156           SEC             um eins herunter und
0930: E9 01           157           SBC  #01        prüft, ob noch
0932: 8D 44 09        158           STA  BLNEED     weitere Blocks
0935: B0 03           159           BCS  TSTNEED    anzuhängen sind
0937: CE 45 09        160           DEC  BLNEED+1
093A: AD 45 09        161  TSTNEED  LDA  BLNEED+1
093D: 0D 44 09        162           ORA  BLNEED     beide auf 00?
0940: 60              163           RTS
                      164  *
                      165  EXISTS   DS   1          <$80, wenn SubDIR neu
                      166  *
                      167  BLWANT   DS   2          BLOCKS_WANTED
                      168  BLNEED   DS   2          BLOCKS_NEEDED
                      169  *
                      170  ************************
                      171  *
0946: A2 00           172  SETPATH  LDX  #00
0948: BD 00 02        173           LDA  $0200,X
094B: 29 7F           174           AND  #$7F       Start des Path
094D: C9 2F           175           CMP  #'/'       mit "/"?
094F: F0 0E           176           BEQ  GOTPATH
0951: 20 00 BF        177           JSR  $BF00      ansonsten
0954: C7              178           DFB  $C7        GET PREFIX
```

peeker Börse

Verkauf Hardware

2 Siem. Slim. Lauf Contr. 500M; 2 TEAC (FD55F) 2×40/80 Ehr. C. DM 620,–; OLYMPIA (DP165) Matr. DR.; IIe Enhanced 128K-Kompa + Operator + Mouse + Accel. + 80Z, Z80B Swyftc. Timem IIH.0. Uhr, Joyst. Softw. Neuw. VS+BN, Tel. 02 28/67 74 23

Fernschreiberinterface am Gameport m. Programm DM 79,– P. Benner, Hubertusstr. 131, 4150 Krefeld

Apple II+ (Komp.), IBM-Look, 2LW, Z80, 64K, Uhr, DR-Schnitt, 80 Zeichen, jede Menge SW und Literatur. 0 63 81 / 15 80 (VHB: DM 2400,–)

Apple comp. PC, 2LW, Wx. Tastatur Monitor 80ZK HB+Software DM 1890,– Printer MP/80 inc. Interface EPSON comp. DM 658,– Monitor, Drives, Zusatzkarten. Preis auf Anfrage. IFC Computer, Tel. 0 22 45 / 47 37

OPERATOR Tastatur IIe DM 420,–; WILDCARD (+/e) DM 300,–; PREMIUM Softcard IIe (Z80B, CP/M, 80ZZ+64K) DM 650,–; TEAC FD55A+Contr. DM 290,– Imagewriter + Kabel DM 850,– Par. Int. (OLYMPIA von (B&K) DM 160,–; Tel. 0 28 01 / 17 86.

Apple II Interface-Karten in verschiedenen Ausführungen **Optokoppler-Karte ab DM 85,–** 6522-Karte ab DM 95,–
Stimmler Elektronik
Heinlenstraße 34, 7400 Tübingen
Tel. 0 70 71 / 7 45 44

Z80B-Karte mit 64KB RAM + 256KB-RAM-Karte für Apple incl. Pseudodisksoftware und Betriebssystemanpassung DOS, CP/M + Pascal zusammen DM 700,–,
G. Hain, Tel. 0 91 22 / 7 77 11

Mac Profi Speichererweiterung 128 ▷ 512KB DM 498,–, Baus. 189,–, Fa. Schlösser ab 17.00 Tel. 089 / 98 58 89

Apple II+/e 512 KB-Ramdisk, Treiber für CP/M 2.20, 2.23, PRODOS, UCSD 1.1, 1.2, DOS. Info kostenlos. Tel. 0 70 31 / 3 62 10

SUPER-Tastatur für Apple II+ und kompat., MAK II von Multitech für DM 376,–, Video 1000 DM 279,–, Grappler+ DM 155,–, HEKO-elektr., PF 54 13, 7750 Konstanz

Drucker Speedy 100-80, wenig benutzt DM 600,–.
Tel. 05 11 / 604 07 78

Orig. Apple Scribe Printer, farb. + grafikfähig, neuwertig nur DM 520,–!!! Tel. 0 72 02 / 81 31

Apple IIe, 2 Disk II mit Contr. 80Z + RGB, Taxan-Farbmonitor, Z80 Superserialcard, Imagewriter, Joystick, Paddle, 16 Handbücher alles org. nur kompl. VB DM 6000,–, Tel. 0 26 02 / 6 02 58 ab 18 Uhr

Verkaufe 7 St. Apple IIe/c/+ mit Laufwerken, Maus, Z-80, 80Z, Video-Digitizer, Joyst., 128K, SSC, Epson-Modem, Literatur, MX-80 Drucker ...
Räumungsverkauf wegen Systemwechsel ca. zum halben Neupreis.
Asolf Lazi Schule Stuttgart,
Tel. 07 11 / 24 06 09

Z80H (AP22) 8Mhz mit Software DM 650,–, 256K RAM Disk (AP17) mit Software DM 300,–, Tel. 0 25 07 / 14 90

Profile Harddisk 10MB mit Acc. Kit zu Apple II Preis VS, Tel. Schweiz 061 / 47 89 34, Hr. Brunner

CPA-80 Matrixdrucker neuwertig mit Kabel für DM 650,– zu verkaufen, Tel. 0 23 51 / 7 92 80

Verkauf Software

IIe+c: C.. paint: exakte Zeichnungen m. Beschriftung (2 Stärken), 40 Funktionen, 80 fertige Symbole für ...spläne + elektr. Schalt... worden, ob Disk: DM 99,– Schr...ame ver..., M.-Müller-Ring 7, 6500 Mainz

Apple: Public Domain, Kermit: Pro Volume DM 15,–. Games, Schach, Graphic u.v.a. m. Derw. Lehrerprogramme, die Graphiksprache 'Minilogo'! Gratisinfo: Fa. Waltraud Muhle, Waldwinkel 3, 2105 Seevetal 3

Verk. weg. Systemw. Einzelstücke alles Orig.: Print Shop DM 75,–; Apple Business Graphics DM 330,–; Quick File IIe DM 110,–; Flight Simulator II DM 90,–; Visischedule/ Netzplantechnik DM 180,–;
Tel. 0 28 01 / 17 86

Verkaufe EXBASIC LEVEL II, Basicerweiterung für Apple II, DM 250,–. Tel. 0 47 06 / 10 22

Schach WM 85: Alle Partien für CHESS 7.0: Disk. DM 45,– GO-Progr. (Hayden): Disk. DM 65,–. Tel. 0 44 81 / 15 40

dBase II, registriertes Originalprogramm, für DM 800,– abzugeben. Tel. 0 47 43 / 55 00.

*******STOCKMASTER II******* Das Apple II-Programm für echte Börsengewinne. Diskette nur DM/ SFr. 485,–. Beschreibung 'pe02' anfordern bei: Töngi Computer-Praxis, Aspeltstr. 4, 6500 Mainz. für die Schweiz: Denton Consultants AG, Auwisstr. 17, CH-8127 Forch/Zürich.

BASPLUS II, 30 neue Basic-Befehle für Apple II+/e/c. z. B. neuer INPUT m. Insert u. Delete usw. Info kostenlos bei Guido Schrörs, Corneliusstr. 29, 4154 Tönisvorst 1

Schönschrift-Editor für PANASONIC-Drucker unter Apple-CP/M (80-Ze.-K.) Komfortable Bedienung mit SCHREIB und CUBIC. Nur DM 89,–. Tuxhorn, St.-Zweig-Str. 45, 4400 Münster, Tel. 0 25 33 / 18 04

3 000 000,– Gewinnen??? Softcontrol Lotto-Prg. (LOT2000) Alle bisherigen Ziehungen abgespeichert. Auswertungen und Ziehungen automatisch. Wird auf Apple II-CP/M-Diskette geliefert. Preis komplett mit P/V DM 29,90
Manfred Filbrich, Hauptstraße 22, 3171 Adenbüttel, Tel. 0 53 04/17 64

APPLEWORKS für IIe, IIc wegen Systemwechsel zu verkaufen, Originalzustand nur 1× eingesetzt! VB DM 500,–. Effer, Köln,
Tel. 02 21 / 68 78 59

CP/M 3.0 – komfortabel & schnell – für Apple //e und //c. Patch nur DM 95,–. Gratisinfo anfordern! Jörg Stattaus Software, Hollandtstr. 55, 4400 Münster

Software Uhr für Apple II+, e, c, Zeitschaltmöglichkeit Diskette + Anleitung DM 25,– Oecking
Tel: Do. 02 31 / 39 19 20

Ankauf Software

PASCAL-Terminal Progr. Gesucht. Tel. 0 73 44 / 52 45

Suche Listschutz-Progr. für PRODOS, Chiffre-Nr. P1007

Wir suchen komfortable Faktura (PRODOS) f. Apple IIe mit kompletter Lagerbewirtschaftung, Statistik, Rückstand, Gutschrift usw.
Intercolor, A-6850 Dornbirn,
Tel. 00 43 - 55 72 - 6 47 10

Wer paßt CP/M an Apple compat./Z80 an oder verkauft Apple-CP/M? Tel. 09 41 / 79 59 55

Ankauf Hardware

Suche Laufwerk für Apple II+; Seeger u.d. Schanz 17, 7135 Wiernsheim

Verschiedenes

APPLE REPARATUREN
(auch compatible M-boards, z.B. **Atlas, Arca, CES, Datastar, Dipa, Lasar, Mewa, PC-48 + 64, Plato, Radix, o. ae.) sowie Zusatzkarten und Disk-Drives** führt unser Spezialistenteam mit mehr als 5-jähriger Kunden- und Reparatur-Dienst-Erfahrung, garantiert zuverlässig und besonders kostengünstig aus.
Bitte genaue Fehlerangabe sowie Tel. Nr. für evtl. Rückfragen nicht vergessen.
Auf Wunsch Kostenvoranschlag.
aaa-electronic gmbh
Habsburgerstr. 134, 7800 Freiburg,
Tel. 0761/276864, Tx. 772642aaad

Absolute Neuheit für Apple IIe!!! Umrüstung auf 32 fest und 5 frei programmierte Tasten. EPROM DM 87,–. F. Fuhrmann, Schwarzwaldstr. 38, 7530 Pforzheim. Info g. Rückporto.

Wir reparieren Ihren Apple komp. ASC-Elektronik, Hirschgraben 9-11, 5100 Aachen, Tel. 02 41/2 52 26

Ihre Erphi-Vertretung für die Schweiz: Beltronic, Im Chapf, 8455 Rüdlingen, Tel. 0 18 67 31 41, Telex 8 25 981.

Kontakte

Suche Vertriebspartner für meine IIer-Profisoftware. E. Heinz Waldgürtel 7, 5060 Berg. Gld. 1

Für Ihre Unterlagen

Abonnement bestellt

am: _____

Vertrauensgarantie:
Ich habe davon Kenntnis genommen, daß ich die Bestellung schriftlich durch Mitteilung an den Dr. Alfred Hüthig Verlag, Postfach 10 28 69, 6900 Heidelberg 1 innerhalb von 7 Tagen widerrufen kann. Zur Fristwahrung genügt die rechtzeitige Absendung des Widerrufs (Datum des Poststempels).

Peeker
Leserservice
Postfach 10 28 69
6900 Heidelberg 1

Für Ihre Unterlagen

Folgende Bücher bestellt:

_____

_____

_____

_____

_____

am: _____

bei:

Peeker
Versandbuchhandlung
Postfach 10 28 69
6900 Heidelberg 1

Für Ihre Unterlagen

Folgende Disketten
und Programme bestellt:

_____

_____

_____

_____

_____

am: _____

bei:

Peeker
Softwareabteilung
Postfach 10 28 69
6900 Heidelberg 1

PEEKER

Abo-Karte

Ja, ich möchte **Peeker** abonnieren.
Liefern Sie mir **Peeker** ab Ausgabe ..... zum Jahresbezugspreis von DM 72,– (Inland) inkl. MwSt. Die Lieferung erfolgt frei Haus. Porto, Verpakkung und Zustellgebühren übernimmt der Verlag. Der Jahresbezugspreis für das Ausland beträgt DM 72,– inkl. MwSt., zzgl. DM 18,– Versandspesen.

Ich wünsche jährliche Berechnung durch:

☐ Verlagsrechnung ☐ Abbuchung von meinem Bank- bzw. Postscheckkonto

Bank / PschA

Bankleitzahl Kto.-Nr.

Datum Unterschrift

PEEKER

Buch-Karte

Bitte senden Sie mir gegen Rechnung folgende Bücher:

Menge	Autor, Titel	à DM	gesamt DM

Datum Unterschrift

PEEKER

Software-Karte

Bitte senden Sie mir gegen Rechnung folgende Disketten:

☐ Peeker-Sammeldiskette, einzeln
Disk# _____, Disk# _____
Disk# _____, Disk# _____
Preis je Disk DM 28,– (einzeln)

☐ Peeker-Sammeldiskette,
im Fortsetzungsbezug
ab Disk # _____
(Mindestbezug 6 Disketten)
Preis je Disk DM 20,–
Neben DOS-Disketten auch liefern
☐ CP/M ja ☐ CP/M nein
☐ Pascal ja ☐ Pascal nein

☐ Apple DOS 3.3, Begleitdiskette, DM 28,–
☐ ProDOS, Band 1, Begleitdiskette, DM 28,–
☐ ProDOS, Band 2, Begleitdiskette, DM 28,–
☐ Apple Assembler, Begleitdiskette, DM 28,–
☐ ProDOS-Editor 1.0, Programm, DM 98,–
☐ MMU 2.0, Programm, DM 98,–
☐ INPUT 2.0, Programm, DM 98,–
☐ Softbreaker 1.0, Programm, DM 48,–
☐ DB-Meister, Programm, DM 290,–
☐ Superplot, Programm, DM 48,–
☐ Superquick, Programm, DM 48,–
☐ Turtle Graphics, Programm, DM 98,–

Datum Unterschrift

Abo-Karte

Name

Firma

Abteilung

Straße

PLZ/Ort

Vertrauensgarantie:
Ich habe davon Kenntnis genommen, daß ich die Bestellung schriftlich durch Mitteilung an den Dr. Alfred Hüthig Verlag, Postfach 10 28 69, 6900 Heidelberg 1 innerhalb von 14 Tagen widerrufen kann. Zur Fristwahrung genügt die rechtzeitige Absendung des Widerrufs (Datum des Poststempels).

Datum

Unterschrift

Verlagshinweis:
Das Abonnement verlängert sich zu den jeweils gültigen Bedingungen um ein Jahr, wenn es nicht 2 Monate vor Jahresende schriftlich gekündigt wird.

Buch-Karte

Karte bitte vollständig ausfüllen

Vorname, Name

Firma

Straße

PLZ/Ort

Telefon mit Vorwahl

Software-Karte

Karte bitte vollständig ausfüllen

Vorname, Name

Firma

Straße

PLZ/Ort

Telefon mit Vorwahl

POSTKARTE

Peeker
Leserservice
Dr. Alfred Hüthig Verlag
Postfach 10 28 69
6900 Heidelberg 1

POSTKARTE

Peeker
Buchabteilung
Dr. Alfred Hüthig Verlag
Postfach 10 28 69
6900 Heidelberg 1

POSTKARTE

Peeker
Softwareabteilung
Dr. Alfred Hüthig Verlag
Postfach 10 28 69
6900 Heidelberg 1

INPUT 2.0

Ein Bildschirm-Maskengenerator für DOS 3.3 und ProDOS

von U. Stiehl

1984, Diskette und Manual, DM 98,–
ISBN 3-7785-1021-5

„Input 2.0" liegt wahlweise in der Bank 1 oder Bank 2 der Language Card und wird durch einen kurzen Driver in den unteren 48K aufgerufen.
Für jedes Feld der Bildschirmmaske lassen sich u. a. definieren: Feldlänge (bis zu 255 Zeichen) – Vtab – Htab – Datentyp (insgesamt 8 Typen) – Scrollflag (starre oder dynamische Maske) – Ctrlflag – Füllflag – Löschflag – Bildschirmflag (40- oder 80-Z-Darstellung). Innerhalb eines Eingabefeldes besteht jeder denkbare Redigierkomfort (Insert, Delete, Rubout, Restore usw.).
Gerätevoraussetzung: Apple IIe oder IIc; ferner Apple II+ im 40-Zeichenmodus

MMU 2.0

Memory Managements Utilities

für die Apple IIe 64K-Karte DOS 3.3 (und ProDOS)

von U. Stiehl

1984, Diskette und Manual, DM 98,–
ISBN 3-7787-1023-1

Insgesamt enthält die neue „MMU 2.0"-Diskette über 25 Programme, die neue Einsatzmöglichkeiten für die Extended 80 Column Card (erweiterte 80-Z-Karte = 64K-Karte für den Apple IIe) erschließen. Ein Teil der Programme laufen auch auf dem Apple II Plus, doch ist „MMU 2.0" primär für 64K-Karte-Besitzer gedacht.
Gerätevoraussetzung: Apple IIe mit 64K-Karte oder IIc

Softbreaker 1.0

Eine softwaremäßige Interrupt-Utility für die Apple IIe 64K-Karte

von U. Stiehl

1984, Diskette und Manual, DM 48,–
ISBN 3-7785-1022-3

Softbreaker ist ein Assemblerprogramm, mit dessen Hilfe Programme, die sich von der 64K-Karte (= Extended 80 Column Card für den Apple IIe) starten lassen, unterbrochen, gespeichert, geladen und exakt an der Stelle der Unterbrechung fortgeführt werden können. Dadurch ist es auch möglich, Sicherungskopien von sogenannten kopiergeschützten Programmen herzustellen.
Mit Softbreaker unterbrochene Programme werden komplett, d. h. die ganzen 64K einschließlich Language Card, in nur ca. 11 Sekunden auf einer formatierten Diskette gesichert.
Gerätevoraussetzung: Apple IIe mit 64K-Karte

Hüthig Software Service,
Postfach 10 28 69, D-6900 Heidelberg

```
0955: 59 09     179             DA   PFXLIST
0957: F0 03     180             BEQ  GOTPFX     "always"!
0959: 01        181  PFXLIST    DFB  01         1 Parameter
095A: 7F 02     182             DA   PATH-1     Ziel: $027F
                183  *
095C: AE 7F 02  184  GOTPFX     LDX  PATH-1     Gesamtlänge
095F: A0 00     185  GOTPATH    LDY  #00
0961: B9 00 02  186  COPYPATH   LDA  $0200,Y    Path wird von $200
0964: 29 7F     187             AND  #$7F       nach PATH kopiert
0966: C9 0D     188             CMP  #$0D       Path-Ende?
0968: F0 0D     189             BEQ  PATHSET
096A: C9 60     190             CMP  #$60       Kleinbuchstabe?
096C: 90 02     191             BCC  ISUC
096E: 29 5F     192             AND  #$5F       Umwandlung in groß
0970: 9D 80 02  193  ISUC       STA  PATH,X     bzw. an das PREFIX
0973: E8        194             INX             angehängt
0974: C8        195             INY
0975: D0 EA     196             BNE  COPYPATH
0977: 8E 7F 02  197  PATHSET    STX  PATH-1     Gesamtlänge
097A: 60        198             RTS
                199  *
097B: A2 00     200  SETNUM     LDX  #00        wandelt ASCII-String
097D: 8E 42 09  201             STX  BLWANT     in BLOCKS_WANTED um
0980: 8E 43 09  202             STX  BLWANT+1
0983: BD 00 02  203  NXTNUM     LDA  $0200,X
0986: C9 8D     204             CMP  #$8D       <CR>?
0988: F0 29     205             BEQ  NUMSET     Zahl zu Ende
098A: C9 E0     206             CMP  #$E0
098C: 90 02     207             BCC  ISCAPS
098E: 29 DF     208             AND  #$DF       Umwandlung in groß
0990: A0 04     209  ISCAPS     LDY  #04
0992: 38        210             SEC
0993: E9 B0     211             SBC  #$B0       minus "0"
0995: 90 26     212             BCC  BADNUM     keine Zahl
0997: C9 0A     213             CMP  #$0A
0999: 90 06     214             BCC  ISNUM      "0".."9"
099B: E9 07     215             SBC  #$07
099D: C9 10     216             CMP  #$10       "A".."F"?
099F: B0 1C     217             BCS  BADNUM     nein
09A1: 0E 42 09  218  ISNUM      ASL  BLWANT     alte Zahl * 16
09A4: 2E 43 09  219             ROL  BLWANT+1
09A7: 88        220             DEY
09A8: D0 F7     221             BNE  ISNUM
09AA: 0D 42 09  222             ORA  BLWANT     neue Zahl dazu
09AD: 8D 42 09  223             STA  BLWANT
09B0: E8        224             INX
09B1: D0 D0     225             BNE  NXTNUM
                226  *
09B3: AD 42 09  227  NUMSET     LDA  BLWANT
09B6: 0D 43 09  228             ORA  BLWANT+1   Zahl > 00 00?
09B9: F0 02     229             BEQ  BADNUM
09BB: 18        230             CLC
09BC: 60        231             RTS
09BD: 38        232  BADNUM     SEC
09BE: 60        233             RTS
                234  *
09BF: 20 00 BF  235  TSTFILE    JSR  $BF00      GET FILE INFO
09C2: C4        236             DFB  $C4        für das neue SubDIR
09C3: 14 0A     237             DA   INFOLIST
09C5: C9 46     238             CMP  #$46       FILE NOT FOUND?
09C7: F0 11     239             BEQ  TSTEND
09C9: C9 00     240             CMP  #$00       File gefunden?
09CB: F0 0B     241             BEQ  TSTZ       ja, ist auch o.k.
09CD: E9 44     242             SBC  #$44       Fehler im
09CF: C9 02     243             CMP  #$02       Bereich $44-45?
09D1: 90 05     244             BCC  TSTZ       ja, PATH NOT FOUND
09D3: 69 43     245             ADC  #$43       => ERR# für MLIERR
09D5: 4C EA 0B  246             JMP  MLIERR
09D8: A0 FF     247  TSTZ       LDY  #$FF       löscht Z-Flag
09DA: 60        248  TSTEND     RTS
                249  *
09DB: AD 7F 02  250  GETROOM    LDA  PATH-1     Gesamtlänge des
09DE: 48        251             PHA             Path speichern
09DF: A2 00     252             LDX  #00
09E1: E8        253  GETVDIR    INX             und nur das VolDIR
09E2: BD 80 02  254             LDA  PATH,X     herausholen
09E5: C9 2F     255             CMP  #'/'       Ende erster Name?
09E7: D0 F8     256             BNE  GETVDIR
09E9: 8E 7F 02  257             STX  PATH-1     nur VolDIR-Name
09EC: 20 00 BF  258             JSR  $BF00
09EF: C4        259             DFB  $C4        GET FILE INFO
09F0: 14 0A     260             DA   INFOLIST   für VolDIR
09F2: F0 03     261             BEQ  GOTVDIR
09F4: 4C EA 0B  262             JMP  MLIERR     "MLI-Fehler: $xx"
09F7: 68        263  GOTVDIR    PLA
09F8: 8D 7F 02  264             STA  PATH-1     RESTORE des Path
09FB: AD 19 0A  265             LDA  AUXRTN     TOTAL_BLOCKS auf
09FE: 38        266             SEC             dem Volume minus
09FF: ED 1C 0A  267             SBC  BLUSED     belegte Blocks auf
0A02: AA        268             TAX             dem Volume ergibt
0A03: AD 1A 0A  269             LDA  AUXRTN+1   BLOCKS_FREE
0A06: ED 1D 0A  270             SBC  BLUSED+1
```

```
0A09: CD 45 09  271             CMP  BLNEED+1   benötigte Blocks
0A0C: 90 05     272             BCC  ROOMGOT    kein Platz!
0A0E: D0 03     273             BNE  ROOMGOT    o.k.
0A10: EC 44 09  274             CPX  BLNEED
0A13: 60        275  ROOMGOT    RTS             C = 0, => kein Platz
                276  *
0A14: 0A        277  INFOLIST   DFB  $0A        10 Parameter
0A15: 7F 02     278             DA   PATH-1     Pathname
                279             DS   2          ACCESS, File Type
                280  AUXRTN     DS   2          AUX_INFO
                281  STYPE      DS   1          Storage Type
                282  BLUSED     DS   2          BLOCKS_USED
                283             DS   8          DATE & TIME
                284  *
0A26: 20 00 BF  285  MAKEDIR    JSR  $BF00
0A29: C0        286             DFB  $C0        CREATE des SubDIR
0A2A: 32 0A     287             DA   CRLIST
0A2C: F0 03     288             BEQ  MKDONE
0A2E: 4C EA 0B  289             JMP  MLIERR
0A31: 60        290  MKDONE     RTS
                291  *
0A32: 07        292  CRLIST     DFB  $07        7 Parameter
0A33: 7F 02     293             DA   PATH-1     Pathname
0A35: C3        294             DFB  $C3        ACCESS
0A36: 0F        295             DFB  $0F        File Type: DIR
0A37: 00 00     296             DA   $0000     AUX_TYPE
0A39: 0D        297             DFB  $0D        Storage Type: SubDIR
0A3A: 00 00 00  298             DFB  0,0,0,0    DATE & TIME
0A3D: 00
                299  *
0A3E: AE 7F 02  300  GETFILE    LDX  PATH-1     Gesamtlänge
0A41: A0 FF     301  RPL1       LDY  #$FF
0A43: CA        302  RPL2       DEX             der Path wird von
0A44: 30 0E     303             BMI  SCANDIRS   "/xxx/yyyy" in
0A46: C8        304             INY             die Form:
0A47: BD 80 02  305             LDA  PATH,X     <03>xxx<04>yyyy
0A4A: C9 2F     306             CMP  #'/'       überführt, d.h.
0A4C: D0 F5     307             BNE  RPL2       alle "/" im Path
0A4E: 98        308             TYA             werden durch Längen-
0A4F: 9D 80 02  309             STA  PATH,X     bytes ersetzt
0A52: D0 ED     310             BNE  RPL1
                311  *
0A54: AE 80 02  312  SCANDIRS   LDX  PATH+0     Start mit dem zweiten
0A57: E8        313             INX             Namen im Path
0A58: 8E 0B 0B  314             STX  PATHIDX    (nach dem VolDIR)
0A5B: A0 02     315             LDY  #$02       und dem ersten Block
0A5D: A9 00     316             LDA  #$00       des VolDIR
0A5F: 20 10 0B  317             JSR  RDBLOCK    READ Block $0002
0A62: AD 28 10  318             LDA  DIRBUF+$28
0A65: AC 27 10  319             LDY  DIRBUF+$27
0A68: 8D 7D 0B  320             STA  VBMNO+1    Blocknummer des
0A6B: 8C 7C 0B  321             STY  VBMNO      ersten VBM-Blocks
                322  *
0A6E: A2 80     323  DIRSET     LDX  #$80       Flag: erster Block
0A70: 8E 0C 0B  324             STX  BLOCK1     wird gesetzt
0A73: 20 9C 0A  325  DOSCAN     JSR  SCAN       sucht nach Path-Namen
0A76: 90 0B     326             BCC  NXTDIR     dieser Name gefunden
0A78: AD 03 10  327             LDA  DIRBUF+3   FORWARD REF: Setzen
0A7B: AC 02 10  328             LDY  DIRBUF+2   des nächsten DIR-Bl.
0A7E: 20 10 0B  329             JSR  RDBLOCK    liest den Block
0A81: F0 F0     330             BEQ  DOSCAN     "always"
                331  *
0A83: AE 0B 0B  332  NXTDIR     LDX  PATHIDX    war dieser Name
0A86: 8A        333             TXA             der letzte im Path?
0A87: 38        334             SEC
0A88: 7D 80 02  335             ADC  PATH,X     Index a. nächst. Namen
0A8B: CD 7F 02  336             CMP  PATH-1     = Gesamtlänge?
0A8E: B0 0B     337             BCS  GOTFILE
0A90: 8D 0B 0B  338             STA  PATHIDX    nein, nächster Name
0A93: 20 00 0B  339             JSR  SETBLOCK1  im Path. Der erste Bl.
0A96: 20 10 0B  340             JSR  RDBLOCK    des hier gefund. DIR
0A99: F0 D3     341             BEQ  DIRSET     wird gelesen => Loop
0A9B: 60        342  GOTFILE    RTS             mit PTR a. neu. SubDIR
                343  *
0A9C: A9 10     344  SCAN       LDA  #>DIRBUF   sucht einen DIR-Block
0A9E: 85 07     345             STA  PTR+1      nach einem Namen im
0AA0: A9 04     346             LDA  #$04       Path a. File-Eintr. ab
0AA2: 18        347             CLC
0AA3: 2C 0C 0B  348             BIT  BLOCK1     1. Block dieses DIR?
0AA6: 10 12     349             BPL  NOT1ST     nein
0AA8: AC 24 10  350             LDY  DIRBUF+$24
0AAB: 8C 0D 0B  351             STY  ENUMBER    Einträge pro Block
0AAE: AC 23 10  352             LDY  DIRBUF+$23
0AB1: 8C 0E 0B  353             STY  ELENGTH    Länge eines Eintrags
0AB4: 6D 0E 0B  354             ADC  ELENGTH    HEADER überspringen
0AB7: 4E 0C 0B  355             LSR  BLOCK1     Flag zurück
                356  *
0ABA: AC 0D 0B  357  NOT1ST     LDY  ENUMBER    Einträge/Block
0ABD: 8C 0F 0B  358             STY  ECOUNT     Herunterzähler
                359  *
0AC0: 85 06     360  NXTENTRY   STA  PTR
0AC2: A5 07     361             LDA  PTR+1      PTR auf ersten/
```

```
ØAC4: 69 ØØ     362             ADC  #ØØ        nächsten Eintrag
ØAC6: 85 Ø7     363             STA  PTR+1      im Block
ØAC8: 2Ø DD ØA  364             JSR  CMPNAME    vergl. Eintrag & Path
ØACB: 9Ø ØF     365             BCC  GOTNAME    gefunden!
ØACD: CE ØF ØB  366             DEC  ECOUNT     noch Einträge übrig?
ØADØ: FØ Ø9     367             BEQ  NOTFOUND   nein, Block-Ende
ØAD2: A5 Ø6     368             LDA  PTR
ØAD4: 18        369             CLC             PTR auf nächst. Eintr.
ØAD5: 6D ØE ØB  37Ø             ADC  ELENGTH
ØAD8: 4C CØ ØA  371             JMP  NXTENTRY
                372 *
ØADB: 38        373 NOTFOUND    SEC
ØADC: 6Ø        374 GOTNAME     RTS
                375 *
ØADD: AØ ØØ     376 CMPNAME     LDY  #ØØ
ØADF: AE ØB ØB  377             LDX  PATHIDX    Index z. moment. Namen
ØAE2: B1 Ø6     378             LDA  (PTR),Y    l.Byte v. File-Eintrag
ØAE4: 29 ØF     379             AND  #$ØF       => Namenslänge
ØAE6: DD 8Ø Ø2  38Ø             CMP  PATH,X     = Path-Name?
ØAE9: DØ 13     381             BNE  NOTSAME
ØAEB: A8        382             TAY
ØAEC: 18        383             CLC
ØAED: 6D ØB ØB  384             ADC  PATHIDX    Start des Vergleichs
ØAFØ: AA        385             TAX             von hinten
ØAF1: B1 Ø6     386 CMP1        LDA  (PTR),Y
ØAF3: DD 8Ø Ø2  387             CMP  PATH,X
ØAF6: DØ Ø6     388             BNE  NOTSAME
ØAF8: CA        389             DEX
ØAF9: 88        39Ø             DEY
ØAFA: DØ F5     391             BNE  CMP1
ØAFC: 18        392             CLC             Namen sind gleich
ØAFD: 6Ø        393             RTS
ØAFE: 38        394 NOTSAME     SEC
ØAFF: 6Ø        395             RTS
                396 *
ØBØØ: AØ 12     397 SETBLOCK1   LDY  #$12       holt die Nummer
ØBØ2: B1 Ø6     398             LDA  (PTR),Y    des ersten Blocks
ØBØ4: 48        399             PHA             im File aus dem
ØBØ5: 88        4ØØ             DEY             File-Eintrag nach
ØBØ6: B1 Ø6     4Ø1             LDA  (PTR),Y    A-Y
ØBØ8: A8        4Ø2             TAY
ØBØ9: 68        4Ø3             PLA
ØBØA: 6Ø        4Ø4             RTS
                4Ø5 *
                4Ø6 PATHIDX     DS   1          Index in PATH
                4Ø7 BLOCK1      DS   1          Flag:$8Ø, b. 1.DIR-Bl.
                4Ø8 ENUMBER     DS   1          Anzahl Eintr./DIR-Bl.
                4Ø9 ELENGTH     DS   1          Länge eines Eintrags
                41Ø ECOUNT      DS   1          Zähler im DIR-Block
                411 *
ØB1Ø: 8D 26 ØB  412 RDBLOCK     STA  RWBLOCK+1  Aufruf mit
ØB13: 8C 25 ØB  413             STY  RWBLOCK    Blocknr. in A-Y
ØB16: 2Ø ØØ BF  414             JSR  $BFØØ
ØB19: 8Ø        415             DFB  $8Ø        READ BLOCK
ØB1A: 21 ØB     416             DA   RWLIST
ØB1C: FØ 14     417             BEQ  RWDONE
ØB1E: 4C EA ØB  418             JMP  MLIERR     "MLI-Fehler: $xx"
                419 *
ØB21: Ø3        42Ø RWLIST      DFB  $Ø3        3 Parameter
                421 RWUNIT      DS   1          Unit-Nummer
ØB23: ØØ 1Ø     422             DA   DIRBUF     Ziel/Quelle: DIRBUF
                423 RWBLOCK     DS   2          Blocknummer
                424 *
ØB27: 2Ø ØØ BF  425 MLIWRITE    JSR  $BFØØ
ØB2A: 81        426             DFB  $81        WRITE BLOCK
ØB2B: 21 ØB     427             DA   RWLIST     Blocknr. ist gesetzt!
ØB2D: FØ Ø3     428             BEQ  RWDONE
ØB2F: 4C EA ØB  429             JMP  MLIERR     "MLI-Fehler: $xx"
ØB32: 6Ø        43Ø RWDONE      RTS
                431 *
ØB33: A9 ØØ     432 READVBM     LDA  #ØØ
ØB35: 8D 78 ØB  433             STA  EXTENT     VBM-Block Nummer 1
ØB38: 8D 7B ØB  434             STA  ALTERED    "unverändert"
                435 *
ØB3B: AD 7C ØB  436 GETVBM      LDA  VBMNO      Nummer l.VBM-Block
ØB3E: 18        437             CLC
ØB3F: 6D 78 ØB  438             ADC  EXTENT     "x-ter" VBM-Block
ØB42: 8D 76 ØB  439             STA  VBLOCK
ØB45: AD 7D ØB  44Ø             LDA  VBMNO+1
ØB48: 69 ØØ     441             ADC  #ØØ
ØB4A: 8D 77 ØB  442             STA  VBLOCK+1   Blocknummer Hi
ØB4D: AD 22 ØB  443             LDA  RWUNIT
ØB5Ø: 8D 73 ØB  444             STA  VUNIT      Unit-Nummer
                445 *
ØB53: 2Ø ØØ BF  446             JSR  $BFØØ
ØB56: 8Ø        447             DFB  $8Ø        READ BLOCK
ØB57: 72 ØB     448             DA   VBMLIST
ØB59: FØ 16     449             BEQ  GOTVBM
ØB5B: 4C EA ØB  45Ø             JMP  MLIERR     "MLI-Fehler: $xx"
                451 *
ØB5E: 2C 7B ØB  452 WRITEVBM    BIT  ALTERED    "verändert"?
ØB61: 1Ø ØE     453             BPL  GOTVBM     nein, kein REWRITE
ØB63: 4E 7B ØB  454             LSR  ALTERED    sonst Flag zurück
ØB66: 2Ø ØØ BF  455             JSR  $BFØØ
ØB69: 81        456             DFB  $81        BLOCK WRITE
ØB6A: 72 ØB     457             DA   VBMLIST
ØB6C: FØ Ø3     458             BEQ  GOTVBM
ØB6E: 4C EA ØB  459             JMP  MLIERR     "MLI-Fehler: $xx"
                46Ø *
ØB71: 6Ø        461 GOTVBM      RTS
                462 *
ØB72: Ø3        463 VBMLIST     DFB  $Ø3        3 Parameter
                464 VUNIT       DS   1          Unit-Nummer
ØB74: ØØ 12     465             DA   VBMBUF     Ziel/Quelle: VBMBUF
                466 VBLOCK      DS   2          Blocknummer
                467 *
                468 EXTENT      DS   1          "x-ter" VBM-Block
                469 PAGE        DS   1          Seite im VBM-Block
                47Ø INBYTE      DS   1          Index in e. VBM-Byte
                471 *
                472 ALTERED     DS   1          $8Ø: VBM-Block veränd.
                473 VBMNO       DS   2          Blocknr. l. VBM-Block
                474 *
ØB7E: AØ ØØ     475 GNBLOCK     LDY  #ØØ        belegt den nächsten
ØB8Ø: 8C 79 ØB  476             STY  PAGE       freien Bl. und returnt
ØB83: 84 Ø6     477             STY  PTR        Blocknr. in A-Y
ØB85: A9 12     478             LDA  #>VBMBUF   Adresse Hi
ØB87: 85 Ø7     479             STA  PTR+1
                48Ø *
ØB89: B1 Ø6     481 TSTBYTE     LDA  (PTR),Y
ØB8B: DØ 1A     482             BNE  GETBIT     mind. 1 Block frei!
ØB8D: C8        483             INY
ØB8E: DØ F9     484             BNE  TSTBYTE
ØB9Ø: E6 Ø7     485             INC  PTR+1
ØB92: EE 79 ØB  486             INC  PAGE       Speicherseite-Zähler,
ØB95: AD 79 ØB  487             LDA  PAGE       wird für Bestimmung
ØB98: C9 Ø2     488             CMP  #Ø2        der Blocknr. benötigt
ØB9A: 9Ø ED     489             BCC  TSTBYTE
                49Ø *
ØB9C: 2Ø 5E ØB  491             JSR  WRITEVBM   REWRITE dieses VBM-Bl.
ØB9F: EE 78 ØB  492             INC  EXTENT     nächster VBM-Block
ØBA2: 2Ø 33 ØB  493             JSR  READVBM
ØBA5: FØ D7     494             BEQ  GNBLOCK    => Loop
                495 *
ØBA7: A2 ØØ     496 GETBIT      LDX  #Ø         holt ein gesetztes
ØBA9: 2A        497 GETB1       ROL             Bit aus dem Akku
ØBAA: BØ Ø3     498             BCS  GOTBIT     und zählt dabei mit
ØBAC: E8        499             INX
ØBAD: DØ FA     5ØØ             BNE  GETB1
ØBAF: 8E 7A ØB  5Ø1 GOTBIT      STX  INBYTE     Ø..7: Nummer des Bits
ØBB2: 18        5Ø2             CLC             dieses Bit löschen!
ØBB3: 6A        5Ø3 GOTB1       ROR             und Restore des Akku
ØBB4: CA        5Ø4             DEX
ØBB5: 1Ø FC     5Ø5             BPL  GOTB1
                5Ø6 *
ØBB7: 91 Ø6     5Ø7             STA  (PTR),Y    Block belegt in VBM
ØBB9: A9 8Ø     5Ø8             LDA  #$8Ø       dieser VBM-Block muß
ØBBB: 8D 7B ØB  5Ø9             STA  ALTERED    zurückgeschr. werden
                51Ø *
ØBBE: AD 78 ØB  511             LDA  EXTENT     "x-ter" VBM-Block:
ØBC1: ØA        512             ASL             Blocknummer * $1ØØØ
ØBC2: ØD 79 ØB  513             ORA  PAGE       "Seite" darin:
ØBC5: 8D DA ØB  514             STA  BLOCKHI    Blocknummer * $8ØØ
ØBC8: 98        515             TYA             Index in der Seite:
ØBC9: AØ Ø3     516             LDY  #Ø3        Blocknummer * $Ø8
ØBCB: ØA        517 X8          ASL
ØBCC: 2E DA ØB  518             ROL  BLOCKHI    insgesamt * 8
ØBCF: 88        519             DEY
ØBDØ: DØ F9     52Ø             BNE  X8
ØBD2: ØD 7A ØB  521             ORA  INBYTE     + Ø...7
ØBD5: A8        522             TAY             Blocknummer Low
ØBD6: AD DA ØB  523             LDA  BLOCKHI
ØBD9: 6Ø        524             RTS
                525 *
                526 BLOCKHI     DS   1          Scratch: Blocknr. Hi
                527 *
                528 ************
                529 *---EXIT---*
                53Ø ************
                531 *
ØBDB: A9 8D     532 EXIT        LDA  #$8D
ØBDD: 2Ø ED FD  533             JSR  COUT       2mal <CR>
ØBEØ: 2Ø ED FD  534             JSR  COUT
ØBE3: A9 ØØ     535             LDA  #$ØØ
ØBE5: 85 48     536             STA  $48        ???
ØBE7: 4C D3 Ø3  537             JMP  RESTART    DOS-Warmstart
                538 *
ØBEA: 48        539 MLIERR      PHA             Fehlernummer
ØBEB: A2 8A     54Ø             LDX  #DSKERR-MSGS
ØBED: 2Ø ØB ØC  541             JSR  PRTERR     "MLI-Fehler: $"
ØBFØ: 68        542             PLA
ØBF1: 2Ø DA FD  543             JSR  PRBYTE     ERR# in Hex
ØBF4: 4C DB ØB  544             JMP  EXIT
                545 *
```

```
ØBF7: A2 98      546  BADTYPE  LDX  #BADTXT-MSGS
ØBF9: DØ ØA      547           BNE  SETERR    "FILE TYPE MISMATCH"
                 548  *
ØBFB: A2 AB      549  BADLEN   LDX  #LENTXT-MSGS
ØBFD: DØ Ø6      55Ø           BNE  SETERR    "Verkürzung ..."
                 551  *
ØBFF: A2 C5      552  NOVOL    LDX  #BADVOL-MSGS
ØCØ1: DØ Ø2      553           BNE  SETERR    "PATH NOT FOUND"
                 554  *
ØCØ3: A2 D4      555  NOROOM   LDX  #VOLFULL-MSGS
                 556  *
ØCØ5: 2Ø ØB ØC   557  SETERR   JSR  PRTERR
ØCØ8: 4C DB ØB   558          JMP  EXIT
                 559  *
ØCØB: A9 87      56Ø  PRTERR   LDA  #$87      BELL
ØCØD: 2Ø ED FD   561          JSR  COUT
                 562  *
ØC1Ø: A9 8D      563  PRTMSG   LDA  #$8D      2mal <CR>
ØC12: 2Ø ED FD   564          JSR  COUT
ØC15: 2Ø ED FD   565          JSR  COUT
ØC18: BD 26 ØC   566  PRT1     LDA  MSGS,X
ØC1B: FØ Ø8      567          BEQ  PRDONE    <ØØ>: Text-Ende
ØC1D: Ø9 8Ø      568          ORA  #$8Ø
ØC1F: 2Ø ED FD   569          JSR  COUT
ØC22: E8         57Ø          INX
ØC23: DØ F3      571          BNE  PRT1      "always"
ØC25: 6Ø         572  PRDONE   RTS
                 573  *
ØC26: 45 72 73   574  MSGS     ASC  'Erstellung von Subdirectories'
ØC29: 74 65 6C 6C 75 6E 67 2Ø
ØC31: 76 6F 6E 2Ø 53 75 62 64
ØC39: 69 72 65 63 74 6F 72 69
ØC41: 65 73
ØC43: 8D 8D      575          DFB  $8D,$8D
ØC45: 6D 69 74   576          ASC  'mit definierter Block-Anzahl'
ØC48: 2Ø 64 65 66 69 6E 69 65
ØC5Ø: 72 74 65 72 2Ø 42 6C 6F
ØC58: 63 6B 2D 41 6E 7A 61 68
ØC6Ø: 6C
ØC61: 8D 8D ØØ   577          DFB  $8D,$8D,ØØ
ØC64: 45 69 6E   578          ASC  'Eingabe Filename: '
ØC67: 67 61 62 65 2Ø 46 69 6C
ØC6F: 65 6E 61 6D 65 3A 2Ø
ØC76: ØØ         579          DFB  ØØ
                 58Ø  *
ØC77: 42 6C 6F   581  GETLEN   ASC  'Block-Anzahl (Hex): '
ØC7A: 63 6B 2D 41 6E 7A 61 68
ØC82: 6C 2Ø 28 48 65 78 29 3A
ØC8A: 2Ø
ØC8B: ØØ         582          DFB  ØØ
                 583  *
ØC8C: 3C 43 52   584  START    ASC  '<CR>=Start  <ESC>=Abbruch'
ØC8F: 3E 3D 53 74 61 72 74 2Ø
ØC97: 2Ø 3C 45 53 43 3E 3D 41
ØC9F: 62 62 72 75 63 68
ØCA5: ØØ         585          DFB  ØØ
                 586  *
ØCA6: 2D 2Ø 46   587  DONMSG   ASC  '- Fertig.'
ØCA9: 65 72 74 69 67 2E
ØCAF: ØØ         588          DFB  ØØ
                 589  *
ØCBØ: 4D 4C 49   59Ø  DSKERR   ASC  'MLI-Fehler: $'
ØCB3: 2D 46 65 68 6C 65 72 3A
ØCBB: 2Ø 24
ØCBD: ØØ         591          DFB  ØØ
ØCBE: 46 49 4C   592  BADTXT   ASC  'FILE TYPE MISMATCH'
ØCC1: 45 2Ø 54 59 5Ø 45 2Ø 4D
ØCC9: 49 53 4D 41 54 43 48
ØCDØ: ØØ         593          DFB  ØØ
ØCD1: 56 65 72   594  LENTXT   ASC  'Verkürzung nicht möglich!'
ØCD4: 6B 7D 72 7A 75 6E 67 2Ø
ØCDC: 6E 69 63 68 74 2Ø 6D 7C
ØCE4: 67 6C 69 63 68 21
ØCEA: ØØ         595          DFB  ØØ
ØCEB: 5Ø 41 54   596  BADVOL   ASC  'PATH NOT FOUND'
ØCEE: 48 2Ø 4E 4F 54 2Ø 46 4F
ØCF6: 55 4E 44
ØCF9: ØØ         597          DFB  ØØ
ØCFA: 56 4F 4C   598  VOLFULL  ASC  'VOLUME FULL'
ØCFD: 55 4D 45 2Ø 46 55 4C 4C
ØDØ5: ØØ         599          DFB  ØØ
                 6ØØ  *
                 6Ø1  DIRBUF   EQU  $1ØØØ
                 6Ø2  VBMBUF   EQU  DIRBUF+$2ØØ

1283 Bytes
```

Ulrich Stiehl

Kyan-Pascal

Datentypen, Unterprogramme, Strings, Dateien

Dieser Aufbaukurs, der sich nahtlos an den Grundkurs anschließt, befaßt sich mit Datentypen, Unterprogrammen, Strings und Dateien, wobei als neue Befehlswörter eingeführt werden:
Datentypen: TYPE; BOOLEAN, FALSE, TRUE; ODD, CHR, ORD; SET, IN; ARRAY, RECORD
Unterprogramme: LABEL, GOTO; PROCEDURE, FUNCTION
Strings: String; #I; Length, Concat, Pos, Copy, Delete und Insert (allesamt benutzerdefiniert)
Dateien: FILE, TEXT, REWRITE, RESET, GET, PUT, SEEK, READLN, WRITELN, EOF; INPUT, OUTPUT
Die Befehlswörter WITH, PACKED, FORWARD, EOLN, PAGE, PR, CHAIN, NEW, DISPOSE und NIL sind aus Platzgründen ausgespart worden. Statt dessen wurden die Standard-Datentypen REAL usw. nochmals vertiefend beschrieben. Die Bereiche „Kyan-Pascal und Assembler" und „Hochauflösende Grafik in Kyan-Pascal" werden in gesonderten Aufsätzen behandelt.
Insgesamt umfaßt der zweiteilige Kyan-Kurs über 110.000 Anschläge und ersetzt damit das englischsprachige Kyan-Manual.

1. Datentypen

Ein Computer dient per definitionem der Verarbeitung von Daten (= Datenverarbeitungsanlage). Zunächst wurden als Daten nur Zahlen verarbeitet, wie dies heute noch bei Taschenrechnern der Fall ist. Dann kamen Zeichen und Zeichenketten hinzu, und nunmehr werden auch grafische Gebilde (Grafiken, Plot-Vektoren usw.) als Daten verarbeitet. Der Vorteil von Pascal gegenüber BASIC liegt nicht in der Vielfalt der Datentypen, sondern in deren müheloser Bildung oder Konstruktion.

Datentypen können im VAR-Teil implizit nach dem Doppelpunkt „:" oder im TYPE-Teil explizit nach dem Gleichheitszeichen „=" definiert werden. Besonders bei den komplexen Datentypen (Array, Record) ist die explizite Definition im TYPE-Teil vorzuziehen. Die im TYPE-Teil definierten Typen können auf die Konstanten des vorangehenden CONST-Teils und die im VAR-Teil definierten Variablen auf die Typen des vorangehenden TYPE-Teils Bezug nehmen. Vereinfachtes Beispiel:

```
PROGRAM DEMO1A;
VAR
 I: INTEGER; {implizit}
BEGIN
 I := 10; WRITELN (I)
END.

PROGRAM DEMO1B;
TYPE
 INT = INTEGER; {explizit}
VAR
 I: INT;
BEGIN
 I := 10; WRITELN (I)
END.
```

Mit der Definition (hier lediglich Umbenennung) eines Datentyps ist es jedoch noch nicht getan. Vielmehr muß man auch wissen, in welcher Form man diese Daten verarbeiten kann. Leider stellt das Kyan-Manual die Datentypen nackt in den Raum, ohne zu präzisieren, welche Befehle für sie implementiert wurden. Daß man beispielsweise mit WRITELN einen ganzen Zeichen-Array, nicht aber einen ganzen Zahlen-Array ausgeben kann, wird nicht besonders vermerkt. In Wirklichkeit mußte in Kyan-Pascal für die Ausgabe eines ganzen Zeichen-Arrays eine gesonderte Ausgabe-Routine implementiert werden. Es wird jedoch verschwiegen, daß die WRITELN-Routine nur beim
String = ARRAY [1..255] OF CHAR
und nicht etwa beim
String = ARRAY [1..16384] OF CHAR
funktioniert. (Das High-Byte fällt unter den Tisch!)

Exkurs „vom Typ"
In der Informatik hat sich die abwegige Redewendung „vom Typ Boolean, vom Typ Real, vom Typ Integer" usw. breitgemacht (s. Bücher von N.Wirth und Epigonen), die offenbar dieser Disziplin einen wissenschaftlichen Anstrich, nein, einen Anstrich vom Typ Wissenschaft verleihen soll. Zu sagen, „1.5 ist eine Dezimalzahl", wäre eine Formulierung, die jeder versteht. Das darf nicht sein! Da klingt es schon viel besser, wenn man schreibt: „1.5 ist eine Konstante vom Typ Real". Würde ein Wirth-Epigone im Restaurant schlicht „Berner Rösti" bestellen, so wäre dies eine Rede vom Typ Mensch. Und der Respekt des Volkes wäre dahin. Unmöglich! Daher pflegt der wahre Informatiker zu ordern: „Bitte eine finite Menge Rösti vom Typ Bern", worauf ein gewitzter Kellner kontern könnte: „Diese werden nur in Häusern vom Typ Toll serviert!"
Wohlgemerkt: Ich störe mich nicht an dem Fremdwort „Typ" oder an der „von"-Konstruktion. Ich kritisiere nur die Verwissenschaftlichung einer Trivialität. Denn „Datentyp" heißt nichts anderes als „Datenart" und „Typ" nichts anderes als „Art". Betrachten wir hierzu das Wirth-Zitat aus „Spektrum der Wissenschaft", Sonderheft Computer-Software, 1985, S. 20:
„Sind alle Komponenten von gleichem Typ, so heißt der zusammengesetzte Typ homogen".
Übersetzt man „Komponenten" mit „Teilen", „von gleichem Typ" mit „gleichartig" und „homogen" (homos = gleich, genos = Art)" ebenfalls mit „gleichartig", dann entsteht: *„Sind alle Teile gleichartig, so heißt die zusammengesetzte Art gleichartig"*, was offenbar Unfug ist.
Übersetzt man „homogen" mit „aus gleichartigen Teilen zusammengesetzt", dann entsteht: *„Sind alle Teile gleichartig, so heißt die zusammengesetzte Art aus gleichartigen Teilen zusammengesetzt"*, was offenkundig eine Tautologie ist.
Auf diese Weise produzieren „Datentypisten", d.h. auf Datentypen spezialisierte Informatiker, mit gespreizten Redewendungen schiere Banalitäten.

Meist gliedert man die Pascal-Datentypen in statische und dynamische Typen (Zeiger), wobei die statischen Typen sich weiter unterteilen in skalare und strukturierte Typen. Zu den skalaren Typen gehören die Standard-Typen (Integer-, Real-, Char- und Boolean-Typen) sowie die benutzerdefinierten Typen (Aufzählungs- und Ausschnittstypen), und zu den strukturierten Typen gehören die Record-, Array-, String-, Set- und File-Typen.
Aus praktischen und didaktischen Gründen haben wir eine etwas andere Gliederung gewählt:
Einfache Typen: Ganzzahl, Dezimalzahl und Einzelzeichen.
Spezielle Typen: Wahrheitswert, Aufzählung, Bereich und Menge.
Komplexe Typen: Datensatz, Feld, Zeichenkette und Datei.
Auf die Darstellung von Zeiger-Typen wurde aus Platzgründen verzichtet.

1.1. Einfache Typen

Ganzzahl = Integer-Typ
VAR I:INTEGER
Dezimalzahl = Real-Typ
VAR R: REAL
Einzelzeichen = Char-Typ
VAR C: CHAR

1.1.1. Ganzzahl (Integer)

Integer-Zahlen sind Ganzzahlen im Bereich -32767 bis +32767, die intern 2 Bytes belegen. Sie können als Konstanten definiert werden, z.B.
CONST IK = 10;
wobei für den maximalen Integer-Wert auch die Systemkonstante MAXINT verwendet werden kann. Die Vorzeichen
+, -
dürfen nur bei der Konstantendefinition und der *einfachen* Wertzuweisung, z.B.
IK = -MAXINT; oder
I := -10;
nicht jedoch bei Ausdrücken bzw. Termen, z.B.
I := -IK + -IK; {falsch}
verwendet werden. Notfalls muß man einklammern:
I := (-IK) + (-IK);
Einfache Integer-Variablen definiert man üblicherweise im VAR-Teil mit dem Typ-Bezeichner INTEGER, z.B.
VAR I:INTEGER;
Für Integer-Zahlen sind die mathematischen Operatoren
+, -, *, DIV, MOD,
die Vergleichsoperatoren
<, =, >, <=, >=, <>,
die Funktionen
SQR, ABS,
ROUND, TRUNC,
die „skalaren" Funktionen
PRED, SUCC, ORD
sowie die Zuweisung
:=
implementiert. Mit den Befehlen
I := ROUND (R); und
I := TRUNC (R);
kann eine Real-Zahl im Bereich -32767 bis +32767 in eine Integer-Zahl umgewandelt werden. Einer Real-Variablen kann eine Integer-Variable zugewiesen werden, z.B.
R := I + I; {zulässig}
nicht jedoch umgekehrt
I := R + R; {verboten}
es sei denn indirekt über ROUND und TRUNC, z.B.
I := ROUND (R) + ROUND (R);
Die Funktion **ODD** (odd = ungerade; Geg.: even = gerade) hat als Argument eine Integer-Variable und als Funktionsergebnis eine boolesche Variable (s.u). Beispiel:

```
PROGRAM DEMO2;
VAR
 B: BOOLEAN;
 I: INTEGER;
BEGIN
I := 3; B := ODD (I); {B = TRUE}
IF B = TRUE THEN WRITELN ('ungerade');
I := 2; B := ODD (I); {B = FALSE}
IF B = FALSE THEN WRITELN ('gerade')
END.
```

Die Befehle READLN und WRITE/WRITELN sind uneingeschränkt auf Integer-Variablen anwendbar, z.B.
READLN (I); {nicht READ (I)}
WRITELN (I); {auch WRITE (I)}
READLN konvertiert automatisch die eingegebene ASCII-Ziffernfolge (Zahlenstring) in die interne Integer-Form (2 Bytes), während WRITELN die interne Integer-Form wieder als Ziffernfolge sichtbar macht. Befehle für die Umwandlung einer Integer-Zahl in einen Zahlenstring und umgekehrt sind nicht vorgesehen.

1.1.2. Dezimalzahl (Real)

Real-Zahlen sind Dezimalzahlen bzw. Fließkomma-Zahlen im Bereich -1.0E99 bis

+1.0E99, die intern 8 Bytes belegen (BCD-Darstellung mit einer 13stelligen Mantisse). Sie können als Konstanten definiert werden, z.B.
CONST RK = 9.9E+99;
Eine Systemkonstante („MAXREAL" analog zu MAXINT) fehlt. Für die Vorzeichen
+, -
gelten die obigen Ausführungen analog.
Einfache Real-Variablen definiert man üblicherweise im VAR-Teil mit dem Typ-Bezeichner REAL, z.B.
VAR R: REAL;
Für Real-Zahlen sind die mathematischen Operatoren
+, -, *, /,
die Vergleichsoperatoren
<, =, >, <=, >=, <>,
die Funktionen
SIN, COS, ARCTAN, LN, EXP, SQRT,
SQR, ABS
sowie die Zuweisung
:=
implementiert. Die „skalaren" Funktionen PRED, SUCC und ORD funktionieren hier nicht. Die Funktionen SIN, COS, ARCTAN, LN, EXP und SQRT können als *Argument* (in runden Klammern) wahlweise einen Real- *oder* einen Integer-Ausdruck haben, doch liefern sie als *Ergebnis* (= Funktionswert) stets eine Real-Zahl. Die Funktionen SQR und ABS liefern ein Real-Ergebnis im Falle eines Real-Arguments und ein Integer-Ergebnis im Falle eines Integer-Arguments. Beispiel:

```
PROGRAM DEMO3;
VAR
 I, IE: INTEGER;
 R, RE: REAL;
BEGIN
R := 1.2;
I := 12;
RE := SIN (R); WRITELN (RE);
RE := SIN (I); WRITELN (RE);
RE := SQR (R); WRITELN (RE);
IE := SQR (I); WRITELN (IE)
END.
```

Die Befehle READLN (nicht READ!) und WRITE/WRITELN sind uneingeschränkt auf Real-Zahlen anwendbar, wobei READLN automatisch die eingegebene ASCII-Ziffernfolge (Zahlenstring) in die interne Real-Form (8 Bytes) konvertiert, während umgekehrt WRITELN die interne Real-Form wieder als Ziffernfolge sichtbar macht. Befehle zur Konvertierung von Integer-Zahlen in Real-Zahlen oder von Zahlenstrings in Real-Zahlen und umgekehrt sind nicht vorgesehen.

1.1.3. Einzelzeichen (Char)

Einzelzeichen („characters") sind ASCII-Zeichen (= Buchstaben, Ziffern, Sonder- und Steuerzeichen), die intern je 1 Byte belegen. Jedem ASCII-Zeichen ist eine Nummer von 0 bis 255 zugeordnet (s. ASCII-Tabelle in Peeker, 7/85, S. 34).
Einzelzeichen können als Konstanten in der Form
CONST CK = 'A';
festgelegt werden werden. Einzelzeichen-Variablen definiert man üblicherweise im VAR-Teil mit dem Typ-Bezeichner CHAR, z.B.
VAR C: CHAR;
Für Einzelzeichen sind die Vergleichsoperatoren
<, =, >, <=, >=, <>,
die Funktionen
SUCC, PRED,
die Konvertierungsfunktionen
CHR, ORD
und die Zuweisung
:=
implementiert. Die Befehle READ(LN) und WRITE(LN) sind mit gewissen Einschränkungen auf Einzelzeichen anwendbar.
Mit den Funktionen **CHR** und **ORD** lassen sich Integer-Char-Konvertierungen vornehmen (ORD = ordinal number = Ordnungszahl = hier ASCII-Nummer; CHR = character = hier ASCII-Zeichen).
I := ORD (C);
weist der Integer-Variablen I die ASCII-Nummer der Char-Variablen C zu.
C := CHR (I);
weist der Char-Variablen C das ASCII-Zeichen mit der ASCII-Nummer I zu.
Man beachte, daß sich die Zuweisung mit Apostroph
CV := 'a';
nur auf sichtbare ASCII-Zeichen mit Bit 7 off (ASCII-Nummern 32-127) erstreckt. Demgegenüber ist bei der Zuweisung mit Hilfe der CHR-Funktion *jedes* Einzelzeichen (ASCII-Nummern 0-255) definierbar. Die CHR-Funktion ist jedoch nicht bei der Konstantenfestlegung zulässig:
CONST CR = CHR (13); {verboten}
Eine ähnliche Einschränkung hinsichtlich der Steuerzeichen im ASCII-Bereich 0-31 ist bei der Tastatureingabe mit Hilfe des READ- bzw. READLN-Befehls gegeben. Man verwende deshalb die RDKEY-Funktion aus dem Grundkurs.
Beispiel:

```
PROGRAM DEMO4;
VAR
 C: CHAR;
 I: INTEGER;
BEGIN
C := 'B';
WRITELN (SUCC (C)); {'C'}
WRITELN (PRED (C)); {'A'}
C := 'A';
I := ORD (C); WRITELN (I); {65}
I := 65;
C := CHR (I); WRITELN (C); {'A'}
C := ''''; WRITELN (C); {Apostroph!}
C := 'Y';
IF C < 'Z' THEN WRITELN ('Y < Z');
FOR C := ' ' TO 'ß' DO WRITE (C);
WRITELN
END.
```

1.2. Spezielle Typen

Wahrheitswert = boolescher Typ
VAR B: BOOLEAN
Aufzählung = skalarer Namenstyp
TYPE NAME = (N1, N2, N3)
Bereich = Bereichstyp
TYPE BEREICH = ANFANG..ENDE
Menge = Set-Typ
TYPE MENGE = SET OF ANFANG..ENDE

1.2.1. Wahrheitswert (Boolean)

Die bereits bekannten Vergleiche mit
<, =, >, <=, >=, <> und
NOT, AND, OR, z.B.
IF (1 < 2) THEN...
erzeugen *implizit* Wahrheitswerte (entweder wahr oder falsch), die auch *explizit* einer booleschen Variablen zugewiesen werden können, z.B.
B := (1 < 2);
Ein Wahrheitswert belegt intern 1 Byte. Da es bereits die vordefinierten Systemkonstanten **TRUE** und **FALSE** gibt, sind selbstdefinierte boolesche Konstanten, z.B.
CONST BK = TRUE;
wenig sinnvoll.
Boolesche Variablen werden üblicherweise im VAR-Teil mit Hilfe des Typ-Bezeichners **BOOLEAN** definiert, z.B.

VAR B: BOOLEAN;
Für Wahrheitswerte sind die Vergleichsoperatoren
<, =, >, <=, >=, <>,
die logischen Operatoren
NOT, AND, OR,
die „skalaren" Funktionen
SUCC, PRED, ORD
und die Zuweisung
:=
implementiert. Die FILE-Befehle EOLN und EOF (s.u.) sowie der SET-Befehl IN (s.u.) liefern ebenfalls boolesche Resultate. Die direkte Ein- und Ausgabe von Wahrheitswerten über READLN und WRITELN ist in Kyan-Pascal *nicht* möglich.
Intern wird FALSE durch 0 und TRUE durch 1 (bzw. <> 0) dargestellt. Deshalb gilt: FALSE kommt vor TRUE, TRUE kommt nach FALSE. Trotzdem dürfen boolesche Ausdrücke nicht mit Integer- oder Real-Ausdrücken gemischt werden:
WRITELN 10 * (1 = 1); {verboten}
Anstelle von
IF B = TRUE THEN...
kann man auch
IF B THEN...
und anstelle von
... UNTIL B = FALSE;
kann man auch
... UNTIL NOT B;

verwenden. Beispiel:

```
PROGRAM DEMO5;
VAR
 B1, B2: BOOLEAN;
BEGIN
B1 := ('A' = 'A'); {TRUE}
B2 := ('A' = 'B'); {FALSE}
{knapper:}
IF B1 THEN WRITELN ('w');
IF NOT B2 THEN WRITELN ('f');
{klarer:}
IF B1 = TRUE THEN WRITELN ('w');
IF B2 = FALSE THEN WRITELN ('f');
B1 := PRED (TRUE); {FALSE}
B2 := SUCC (FALSE); {TRUE}
IF B1 < B2 THEN WRITELN ('f-w');
  {0 AND 0 = 0
   0 AND 1 = 0
   1 AND 0 = 0
   1 AND 1 = 1}
FOR B1 := FALSE TO TRUE DO
FOR B2 := FALSE TO TRUE DO
WRITELN (ORD (B1), ' AND ', ORD (B2),
        ' = ', ORD (B1 AND B2))
END.
```

Eine boolesche Variable läßt sich in eine Integer-Variable mit
I := ORD (B);
umwandeln. Umgekehrt kann eine Integer-Variable im Wertbereich 0 bis 1 mit
B := (I = 1);
in eine boolesche Variable konvertiert werden.

1.2.2. Aufzählung (Namenstyp)

Die Aufzählung oder der Namenstyp ist eine geordnete Folge von Namen N0, N1, N2... (*nicht* von Werten). Ein Aufzählungsname wird als skalar (von „Skala") bezeichnet, weil er intern als natürliche Zahl (N0 = 0, N1 = 1 usw.; 2 Bytes pro Name) gespeichert wird. Sinngemäß sind die skalaren Funktionen
PRED, SUCC, ORD,
die Vergleichsoperatoren
<, >, =, <=, >=, <>
und die Zuweisung
:=
auf Namenstypen anwendbar. Ein Aufzählungsname darf nicht im CONST-Teil festgelegt werden. READLN und WRITELN sowie sonstige Befehle sind nicht implementiert. Die Definition einer Aufzählung erfolgt im VAR-Teil, z.B.
VAR NAME: (N0, N1, N2);
oder häufiger im TYPE-Teil, z.B.
TYPE NAME = (N0, N1, N2);
VAR N: NAME;
wobei die Namen in *runde Klammern* gesetzt werden. Beispiel:

```
PROGRAM DEMO6;
TYPE
 ZAHL = (NULL, EINS, ZWEI);
VAR
 Z: ZAHL;
BEGIN
WRITELN (ORD (EINS));          {1}
WRITELN (ORD (SUCC (EINS))); {2}
WRITELN (ORD (PRED (EINS))); {0}
Z := NULL; WRITELN (ORD (Z)) {0}
END.
```

Als **skalare Typen** gelten neben den Namenstypen auch die Ganzzahlen, Einzelzeichen und Wahrheitswerte, nicht jedoch die Dezimalzahlen, da bei letzteren wegen möglicher Rundungsfehler Vorgänger (PRED) und Nachfolger (SUCC) nicht immer eindeutig angegeben werden können, denn in einem Bereich, z.B. 0.5 . . . 1.5, gibt es theoretisch unendlich viele Real-Zahlen.

Exkurs: Datentypnutzen
Persönlich halte ich die Aufzählung für einen weitgehend nutzlosen Datentyp. Wenn man z.B.
TYPE STERN = (SONNE, MOND);
definiert, so wird von der Tatsache abgelenkt, daß der Name SONNE gar nicht die Sonne, sondern einen computerinternen Wert (hier 0) bezeichnet. Bei infantiler Betrachtungsweise könnte man auf den Gedanken kommen, daß nach
WRITELN (SONNE);
auf dem Monitor die Sonne aufgeht, was natürlich nicht der Fall ist. Wenn man auf den skalaren Namenstyp nur mit PRED, SUCC und ORD zugreifen kann, ist der praktische Wert mehr als gering. Ein Datentyp ist erst dann von Nutzen, wenn für ihn *spezielle Befehle* implementiert wurden. Mit
TYPE NATZAHL = 0..255;
hat man noch keine Integer-Arithmetik für natürliche Zahlen im Bereich 0 bis 255 geschaffen. Sowohl der Aufzählungstyp als auch der folgende Bereichstyp dienen damit mehr pädagogischen als programmierpraktischen Zwecken.

1.2.3. Bereich (Anfang..Ende)

Der Bereichs- oder Ausschnittstyp ist ein Teilbereich aus einem Datentyp (Integer, Char, Aufzählung, nicht aber Real), für den dann dieselben Befehle wie für den gesamten Datentyp gelten. Anfangs- und Endwert des Teilbereichs werden durch zwei Punkte („..") begrenzt. Die Definition erfolgt implizit im VAR-Teil oder explizit im TYPE-Teil, z.B.:
VAR
INDEX: -30..+30;
VERSAL: 'A'..'Z';
Ein Bereich darf nicht im CONST-Teil definiert werden.
Der Bereichstyp spielt besonders bei der Indizierung (Indextyp, s.u.) eine Rolle. Man beachte, daß etwa der Typ
TYPE BYTE = 0..255;
VAR B: BYTE;
in Kyan-Pascal nur der Optik dient, denn die Variable B belegt nach wie vor intern 2 Bytes und wird damit wie eine gewöhnliche Integer-Zahl behandelt. Auch kann man beispielsweise bei
READLN (B);
eine größere Zahl als 255 eingeben, ohne daß eine Fehlermeldung erfolgt. Dies gilt auch für Turbo- und UCSD-Pascal. Lediglich die Überschreitung des Index-Bereichs eines Arrays wird in Kyan-Pascal mit einem Runtime-Error quittiert.

1.2.4. Menge (Set)

Kyan-Pascal verfügt über einen Mengentyp mit maximal 256 Elementen (intern 32 Bytes = 256 Bits). Die Definition einer Menge erfolgt meist im TYPE-Teil mit dem Typ-Bezeichner **SET OF**. Beispiel:
TYPE
MENGE1 = SET OF 1..10;
MENGE2 = SET OF INTEGER;
MENGE3 = SET OF (SONNE, MOND);
VAR
M1: MENGE1; M2: MENGE2; M3: MENGE3;
Die Elemente einer Menge können Ganzzahlen, Einzelzeichen, Wahrheitswerte oder Aufzählungsnamen sein (nicht gemischt und kein Element doppelt!). Bei SET OF INTEGER gelten nur die ersten 256 Ganzzahlen (0..255).
Bei der Wertzuweisung mit „:=" im eigentlichen Programm müssen die Elemente, deren Reihenfolge im Gegensatz zum Aufzählungstyp beliebig ist, in *eckige Klammern* gesetzt werden, z.B.
M1 := [7, 3, 2];
M2 := [100, 200];
M3 := [SONNE];
Eine Integer- oder Char-Menge kann im TYPE- oder VAR-Teil nicht durch Aufzählung der Elemente oder Bereiche definiert werden, weil nur im Programm selbst eckige Klammern erlaubt sind:
VAR M: 1, 10, 100; {falsch}
M := [1, 10, 100]; {richtig}
VAR M: 1..10, 20..30; {falsch}
M := [1..10, 20..30]; {richtig}
Die Eingabe/Ausgabe über READLN/WRITELN ist nicht vorgesehen. Von den zahlreichen Mengenbefehlen, auf deren Darstellung wir hier aus Platzgründen verzichten müssen, ist nur der Befehl **IN**, der sich zur Menü-Auswahl eignet, für die Programmierpraxis von Bedeutung. Hier wird im Gegensatz zum Bereichstyp genau geprüft, ob der Wert der Variablen in der Menge enthalten ist. Beispiel:

```
PROGRAM DEMO7;
TYPE
 SET1 = SET OF 1..5; {5}
 SET2 = SET OF INTEGER; {nur 0..255}
 SET3 = SET OF 'A'..'Z'; {26}
 SET4 = SET OF CHAR; {0..255}
VAR
 M1: SET1; M2: SET2; M3: SET3; M4: SET4;
BEGIN
M2 := [1, 10, 100..200];
IF 199 IN M2 THEN WRITELN ('ja');
M4 := ['A'..'Z', 'a'..'z'];
IF NOT ('#' IN M4) THEN WRITELN ('nein')
END.
```

1.3. Komplexe Typen

Datensatz = Record-Typ
TYPE R = RECORD ... END
Feld = Array-Typ
TYPE A = ARRAY ... OF ...
Zeichenkette = String-Typ
TYPE S = ARRAY [1..255] OF CHAR
Datei = File-Typ
TYPE F = FILE OF ...
(Zeichenketten und Dateien werden in gesonderten Abschnitten behandelt.)

1.3.1. Datensatz (Record)

Ein Datensatz in der Adreßverwaltung umfaßt die Elemente oder Komponenten Name, Straße, Wohnort usw. In Pascal können *verschiedene* Datentypen, und zwar sowohl einfache Typen (z.B. Ganzzahl, Dezimalzahl) als auch komplexe Typen (z.B. String) als einzelne Elemente zu einem Datensatz (= Record = Verbund) zusammengefaßt werden. Man kann sogar noch weitergehen und Records konstruieren, die selbst aus Records bestehen.
Die Definition eines Records erfolgt im TYPE- oder im VAR-Teil mit Hilfe des Typ-Bezeichners **RECORD**, z.B.:
VAR REC: RECORD... END;
Beispiel:

```
PROGRAM DEMO8;
TYPE
 REC = RECORD
       I: INTEGER;
       R: REAL {kein ';' vor END!}
       END;
VAR
 R1, R2: REC;
BEGIN
 R1.I := 123;
 R1.R := 123.456;
WRITELN (R1.I:5);
WRITELN (R1.R:10:3);
 R2 := R1;
WRITELN (R2.I:5);
WRITELN (R2.R:10:3)
END.
```

Auf den selbstgewählten *Recordtypnamen*, hier „REC", folgt im TYPE-Teil ein „=" (oder im VAR-Teil ein „:") und dann der Typ-Bezeichner RECORD. Danach werden die Elemente des Records so definiert, wie Sie es bereits von den einfachen Variablen her kennen. Jedes Element erhält dabei einen *Elementnamen* (oder Komponentennamen). Nach dem letzten Element wird die Record-Definition mit „END;" abgeschlossen, vor dem in Kyan-Pascal *kein Semikolon* stehen darf.
Auf ein *einzelnes* Element eines Records greifen Sie im Programm zu, indem Sie den *Koppelnamen*
Recordname.Elementname
bilden, z.B. R1.I, R1.R usw. Zwischen Recordname und Elementname steht ein Punkt. Für ein mit einem Koppelnamen bezeichnetes Element gelten dann alle Befehle des Datentyps dieses Elements, z.B.
WRITELN (SIN (R1.R));
READLN (R1.R); usw.
Darüber hinaus können Sie einen Record1 *insgesamt* einem anderen Record2 mit
R2 := R1;
zuweisen, falls beide Records die gleiche Struktur haben.
Der Vergleich zweier Records mit
IF R1 = R2 THEN...
ist in Kyan-Pascal nicht zulässig. Dies gilt auch für Turbo-, nicht aber für UCSD-Pascal.
Man beachte, daß im Falle mehrerer Records die *Record*namen verschieden sein *müssen*, die *Element*namen jedoch gleich sein *dürfen*.
Beispiel:
VAR
A: RECORD A, B: INTEGER END;
B: RECORD A, B: INTEGER END;
Die möglichen Koppelnamen sind nämlich hier alle verschieden: A.A, A.B, B.A, B.B.
Die einzelnen Elemente eines Records befinden sich im Speicher von unten nach oben in der Reihenfolge der Definition der Elemente. Beispiel:

```
VAR R = RECORD
        I1: INTEGER;
        I2: INTEGER;
        I3: INTEGER
        END;

R.I3 = 2 Bytes   oben
R.I2 = 2 Bytes
R.I1 = 2 Bytes   unten
```

(Hinweis: Die WITH-Anweisung und der variante Record mit CASE-OF wurden aus Platzgründen ausgespart.)

1.3.2. Feld (Array)

Etymologischer Hinweis: „Array" bedeutete ursprünglich militärische Schlachtordnung, dann mathematische Reihe, dann Spalte oder Zeile in einer Zahlentabelle. Die germanische Herkunft des englischen „Array" = Phalanx ist dubios. „Array" ist jedenfalls nicht mit „arrangieren" verwandt. Das deutsche „Feld" ist eine Behelfsübersetzung. In der Statistik würde man den Array als Tabelle und die Elemente eines Arrays als Fächer bezeichnen.

Während ein Record in der Regel aus einer Anzahl *verschiedener* Elemente besteht, enthält ein Array eine Vielzahl *gleicher* Elemente. Während die Elemente eines Records durch Koppelnamen angesprochen werden, wird auf die Elemente eines Feldes indiziert zugegriffen. Ein Array wird im TYPE- oder VAR-Teil mit **ARRAY ... OF ...** definiert. Zwischen „ARRAY" und „OF" steht der Bereichstyp (Indextyp) in *eckigen Klammern* [Anfang..Ende], und auf „OF" folgt der Element-Typ. Beispiele:
VAR
I: ARRAY [1..100] OF INTEGER;
R: ARRAY [0..1000] OF REAL;
C: ARRAY [1..255] OF CHAR;
Im eigentlichen Programm kann dann auf die einzelnen Elemente eines Arrays so zugegriffen werden (Beispiele):
I[1] := 32767;
R[500] := 123.456;
C[100] := 'A';
Der Index darf im Programm auch durch eine Variable ausgedrückt werden, z.B.:
J := 100; WRITELN (R[J]);

Zum besseren Verständnis des Feld-Begriffs betrachten wir die Speicherbelegung des Mini-Arrays
A: ARRAY [0..2] OF INTEGER;

```
A[2], 2 Bytes, 2 * 2 = 4 = $8004 oben
A[1], 2 Bytes, 1 * 2 = 2 = $8002
A[0], 2 Bytes, 0 * 2 = 0 = $8000 unten
```

Das Element mit dem kleinsten Index, nämlich A[0], befindet sich unten an der sog. Basis des Arrays. Durch Addition von Basisadresse (hier fiktiv $8000) und Produkt aus Index mal Byte-Anzahl pro Element können wir auf die Speicherstelle des gewünschten Elements des Arrays zugreifen. Diese Berechnung nimmt uns natürlich der Pascal-Compiler ab.
Am einfachsten liegen die Dinge für den Compiler, wenn der Index-Bereich eines Arrays mit 0 beginnt, weil dann intern keine Umrechnung erforderlich ist. Pascal erlaubt jedoch auch andere Index-Bereiche. Im Falle einer Tagestemperatur-Statistik (Streuung -30 bis +40 Grad) würden wir z.B. den Array
VAR T: [-30..+40] OF INTEGER;
wählen. Dann wäre jedoch bei jedem T-Array-Zugriff intern eine Umrechnung erforderlich, die in Kyan-Pascal immerhin über 10 Bytes an zusätzlichem Objektcode erforderlich machen würde. Der Index-Bereich 0..N ist also speichereffizienter.

Arrays sind bei fast allen Datentypen möglich:
ARRAY OF INTEGER,
ARRAY OF REAL,
ARRAY OF BOOLEAN,
ARRAY OF CHAR,
ARRAY OF RECORD usw.
Der oben demonstrierte Integer-Array ist ein sog. *eindimensionaler* Array, den man gelegentlich auch als *Vektor* bezeichnet. Es lassen sich jedoch auch mehrdimensionale Arrays definieren. Bei zweidimensionalen Arrays, die auch *Matrizen* heißen, muß man sich die Elemente in Spalten und Zeilen angeordnet vorstellen. Beispiel:

```
PROGRAM DEMO9;
CONST
 S = 9;
 Z = 9;
TYPE
 MAT = ARRAY [0..S, 0..Z] OF INTEGER;
VAR
 M1, M2: MAT;
 I, J: INTEGER;
BEGIN
FOR I := 0 TO S DO
FOR J := 0 TO Z DO
M1[I,J] := I * J;
M2 := M1;
FOR I := 0 TO S DO
BEGIN
 FOR J := 0 TO Z DO
 WRITE (M2[I,J]:5);
 WRITELN
END
END.
```

Wie Sie dem Programm entnehmen können, läßt sich ein Array1 einem anderen Array2 *insgesamt* zuweisen, wenn beide Arrays die gleiche Struktur aufweisen. Der **Gesamt**vergleich zweier Arrays (IF M1 = M2 THEN . . .) ist in (Kyan-)Pascal nicht möglich.

Wegen der grundsätzlichen Bedeutung des ARRAY OF RECORD für die Dateiverwaltung bringen wir noch ein stark vereinfachtes Demo für eine Mitgliederverwaltung. Mit dem Ausdruck
M[1].ZUNAME
wird die Komponente ZUNAME des ersten Elements des Record-Arrays angesprochen. Der Koppelname-Punkt steht zwischen „M[1]" und „ZUNAME". Der Ausdruck
M.ZUNAME[1]
wäre hier falsch, denn [1] würde sich dann auf ZUNAME und nicht auf M beziehen.

```
PROGRAM MITGLIEDER;
CONST
 N = 3;
TYPE
 MITGLIED = RECORD
 ZUNAME:   ARRAY [1..20] OF CHAR;
 EINTRITT: INTEGER;
 BEITRAG:  REAL
 END;
VAR
 M: ARRAY [1..N] OF MITGLIED;
 I: INTEGER;
BEGIN
FOR I := 1 TO N DO
BEGIN
 WRITELN ('Nummer ', I);
 WRITE   ('Zuname: ');
 READLN  (M[I].ZUNAME);
 WRITE   ('Eintritt: ');
 READLN  (M[I].EINTRITT);
 WRITE   ('Beitrag: ');
 READLN  (M[I].BEITRAG)
END;
WRITELN;
FOR I := 1 TO N DO
BEGIN
 WRITE   (M[I].ZUNAME:20);
 WRITE   (M[I].EINTRITT:10);
 WRITELN (M[I].BEITRAG:12:2)
END
END.
```

Wenn Sie mit Arrays arbeiten, müssen Sie sich Gedanken über die Speicherkapazität machen. In Kyan-Pascal stehen Ihnen 36K (= 36.864 Bytes; $2000-$AFFF) für Programm *und* Daten zur Verfügung. So belegt beispielsweise der Array
VAR R: ARRAY [1..2000] OF REAL;
2000 * 8 = 16.000 Bytes,
und der Array
VAR I: ARRAY [1..10000] OF INTEGER;
10000 * 2 = 20.000 Bytes.

Hinweis: Auf der Peeker-Sammeldisk sind DEMO1 bis DEMO9 sowie MITGLIEDER als Prozeduren zu einem Programm namens **DATENDEMOS** zusammengefaßt.

INTUS-Lern- und Anwenderprogramme

für Apple II - Computer

- Rechtschreibtrainer für Deutsch und Englisch
- Business-English und Alltags-Englisch
- Maschinenschreiben wie der Blitz
- Basic-Lernprogramm, sehr umfangreich
- Kinderschule, Lernen für Vorschulkinder
- AppleGraph, Erstellen von Kreis- und Balken-Graphiken
- Rechenmodelle für AppleWorks-Rechenblatt
- **NEU: MailWorks für AppleWorks-Serienbriefe, für alle Drucker geeignet.**
- PriBu-Privatbuchhaltung gibt den Überblick
- DMP-Charger zur Gestaltung eigener Zeichensätze auf dem Matrix-Drucker.
- und über 200 weitere Programme. Katalog gratis
- Demo-Disketten mit 5-9 Teilprogrammen DM 10,–
- **6000 Frei-Programme (fast) gratis Programm-Liste (Vorkasse) DM 10,–**

INTUS SOFTWARE

Kaiserstr. 21, 7890 Waldshut, Tel. 07751-7920

Ausgabe und **Eingabe** mit

TYPETERM®

im Slot Ihres

APPLE II/IIe

Das bedeutet: Computertextverarbeitung von der Schreibmaschinentastatur! Steckerfertig ohne Umbau.

TYPETERM-Interface **DM 479,–**
für alle BROTHER-Typenradschreibmaschinen ab CE-51

Paketpreis: **DM 1663,–**
Schreibmaschine CE-61 mit TYPETERM

CE-68 mit TYPETERM ..... DM 2143,–
CE-70 mit TYPETERM ..... DM 2758,–
EM-80 mit TYPETERM ..... DM 1887,–
TYPETERM-Kit für CE-50 ..... DM 468,–
Cable Kit A
(erforderlich ab EM-80) ..... DM 103,–

TYPETERM – ein starkes Interface für starke Maschinen! Alle Cursor- und Ctl-Befehle. 4k ROM auf der Karte für DOS, PRODOS, CP/M, PASCAL. 2 Zeichensätze verfügbar z. B. deutsch u. ASCII. Alle Features: Hoch-/Tiefstellen, autom. Unterstreichen, var. Zeichen und Zeilenabst., autom. Papierzuführung usw. Ausführl. Handbuch vorab: 10,– DM auf Konto 14770-306 PGiroA Han (Anrechnung).

TYPETERM – ein Produkt von

interkom electronic Kock & Mreches GmbH Postf., 3004 Isernhagen 4 Telefon 05139-87393

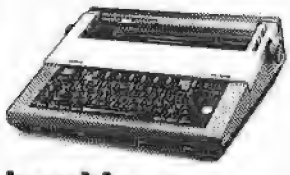
Ausgabe mit

TYPETERM® JUNIOR

im Slot Ihres

APPLE II/IIe

Paketpreis **DM 899,–**
Schreibmaschine AX-10 mit Interface TYPETERM JUNIOR, steckfertig.



brother
QUALITÄT AUS ERSTER HAND.

TYPETERM JUNIOR mit AX-10 – unser besonders günstiges Gespann, ebenfalls steckfertig. Mit TYPETERM JUNIOR kann die AX-10 mehr. Sie wird zum vollwertigen Typenraddrucker für Ihren Apple:

- 3 verschiedene Schriftstärken
- Automatisches Unterstreichen
- 2 Zeichensätze z.B. deutsch u. ASCII
- 2 Zeichenabstände
- 2k ROM auf der Karte für Ausgabe unter DOS, PRODOS, CP/M u. PASCAL.

TYPETERM JUNIOR – ein Produkt von

interkom electronic Kock & Mreches GmbH Postf., 3004 Isernhagen 4 Telefon 05139-87393

ZUSATZ-KARTEN:

V-24-Schnittstelle	199,-	Z-80-Karte	98,-
80-Zeichen-Karte m.Softswitch	236,-	16 K-Language-Karte	98,-
Joy Stick De Luxe	59,-	Accelerator 3,6 MHz	950,-
68000 Intemex	1600,-	PAL Karte	110,-
RGB Karte	239,-	IEEE 488	312,-
Koppler dataphon m. FTZ	325,-	Z 80 B Karte mit Software	919,-
Centronics-Karte von Epson	für Graphik 210,-	für Text	145,-
Centronics-Schnittstelle für 2 Drucker gleichzeitig			129,-

Super-Eprommer 239,-
belegt keinen Slot, incl. Software für 2716-27128

Floppy-Controller

FDC 4 für alle Laufwerke ..... 169,- Bausatz wie links 159,-
Leerplatine wie oben incl. Prom u. Eprom 98,-

Erphi-Controller 298,-

Disketten 1D, 48 tpi ..... 10 St. 29,-
Disketten 2D, 48 tpi ..... 10 St. 36,-

Preh Commander Keyboards

Wir bieten Ihnen die **Preh-Qualität** auch für Apple. AK 88 spez. mit Gehäuse, Anschlußkabel, Zehner-Tastenfeld, dt. Zeichensatz, Sondertasten für Ctrl-Codes und Rechenfunktionen 339,-
Preh Commander Keyboard, frei programmierbar bis zu 10 Ebenen, pro Taste bis zu 250 Zeichen ..... nur 599,-
Gleiche Tastatur wie oben **für Apple IIe** ..... nur 698,-

TEAC 3½" Laufwerk FD 35 F 498,-
Speicherkapazität 1 MB, (formatiert 640 KB) jetzt für nur .....

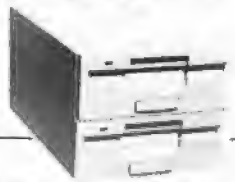
TEAC FD 55 AV 1 x 40 Track .... 395,- TEAC FD 55 BV 2 x 40 Track 460,-
TEAC FD 55 EV 1 x 80 Track .... 445,- TEAC FD 55 FV 2 x 80 Track 398,-

Apple®-kompatibles Laufwerk incl. Gehäuse + Kabel 599,-
320 KB Laufwerk für IIc 948,-
640 KB Laufwerk für IIc 1088,-

Panasonic Drucker: 1090 ..... nur 849,-
1091 ..... nur 1095,- 1092 ..... nur 1295,-

Die Microfloppy mit Zukunft:

Speicherkapazität: 2 x 1 MByte formatiert: 2 x 640 kByte. Anschlußfertig mit PROM-residenter Patchsoftware für CP/M 2.2, Apple DOS 3.3, DiversiDOS 2-C, 4-C (DD MOVER), Apple Pascal 1.1, Pascal 1.2, Pro-DOS 1.0.1, 1.1, 1.1.1 zum Preis von 1498,-
Low Power Version 1598,-



10 MB Winchester
mit Software für DOS 3.3, CP/M 2.20, Pascal, Pro-DOS, incl. Controler und Gehäuse 3980,-

Sonderangebot

Chinon Laufwerk für II + IIe, incl. Kabel u. Gehäuse ..... jetzt nur 339,-

Gesamt-Preisliste anfordern! **Preise inclusive gesetzlicher Mehrwertsteuer.**
Händlerpreisliste bitte schriftlich anfordern!

UEDING electronics

Holtewiese 2
5750 Menden 1

DFÜ 02373/66877
Tel. 02373/63159

2. Unterprogramme

Normalerweise werden die Befehle in Pascal von oben nach unten, d.h. in der Reihenfolge des Quelltextes, abgearbeitet. Dies gilt nicht für
1. Verzweigungen und Schleifen (IF-THEN-ELSE, CASE-OF-END, FOR-TO-DO, WHILE-DO, REPEAT-UNTIL),
2. Sprünge (LABEL, GOTO) und
3. Unterprogramme (PROCEDURE, FUNCTION).

2.1. LABEL und GOTO

GOTO-Sprünge sind in Pascal aus pädagogischen Gründen bewußt erschwert worden. Zunächst müssen die Labels (= Marken, Sprungadressen als Nummern) im LABEL-Teil definiert werden. Danach kann *innerhalb* eines sog. *Blocks* (Hauptprogramm, Prozedur oder Funktion) zu einem Befehl gesprungen werden, der mit einer Nummer und „:" markiert worden ist. Beispiel:

```
PROGRAM GOTODEMO;
LABEL
 1, 2, 3, 4;
CONST
 N = 1000;
VAR
 I: INTEGER;
BEGIN
I := 0;
1: WRITE ('1,'); I := I + 1;
   IF I = N THEN GOTO 4 ELSE GOTO 3;
2: WRITE ('2,'); I := I + 1;
   IF I = N THEN GOTO 4 ELSE GOTO 1;
3: WRITE ('3,'); I := I + 1;
   IF I = N THEN GOTO 4 ELSE GOTO 2;
4: WRITELN ('4.'); WRITELN ('Ende')
END.
```

Nach der Ausgabe von „1" erfolgt ein Sprung zur Marke 3, von dort nach Ausgabe von „3" ein Sprung zur Marke 2, von dort nach Ausgabe von „2" ein Sprung zur Marke 1 usw. Das Programm endet, wenn I = N ist. Mit GOTOs ließen sich theoretisch auch Unterprogramme konstruieren, doch gibt es dafür in Pascal die Prozeduren und Funktionen.

2.2. Prozeduren

In den meisten BASIC-Dialekten werden Unterprogramme mit GOSUB n aufgerufen und mit RETURN verlassen. In Pascal haben Unterprogramme einen benutzerdefinierten Namen (z.B. SUB), und der Aufruf eines Unterprogramms erfolgt im Hauptprogramm im einfachsten Fall durch Angabe des Namens, z.B.
SUB;
Prozeduren sind Unterprogramme, die mit dem reservierten Wort **PROCEDURE** eingeleitet werden, z.B.
PROCEDURE SUB;
Eine Prozedur ist im übrigen wie das Hauptprogramm selbst aufgebaut und kann die Teile LABEL, CONST, TYPE und VAR aufweisen. Die eigentliche Prozedur beginnt mit „BEGIN" und endet mit „END;", wohingegen das Hauptprogramm mit „END." endet. Die weiteren Feinheiten wollen wir Schritt für Schritt anhand des Programms **SUBDEMOS** untersuchen.

SUBDEMOS

```
PROGRAM SUBDEMOS;      {Kopf}
VAR                     {von}
 G: INTEGER;        {Level 0}
 F: INTEGER;

{------------------------------}

PROCEDURE LOCAL;
VAR
 I: INTEGER;
BEGIN
 FOR I := 1 TO 10 DO WRITE ('-');
 WRITELN
END;

PROCEDURE GLOBAL;
BEGIN
 G := G + 10; {G geaendert}
 WRITELN (G)
END;

PROCEDURE WERTPAR (L: INTEGER);
BEGIN
 L := L + 10; {G unveraendert}
 WRITELN (L)
END;

PROCEDURE VARPAR (VAR L: INTEGER);
BEGIN
 L := L + 10; {G geaendert}
 WRITELN (L)
END;

{------------------------------}

FUNCTION FOHNE: INTEGER;
BEGIN
 FOHNE := 2;
 WRITELN (2) {Nicht mit FOHNE!}
END;

FUNCTION FMIT (L: INTEGER):INTEGER;
BEGIN
 FMIT := L * L;
 WRITELN (L * L) {Nicht mit FMIT!}
END;

{------------------------------}
PROCEDURE LEVEL1; {Kopf}
VAR               {von}
 L1: INTEGER;     {Level 1}
{------------------------------}
PROCEDURE LEVEL2; {Kopf}
VAR               {von}
 L2: INTEGER;     {Level 2}
{------------------------------}

BEGIN    {Main von Level 2}
 LOCAL;
 WRITELN ('Level ', G);
 WRITELN ('Level ', L1);
 L2 := 2;
 WRITELN ('Level ', L2)
END;
{------------------------------}

BEGIN   {Main von Level 1}
 LOCAL;
 WRITELN ('Level ', G);
 L1 := 1;
 WRITELN ('Level ', L1);
 LEVEL2
END;
{------------------------------}

BEGIN   {Main von Level 0}

LOCAL;
G := 1; GLOBAL;          {11}
WRITELN (G);             {11}

LOCAL;
G := 1; WERTPAR (G);     {11}
WRITELN (G);             {1}
G := 1; WERTPAR (2);     {12}
WRITELN (G);             {1}

LOCAL;
G := 3; VARPAR (G);      {13}
WRITELN (G);             {13}

LOCAL;
F := FOHNE;              {2}
WRITELN (F);             {2}

LOCAL; G := 2;
F := FMIT (G);           {4}
WRITELN (G);             {2}

LOCAL; G := 0;
WRITELN ('Level ', G);
LEVEL1;
END.
```

2.2.1. Lokale Variablen

Die Prozedur LOCAL hat keine Parameterliste (s.u.) und verwendet keine der nach dem Kopf des Hauptprogramms SUBDEMOS definierten Variablen G und F. Wir sprechen deshalb von einer sog. *geschlossenen* Prozedur, weil sie weder Werte des Hauptprogramms übernimmt noch Werte an das Hauptprogramms übergibt. Trotzdem wird in der Prozedur LOCAL eine Variable namens I definiert und benutzt. *Alle* im CONST-, TYPE- und VAR-Teil einer Prozedur definierten Variablen sind jedoch sog. **lokale** Variablen, die nur *innerhalb* der Prozedur und damit nicht außerhalb im Hauptprogramm bekannt sind. Der *Name* einer Variablen ist dabei irrelevant, vorausgesetzt die Variable wird im VAR-Teil definiert. Wenn wir in der Prozedur LOCAL den Namen I durch den Namen G ersetzen, so führt dies *nicht* zu einer Fehlermeldung während des Compilierens, obwohl der Name G bereits im Hauptprogramm vorkommt. Grund: Die Variable G des Hauptprogramms hat mit der Variablen G der Prozedur nur den äußeren Namen gemein; in Wirklichkeit sind es zwei verschiedene Variablen.

Allgemein formuliert: Wenn im VAR-Teil eines Unterprogramms eine Variable definiert wird, so ist sie dem übergeordneten Hauptprogramm nicht bekannt. Haben zufällig die Variable X des Hauptprogramms und die Variable X des Unterprogramms dieselbe äußere Form, dann gilt innerhalb des Unterprogramms nur die im VAR-Teil definierte Variable X, und das Unterprogramm weiß nichts vom Wert der im Hauptprogramm definierten Variablen X. Diesen Sachverhalt bezeichnen wir als *Namenspriorität*, die auf Prozeduren und Funktionen (s.u.) gleichermaßen zutrifft.

2.2.2. Globale Variablen

Betrachten wir nunmehr die Prozedur GLOBAL. Hier fehlt der VAR-Teil, und folglich muß die in GLOBAL benutzte Variable G eine sog. **globale** Variable sein, die bereits im übergeordneten Hauptprogramm definiert worden ist. Wenn eine globale Variable nicht zufällig wegen Namensgleichheit im Prozedur-VAR-Teil außer Kraft gesetzt wurde, so ist sie der Prozedur bekannt.

Merksatz:
global definiert = lokal bekannt.
lokal definiert = global unbekannt.
In der Prozedur GLOBAL wird die globale Variable G nicht nur benutzt, sondern auch verändert. Dies bezeichnet man als *Seiteneffekt*, der in der Regel unerwünscht ist. Denn dadurch, daß im VAR-Teil einer Prozedur lokal definierte Variablen mit im Hauptprogramm global definierten Variablen namensgleich sein können, besteht erst die Möglichkeit, daß mehrere Programmierer *unabhängig voneinander* verschiedene Prozeduren für *dasselbe* Hauptprogramm entwickeln. Wenn jedoch Seiteneffekte, d.h. die Veränderung von Hauptprogrammvariablen in Unterprogrammen, vermieden werden sollen, so muß es eine andere Möglichkeit der Parameterübergabe geben, weil sonst nur geschlossene Unterprogramme in der Art der obigen Prozedur LOCAL geschrieben könnten.

2.2.3. Wertparameter

Die Prozedur WERTPAR enthält im sog. Prozedurkopf in *runden Klammern* eine besondere Variable, die wir als *Parameter* oder im Falle mehrerer Variablen als Parameterliste bezeichnen. Hier wird lediglich 1 Parameter mit Typzusatz definiert. Ferner entnehmen wir dem Hauptprogramm, daß der Aufruf der Prozedur mit einem Parameter erfolgt, und zwar ebenfalls in runden Klammern, aber *ohne* Typzusatz, weil der Typ bereits im Kopf des Hauptprogramms definiert wurde. Es liegt damit eine Typübereinstimmung von Prozedurkopf-Parameter und Prozeduraufruf-Parameter vor:
PROCEDURE WERTPAR (L:INTEGER);
WERTPAR (G);
Was passiert nun beim Aufruf der Prozedur? Der *Wert* der globalen Variablen G wird in die Speicherstelle des Prozedur-Parameters L kopiert. Da G ursprünglich den Wert 1 hatte, hat nun auch der Parameter L den Wert 1. Wenn nun L *innerhalb* der Prozedur WERTPAR geändert wird, so hat dies keinen Einfluß auf den Wert der globalen Variablen G. Wir bezeichnen deshalb den im Prozedurkopf definierten Parameter L als sog. **Wertparameter**: Das Hauptprogramm übergibt an die Prozedur einen Wert (= Parameter-Input), aber die Prozedur gibt keinen Wert an das Hauptprogramm zurück (= kein Parameter-Output).
Auch für den Wertparameter L gilt wie für die Definition von Variablen im VAR-Teil von Prozeduren, daß der äußere Name irrelevant ist. Statt L hätten wir auch G wählen können:
PROCEDURE WERTPAR (G: INTEGER);
WERTPAR (G);
Dann wäre ebenfalls der Wert der globalen Variablen G in die Speicherstelle des Prozedur-Parameters G dupliziert worden, und die globale Variable G wäre nicht durch die Prozedur WERTPAR verändert worden. Wie der Ausdruck Wertparameter impliziert, kann der Wert auch als Konstante übergeben werden:
PROCEDURE WERTPAR (G: INTEGER);
WERTPAR (1);
Hier wird der Wert 1 direkt in den Prozedur-Parameter G kopiert, so daß der Umweg über die globale Variable G entfällt. Wir halten fest:
Bei einer Wertparameter-Prozedur ist der Parameter bei der aufgerufenen Prozedur stets eine Variable, während der Parameter beim Prozeduraufruf eine globale Variable oder auch eine Konstante bzw. ein Formelausdruck sein kann.
(Prozedur-Parameter sind eine besondere Art von lokalen Variablen, die sich im Speicher direkt an die im VAR-Teil der Prozedur definierten normalen lokalen Variablen anschließen.)

2.2.4. Variablenparameter

Nun mag es wünschenswert sein, daß einer Prozedur nicht nur Werte vom Hauptprogramm zugeführt werden (Parameter-Input), sondern daß auch Werte an das Hauptprogramm zurückgegeben werden (Parameter-Output). Betrachten wir hierzu die Prozedur VARPAR mit dem Prozedurkopf

PROCEDURE VARPAR (VAR L: INTEGER);
die im Hauptprogramm durch
VARPAR (G);

aufgerufen wird. Sie unterscheidet sich von der Prozedur WERTPAR lediglich dadurch, daß dem Prozedur-Parameter L das Wort „VAR" vorangestellt wurde, das hier eine besondere Bedeutung hat. VAR vor dem Parameter L besagt, daß sich der Wert der globalen Variablen G und der Wert des Variablenparameters L *denselben* Speicherplatz teilen. Während bei einem Wertparameter der *Wert* der globalen Variablen in die Speicherstelle des Wertparameters kopiert oder dupliziert wird, wird bei einem **Variablenparameter** intern nur ein *Zeiger* übergeben, der auf die Speicherstelle der globalen Variablen zeigt. Wenn nun der Variablenparameter innerhalb einer Prozedur geändert wird, so wird damit gleichzeitig die globale Variable geändert, denn der Wert des Variablenparameters und der Wert der globalen Variablen sind physisch identisch. Obwohl damit praktisch nur eine einzige Variable existiert, ist auch hier der äußere Name des Variablenparameters irrelevant, denn statt L mußten wir nicht G nehmen, um G zu ändern.

Der Begriff Variablenparameter impliziert bereits, daß der beim Prozeduraufruf genannte Parameter stets eine Variable sein muß, während bei Wertparametern auch Konstanten zulässig sind.

Es gibt damit im einzelnen folgende Prozedurtypen:
1. Prozeduren ohne lokale Variablen,
2. Prozeduren mit lokalen Variablen, die im VAR-Teil der Prozedur definiert werden,
3. Prozeduren ohne Parameter,
4. Prozeduren mit Wert- und/oder Variablenparametern, die im Prozedurkopf definiert werden.
Beispiele für Prozedurköpfe und Prozeduraufrufe:

```
PROCEDURE P1;
P1;
PROCEDURE P2 (I: INTEGER; C: CHAR);
P2 (10, 'A');
PROCEDURE P3 (R: REAL; B: BOOLEAN);
P3 (R1, B1);
PROCEDURE P4 (VAR I: INTEGER; R: REAL);
P4 (I, 123.456);
PROCEDURE P5 (VAR C1, C2: CHAR; R:REAL);
P5 (CC1, CC2, R);
PROCEDURE P6 (R: REAL; C: CHAR);
P6 (R1 * 3.1415, CHR(65));
```

2.3. Funktionen

Wir haben bereits die meisten *vordefinierten Funktionen* kennengelernt, z.B. SIN, LN, CHR, ORD usw. Betrachten wir noch einmal folgende Beispiele:
Y := SIN (X);
Y := SIN (30);
WRITELN (SIN (X));
WRITELN (SIN (30));
Y ist der *Funktionswert* (abhängige Variable, Ergebnis, Ausgabewert), der stets eine Variable sein muß, sofern nicht wie im Falle von WRITELN das Ergebnis direkt ausgegeben wird.
X ist das *Argument* (unabhängige Variable, Stelle, Eingabewert), das sowohl eine Variable als auch eine Konstante sein kann.
Die *benutzerdefinierten Funktionen* unterscheiden sich einerseits von den vordefinierten Funktionen und andererseits von den Prozeduren:
1. Benutzerdefinierte Funktionen geben immer einen Funktionswert zurück. Dieser muß *innerhalb* der Funktionsroutine dem Funktionsnamen zugewiesen werden, der gewissermaßen als abhängige Variable fungiert:
SIN := 0.5;
Bei vordefinierten Funktionen wird diese Tatsache verdeckt, denn im (Haupt)programm kann man nicht „SIN := 0.5" schreiben.
2. Der Prozedurkopf beginnt mit PROCEDURE + Prozedurname und der Funktionskopf mit **FUNCTION** + Funktionsname.
3. Wie nach dem Prozedurnamen werden nach dem Funktionsnamen die Parameter in *runde Klammern* gesetzt. Wert- und Variablenparameter sind gleichermaßen zulässig.
4. Im Gegensatz zur Prozedur muß der Datentyp des Funktionsnamens, der den Funktionswert aufnimmt, definiert werden. Dies geschieht durch den Zusatz von „:" + Datentyp *nach* der eingeklammerten Parameterliste (falls eine solche existiert).

2.3.1. Funktionen ohne Argument

Normalerweise hat jede Funktion ein Argument. Wie wir der Funktion FOHNE entnehmen können, sind jedoch auch Funktionen ohne Argument denkbar. Die Funktion
FUNCTION FOHNE: INTEGER;
wird durch
F := FOHNE;
im Hauptprogramm aufgerufen. Innerhalb der Funktion erfolgt mit
FOHNE := 2;
die Zuweisung von 2 an den Funktionsnamen, und dieser Wert 2 wird dann quasi nach dem „END;" der Funktion in die Variable F kopiert.
Funktionen ohne Argument sind normalerweise sinnlos. Man benötigt sie jedoch, um gewisse I/O-Adressen (Drehregler, Tastatur und sonstige Softswitches) abzufragen. Die Eingabewerte kommen dann nicht vom Pascal-Hauptprogramm, sondern von den I/O-Adressen, und die Funktion selbst wird üblicherweise in Assembler geschrieben. Ein Beispiel für eine sinnvolle Funktion „ohne" Argument ist die RDKEY-Routine aus dem Grundkurs.

2.3.2. Funktionen mit Argument

Die Funktion FMIT ist ein Beispiel für eine Funktion mit *einem* Argument, hier L, das als Parameter in

FUNCTION FMIT (L: INTEGER): INTEGER;
enthalten ist. Der Funktionsaufruf mit
F := FMIT (G);
könnte auch durch
WRITELN (FMIT (G));
ersetzt werden.
Es gibt auch Funktionen mit *mehreren* Argumenten. Ferner können auch Variablenparameter definiert werden, die man jedoch aus Gründen der Übersichtlichkeit meistens vermeidet. Beispiele für Funktionsköpfe und Funktionsaufrufe:

```
FUNCTION F1 (I1, I2: INTEGER): INTEGER;
FI := F1 (10, 20);
FUNCTION F2 (R: REAL; VAR R: REAL): REAL;
FR := F2 (10.5, RR);
FUNCTION F3 (I: INTEGER): BOOLEAN;
FB := F3 (10 * IV);
FUNCTION F4 (R: REAL; C: CHAR): CHAR;
FC := F4 (R1, C1);
```

Hinweis: Im Gegensatz zu Standard-, UCSD- und Turbo-Pascal sind in Kyan-Pascal auch String-Funktionen definierbar. Komplizierte Beispiele für Funktionen und Prozeduren können Sie dem Abschnitt über Strings entnehmen.

2.4. Verschachtelung

Bislang haben wir vereinfachend angenommen, daß Prozeduren und Funktionen nur vom Hauptprogramm aufgerufen werden können. In Wirklichkeit kann jedoch ein aufgerufenes Unterprogramm selbst wieder andere Unterprogramme aufrufen:
Level 0: Das Hauptprogramm hat den Level = den Grad = die Hierarchie = die Verschachtelungstiefe 0 (= niedrigstes = unterstes Level).
Level 1: Die Prozeduren und Funktionen, die vom Hauptprogramm aufgerufen werden, haben den Level 1.
Level 2: Die Prozeduren und Funktionen, die von den Prozeduren und Funktionen, die vom Hauptprogramm aufgerufen werden, aufgerufen werden, haben den Level 2.
Das „Spielchen" ließe sich weiter fortsetzen. Wenn wir die Prozeduren LEVEL1 und LEVEL2 in dem Listing SUBDEMOS genauer ansehen, so können wir unseren Merksatz
global definiert = lokal bekannt
lokal definiert = global unbekannt
dahingehend generalisieren, daß wir nunmehr „global" im Sinne von relativ niedrigerem und „lokal" im Sinne von relativ höherem Level interpretieren. Wenn es in einem konkreten Pascal-Programm drei Levels gibt, so sind dem mittleren Level die Variablen des untersten Levels bekannt und die Variablen des obersten Levels unbekannt. Der höchste Level kennt die meisten Variablen, der niedrigste Level die wenigsten. Der höchste Level ist damit am meisten für Seiteneffekte anfällig.

Forward: Wie die Variablen müssen in Pascal auch die Unterprogramme vor deren Benutzung bzw. Aufruf definiert werden. In gewissen Fällen rufen sich jedoch zwei Unterprogramme gegenseitig auf. Da man aber nicht zwei Unterprogramme simultan definieren kann, behilft man sich mit dem FORWARD-Befehl, der zu einem späteren Zeitpunkt erläutert wird.
Rekursion: Ein Unterprogramm kann nicht nur andere Unterprogramme, sondern auch sich selbst aufrufen, was als Rekursion bezeichnet wird. Wenn man z.B. in SUBDEMOS innerhalb der FOHNE-Funktion die Prozedur WRITELN (FOHNE) verwenden würde, so entstünde eine Rekursion.

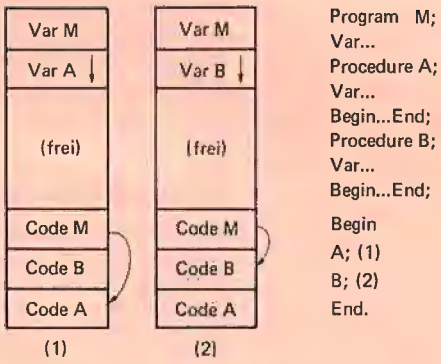


Abb. 1
Getrennte Aufrufe von Level-1-Prozeduren

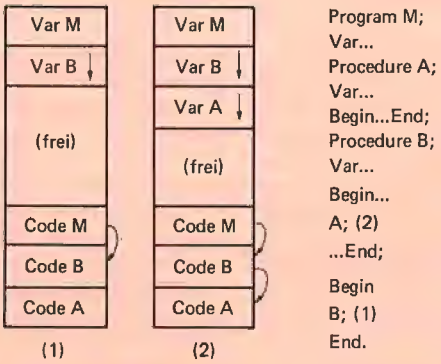


Abb. 2.
Verkettete Aufrufe von Level-1-Prozeduren

Exkurs: Fehlerbehandlung
Wer bislang in BASIC programmiert hat, wird als Pascal-Neuling schnell die leidvolle Erfahrung machen, daß die mächtigeren Pascal-Befehle mit einer geradezu vorsintflutlichen Speicherverwaltung gekoppelt sind. Noch nie konnte ein Applesoft-Programmierer mit einem normalen BASIC-Befehl Programm und DOS zerstören. Pascal ist hier weniger narrensicher: Eine falsch gewählte Array-Dimensionierung (in UCSD- und Kyan-Pascal), eine irrtümlich programmierte Endlos-Rekursion (in Turbo-Pascal) – und schon werden Programm, Daten und Betriebssystem sprichwörtlich ausgelöscht. Die obige FOHNE-Rekursion habe ich unter Turbo-Pascal 3.0 mit der Premium Softcard getestet. Es zeigte sich, daß hierdurch die *gesamten* 64K bis auf ein paar Bytes gnadenlos überschrieben wurden. Deshalb hier ein mahnendes Wort an alle Pascal-Systemprogrammierer: Wer den Top- nicht vom Bottom-Pointer unterscheiden kann, sollte Fibonacci in Frieden ruhen lassen.

3. Strings

In UCSD- und Turbo-Pascal gibt es den vordefinierten Typ-Bezeichner STRING sowie die Befehle LENGTH, CONCAT, POS, COPY, DELETE und INSERT. Hier ist somit „STRING" ein *einfacher* Datentyp.
In Standard- und Kyan-Pascal muß man den String-Typ als ARRAY OF CHAR (= Zeichenkette) definieren und entsprechende String-Befehle selbst implementieren. Hier ist somit „STRING" ein *komplexer* Datentyp.
Für Kyan-Pascal wird die Typ-Definition
TYPE STRING =
ARRAY [1..STRLEN] OF CHAR;
empfohlen. Es sollte der Typname „STRING" verwendet werden. Der Bereichsanfang sollte 1 betragen, und die Integer-Konstante STRLEN (= String-Länge) sollte 255 nicht überschreiten. Die maximale String-Länge ist damit 255, und intern belegt ein String maximal 255 Bytes; ein spezielles Länge-Byte wie in Turbo- und UCSD-Pascal gibt es nicht. Anomale Bereichswerte, z.B.
[0..255] {0 statt 1}
[1..1000] {1000 > 255}
sind zulässig, doch gelten dann die folgenden Ausführungen nur noch mutatis mutandis.

3.1. Vordefinierte Befehle

1: Ein String kann als Konstante definiert werden, wobei die Längenangabe entfällt, z.B.
CONST SK = 'Peeker';
Ein String-Wert ('abc...') sollte nur *eine* Quelltextzeile einnehmen (knapp 80 Einzelzeichen).

2: Strings können indirekt im TYPE-Teil oder direkt im VAR-Teil definiert werden, z.B.
VAR S: ARRAY [1..80] OF CHAR;

3: Ein String kann mit READLN eingegeben und mit WRITELN ausgegeben werden. Die maximale Länge des ein- oder auszugebenden Strings darf 255 nicht überschreiten. Programmfragment:

```
VAR S: ARRAY [1..255] OF CHAR;
BEGIN
READLN (S);
WRITELN (S);
```

Gibt man bei READLN z.B. „12345" – gefolgt von Return – ein, so werden die restlichen 250 Zeichen des Strings S mit Leertasten (Spaces) aufgefüllt. Bei WRITELN werden diese Leertasten mit ausgegeben. Dies läßt sich durch die LENGTH-Funktion vermeiden (s.u.).

4: Die Zuweisung eines String-Wertes zu einer String-Variablen ist nur unter Beachtung der definierten String-Länge zulässig. Programmfragment:

```
VAR S: ARRAY [1..5] OF CHAR;
BEGIN
S := '12345'; {richtig}
S := '123456'; {zuviele Zeichen}
S := '1234'; {zuwenige Zeichen}
S := '123□□'; {Spaces erlaubt}
```

5: Zwei Strings gleicher Länge können alphabetisch gemäß der ASCII-Tabelle verglichen werden (<, =, >, <=, >=, <>). Programmfragment:

```
TYPE
STRING = ARRAY [1..5] OF CHAR;
VAR S1, S2;
BEGIN
S1 := 'abcde';
S2 := 'abcdf';
IF S1 < S2 THEN WRITELN ('stimmt');
```

6: Die Elemente = Einzelzeichen eines Strings (= Einzelzeichen-Arrays) sind von 1 bis STRLEN durchnumeriert. Man sagt auch, daß sich das 1. Zeichen eines Strings in Position 1 (= Index 1), das 2. Zeichen in Position 2 (= Index 2) befindet usw.:

```
'ABCDEFGHIJ'   S: ARRAY OF CHAR[1..10]
 1234567890    Positionen P 1-10
    P          CHAR 'D' in P = 4 von S
```

Damit ist es möglich, das pte Zeichen eines Strings durch ein Einzelzeichen zu ersetzen und umgekehrt. Programmfragment:

```
VAR
S: ARRAY [1..5] OF CHAR;
C: CHAR;
BEGIN
S := 'ABCDE';
C := 'X';
S[3] := C;
WRITELN (S); {'ABXDE'}
C := S[1];
WRITELN (C); {'A'}
```

7: Wenn man den String als einfachen Datentyp ansieht (s.o.), dann ist der String-Array (ARRAY OF STRING) ein eindimensionales Feld. Wenn man umgekehrt den String als komplexen Datentyp betrachtet, dann ist der String-Array (ARRAY OF ARRAY OF CHAR) ein zweidimensionales Feld. Letzteres gilt für Kyan-Pascal. Programmfragment:

```
VAR
SA: ARRAY [1..10] OF
ARRAY [1..5] OF CHAR;
SB: ARRAY [1..10,1..5] OF CHAR;
C: CHAR;
BEGIN
SA[3] := 'ABCDE';
SB[4] := '12345';
C := 'X';
SA[3][1] := C; SB[4,5] := C;
WRITELN (SA[3]); {XBCDE'}
WRITELN (SB[4]); {1234X'}
```

Wie ersichtlich, kann ein String-Array als ARRAY [1..M] OF ARRAY [1..N] OF CHAR oder als ARRAY [1..M, 1..N] OF CHAR definiert werden. Die erste Dimension 1..M bedeutet die Anzahl der Strings, die zweite Dimension 1..N die Anzahl der Zeichen je String. Ein einzelnes Zeichen eines Strings des String-Arrays S kann mit S[M,N] oder mit S[M][N] selektiert werden.

3.2. Benutzerdefinierte Befehle

In Kyan-Pascal sind keine weiteren String-Befehle implementiert. Es ist jedoch möglich, eigene Befehle als Funktionen oder Prozeduren zu definieren. Diese können als sog. **Include-Files** angelegt werden. Ein Include-File (to include = einfügen) ist eine normale Quelltextdatei, die beim Compilieren in einen *anderen* Quelltext (Main-File) eingefügt wird. Beispiel:

```
{/RAM/MAIN}                    MAIN
PROGRAM MAIN;
BEGIN
WRITELN ('Main-File');
#I /RAM/INCLUDE
END.
```

```
{/RAM/INCLUDE}                 INCLUDE
WRITELN ('Include-File);
```

Angenommen, der Main-File sei unter dem Namen „/RAM/MAIN" und der Include-File unter dem Namen „/RAM/INCLUDE" auf der RAM-Disk gespeichert. Wenn man nun den Kyan-Compiler mit „PC" startet und „/RAM/MAIN" als Quelltext angibt, so wird zunächst der Main-File eingelesen. Sobald der Compiler auf eine Quelltextzeile stößt, die mit
#I
beginnt, interpretiert er die nachfolgende Zeichenkette als ProDOS-Dateinamen für den Include-File. *Der Include-File wird in den Main-File so eingespielt, als wäre der Include-File physischer Bestandteil des Main-Files.*
Das Programm **STRINGDEMOS** enthält Funktionen und Prozeduren für die String-Befehle LENGTH, CONCAT, POS, COPY, DELETE und INSERT. Zwischen den Trennstrichen („=====") befindet sich der eigentliche Include-File. Danach folgenden jeweils mehrere Anwendungsbeispiele für die Befehle. Zweck und Anwendung der Befehle ist in dem Listing genau beschrieben, so daß wir uns hier auf einige allgemeine Angaben beschränken können:

1 Ihr eigenes Hauptprogramm muß die Konstante STRLEN sowie den Typ STRING im Programmkopf festlegen. Die Namen „STRLEN" und „STRING" sind obligatorisch, weil sie in den String-Routinen verwendet werden.

2 Die String-Befehle funktionieren nur in bezug auf *einen* String-Typ mit *einer* bestimmten Länge. Dies ist der entscheidende Nachteil gegenüber UCSD- und Turbo-Pascal, deren eingebaute Routinen sich auf Strings beliebiger Länge beziehen. Wir werden deshalb im Peeker String-Routinen veröffentlichen, die diese Beschränkung umgehen und im übrigen aus Gründen der Kompaktheit und Geschwindigkeit in Assembler geschrieben sind.

3 Die String-Routinen sind bewußt fehlertolerant programmiert, weil UCSD- und Turbo-Pascal bei illegalen Parametern (z.B. Position P > Länge L) ohne Fehlermeldung durchdrehen.

4. Dateien

Aus Platzgründen ist die Darstellung der Dateiverwaltung stark gestrafft und setzt zudem ProDOS-Kenntnisse voraus.

Kyan-Pascal benutzt die drei ProDOS-Dateitypen SYS, BIN und TXT. *Programmdateien* sind SYS-Dateien. *Datendateien* können BIN- oder TXT-Dateien sein (s.a. Listing **FILEDEMOS**).

4.1. Textfiles

Eine **TXT**-Datei (= Textfile), die in Kyan-Pascal **FILE OF TEXT** heißt und mit WRITELN/READLN bearbeitet wird, ist eine sequentielle ASCII-Textdatei mit zumeist *unterschiedlich langen* Dateifeldern, die durch Returns (R) begrenzt werden und von 0 bis n durchnumeriert sind, z.B.
Feld 0: Hans + R = 5 Zeichen
Feld 1: im + R = 3 Zeichen
Feld 2: Glück + R = 6 Zeichen
Als Feldinhalte sind Einzelzeichen, Zeichenketten, Ganzzahlen und Dezimalzahlen zugelassen, wobei Zahlen durch WRITELN in ASCII-Ziffernfolgen konvertiert und durch READLN in die binäre Form zurückverwandelt werden.
Vor dem Öffnen einer Datei (ProDOS-OPEN) müssen der externe Dateiname, z.B.
CONST TEXTNAME = '/VOL/TEXTFILE';
und der interne Dateiname, z.B.
VAR TEXTFILE: FILE OF TEXT;
festgelegt werden. Der externe Dateiname unterliegt den üblichen ProDOS-Regeln und kann im Falle einer Variablen auch über die Tastatur eingegeben werden.
Das Speichern einer *neuen Datei* beginnt mit
REWRITE (TEXTFILE, TEXTNAME);
und das Einlesen oder Überschreiben einer *alten Datei* mit
RESET (TEXTFILE, TEXTNAME);
Für die nunmehr geöffnete Datei wird ein 512-Byte-I/O-Puffer im RAM-Speicher zwischen Programm und Variablen bereitgestellt, und der Positionszeiger wird auf Feld 0 gesetzt.
Danach können die Variablen V1, V2 usw. sequentiell mit
WRITELN (TEXTFILE, V1);
WRITELN (TEXTFILE, V2); usw.
gespeichert und mit
READLN (TEXTFILE, V1);
READLN (TEXTFILE, V2); usw.
eingelesen werden. Ein ProDOS-CLOSE wird am Ende des Kyan-Programms *automatisch* durchgeführt, so daß man die Diskette nicht vorzeitig aus dem Laufwerk nehmen sollte.

4.2. Datafiles

Eine **BIN**-Datei (= Datafile), die in Kyan-Pascal **FILE OF** + Datentyp heißt (FILE OF INTEGER, FILE OF REAL usw.) und mit **PUT** (= Schreiben) und **GET** (= Lesen) bearbeitet wird, ist eine „sequentielle Random-Datei" mit *gleich langen* Dateifeldern, die nicht durch Returns begrenzt werden und von 0 bis n durchnumeriert sind, z.B.
Feld 0: Hans□□ = 6 Zeichen
Feld 1: im□□□□ = 6 Zeichen
Feld 3: Glück□ = 6 Zeichen
Da keine „Loch"-Random-Dateien (s. „ProDOS für Aufsteiger", Bd. 2, S. 62) zulässig sind, kann auf BIN-Dateien sowohl sequentiell als auch – mit Hilfe des SEEK-Befehls – „at random", d.h. wahlfrei zugegriffen werden. Als Feldinhalte sind alle Pascal-Datentypen zulässig, die 1:1 gespeichert werden (z.B. bei Integer- und Real-Zahlen in binärer Form und nicht als ASCII-Ziffernfolgen).
Vor dem Öffnen einer Datei (ProDOS-OPEN) müssen der externe Dateiname, z.B.
CONST DATAFILE = '/VOL/DATAFILE';
und der interne Dateiname, z.B.
VAR DATAFILE: FILE OF INTEGER;
festgelegt werden.
Das Speichern einer *neuen Datei* beginnt mit
REWRITE (DATAFILE, DATANAME);
und das Einlesen oder Überschreiben einer *alten Datei* mit
RESET (DATAFILE, DATANAME);
Für die nunmehr geöffnete Datei werden ein I/O-Puffer sowie eine Dreieckstauschvariable (= Puffer-Variable = Fenster-Variable) namens DATAFILE↑ (interner Dateiname + ↑) bereitgestellt. Während REWRITE den Positionszeiger auf Feld 0 setzt und sonst nichts macht, wird bei RESET (1) der Positionszeiger auf Feld 0 gesetzt, (2) Feld 0 der Variablen DATAFILE↑ zugewiesen und (3) der Positionszeiger auf Feld 1 vorgerückt. Mit anderen Worten: Nach RESET ist bereits Feld 0 eingelesen. Um diese unnötige Komplikation zu vermeiden, die noch von der Lochstreifenära herrührt, ersetze man
RESET (DATAFILE, DATANAME);
durch den Doppelbefehl
RESET (DATAFILE, DATANAME);
SEEK (DATAFILE, 0);
Um die Variablen V1, V2 usw. zu speichern oder einzulesen, muß man die ebenfalls aus der Lochstreifenära stammende Dreieckstauschvariable zu Hilfe nehmen.
Schreiben:
DATAFILE↑ := V1;
PUT (DATAFILE);
DATAFILE↑ := V2;
PUT (DATAFILE); usw.
Lesen:
GET (DATAFILE);
V1 := DATAFILE↑;
GET (DATAFILE);
V2 := DATAFILE↑; usw.
Bei Datafiles ist wie bei Textfiles ein ProDOS-CLOSE entbehrlich.

Sonstiges

1 Mit dem nur bei Datafiles gültigen Befehl
SEEK (DATAFILE, Feldnummer);
läßt sich ein beliebiges Feld „at random" ansteuern.

2 Mit dem booleschen Befehl
IF **EOF** (DATAFILE) THEN...
kann sowohl bei Datafiles als auch bei Textfiles geprüft werden, ob das Dateiende (= End Of File) bereits erreicht wurde.

3 **INPUT** und **OUTPUT** sowie sonstige Dateinamen im Programmkopf werden ignoriert, weil die Standard-Ausgabe in Kyan-Pascal über **PR** umgelenkt wird.

Kurzhinweise zur Sammeldisk

Die Peeker-Sammeldisk enthält die Quelltexte
DATENDEMOS
SUBDEMOS
STRINGDEMOS
FILEDEMOS
– Quelltexte über DOSTOPRO auf Ihre Kyan-ProDOS-Arbeitsdiskette konvertieren.
– Quelltexte über PC compilieren.
– Mittleren Teil von STRINGDEMOS als Include-File gesondert abspeichern.

STRINGDEMOS

```
PROGRAM STRINGDEMOS;

CONST
 STRLEN  = 10; {Name 'STRLEN' erforderlich}
TYPE
 STRING  = ARRAY [1..STRLEN] OF CHAR; {Name 'STRING' erforderlich}

VAR   S, G, T, T1, T2: STRING; L, P, A: INTEGER; {Nur fuer Demo}
{=============================================================}
{String-Include-Datei, zusammengestellt von U.Stiehl im Dez. 1985}
{
String-Befehle
===============

L := LENGTH (S);       Integer-Funktion: L wird geaendert
G := CONCAT (T1,T2);   String-Funktion:  G wird geaendert
P := POS (T,G);        Integer-Funktion: P wird geaendert
T := COPY (G,P,A);     String-Funktion:  T wird geaendert
DELETE (G,P,A);        String-Prozedur:  G wird geaendert
INSERT (T,G,P);        String-Prozedur:  G wird geaendert

S = String,    G = Gesamtstring,   T = Teilstring, ferner T1, T2
L = Laenge,    P = Position,       A = Anzahl der Zeichen

Im Hauptprogrammkopf muessen definiert werden:
als CONST STRLEN mit Integer-Wert >= 2,
als TYPE  STRING mit 1..STRLEN.
}
{-------------------------------------------------------------------}
{
LENGTH ermittelt Netto-Laenge ohne Leertasten am String-Ende, d.h.
Leertasten links werden mitgezaehlt, Leertasten rechts abgezogen.
Aufrufen mit L := LENGTH (S). Beispiele:

      123  Zaehlleiste
S := 'abc' L = 3: Keine Leertasten.
S := '   ' L = 0: Nur Leertasten = Nullstring.
S := 'a  ' L = 1: Leertasten rechts entfernt.
S := ' a ' L = 2: Fuehrende Leertaste zaehlt mit.
S := 'a c' L = 3: Leertaste in der Mitte zaehlt mit.
}
FUNCTION LENGTH (S:STRING):INTEGER;
VAR L:INTEGER;
BEGIN
 L := STRLEN; WHILE (S[L] = ' ') AND (L > 1) DO L := L - 1;
 IF (L = 1) AND (S[1] = ' ') THEN LENGTH := 0 ELSE LENGTH := L
END;
{-------------------------------------------------------------------}
{
CONCAT vereinigt die 2 Teilstrings T1 und T2 zum Gesamtstring G.
Aufrufen mit G := CONCAT (T1,T2). Beispiel:

 1234567890  Zaehlleiste
      P
'abcde     ' T1
'fghij     ' T2
'abcde     ' Zunaechst T1 in G ab P = 1
'abcdefghij' Dannn T2 in G ab P = 6
Wenn T1 und T2 nicht zusammen in G passen, wird zunaechst T1 in G
kopiert und dann soviel von T2, wie in G noch Platz ist.
}
 FUNCTION CONCAT (T1,T2: STRING):STRING;
 VAR I,J,P:INTEGER; G:STRING;
 BEGIN
 P := 0;
 FOR I := 1 TO STRLEN DO
 BEGIN
  G[I] := T1[I]; {T1 ab links in G kopieren}
  IF (G[I] <> ' ') THEN P := I
 END;
 J := 1;
 FOR I := P TO STRLEN-1 DO
 BEGIN
  G[I+1] := T2[J]; {T2 ab Leertasten in G kopieren}
  J := J + 1
 END;
 CONCAT := G
END;
{-------------------------------------------------------------------}
{
POS (= Instring) ermittelt die Position P des Teilstrings T im
Gesamtstring G. Wenn T in G nicht enthalten ist,
wir P auf 0 gesetzt. Aufrufen mit P := POS (T,G). Beispiel:

 123456  Zaehlleiste
  P      P errechnet
'Peeker' G
'ee    ' T, ab P in G
}
 FUNCTION POS (T,G:STRING):INTEGER;
 VAR I,J,L:INTEGER; F: BOOLEAN;
 BEGIN
 L := STRLEN;
 WHILE (T[L] = ' ') AND (L <> 1) DO L := L - 1; {Laenge von T}
 I := 0; {Zaehler fuer G}
 REPEAT
  J := 1; {Zaehler fuer T}
  F := TRUE; {Flag}
  WHILE (J <= L) DO
  BEGIN
   IF (G[I+J] <> T[J]) THEN F := FALSE;
   J := J + 1;
  END;
  I := I + 1
 UNTIL (F = TRUE) OR (L+I > STRLEN);
 IF (F = TRUE) THEN POS := I ELSE POS := 0
END;
{-------------------------------------------------------------------}
{
COPY (= Substring) extrahiert den Teilstring T aus dem
Gesamtstring G ab Position P in der Anzahl A Zeichen;
bei fehlerhaften Parametern soweit wie moeglich.
Aufrufen mit T := COPY (G,P,A). Beispiel: T := COPY (G,4,3)

 123456  Zaehlleiste
    P    P = 4
    123  A = 3
'Pascal' G
'cal   ' T
}
FUNCTION COPY (G:STRING; P,A:INTEGER):STRING;
VAR I:INTEGER; T:STRING;
BEGIN
 FOR I := 1 TO STRLEN DO T[I] := ' '; {Loeschen}
 FOR I := 1 TO A DO {Wenn A < 1, dann Exit!}
 IF (P+I <= STRLEN+1) AND (P>0) THEN T[I] := G[P+I-1];
 COPY := T
END;
{-------------------------------------------------------------------}
{
DELETE entfernt aus Gesamtstring G die Anzahl A Zeichen
ab Position P. Danach ist G rechts mit der Anzahl A
Leerzeichen aufgefuellt. Bei zu grossen Werten
von P und A wird entfernt und geloescht, soweit es geht.
Aufrufen mit DELETE (G,P,A). Beispiel: DELETE (G,5,5)

 1234567890  Zaehlleiste
     P       P = 5
     12345   A = 5
'Kyan-Kurs:' G vorher
'Kyan:     ' G nachher
}
PROCEDURE DELETE (VAR G: STRING; P,A: INTEGER);
VAR I: INTEGER;
BEGIN
IF (P<1) THEN P := 1; IF (P > STRLEN) THEN P := STRLEN; {illegal}
WHILE (P+A > STRLEN+1) DO A := A - 1; {illegaler A-Wert}
FOR I := P TO STRLEN-1 DO IF (I+A <= STRLEN)
 THEN G[I] := G[I+A] ELSE G[I] := ' ';
FOR I := STRLEN DOWNTO STRLEN-P+1 DO IF (I>0) THEN G[I] := ' '
END;
{-------------------------------------------------------------------}
{

INSERT fuegt den Teilstring T in den Gesamtstring G
ab Position P ein. Wenn P zu gross ist,
wird G von rechts entsprechend gekuerzt.
Aufrufen mit INSERT (T,G,P). Beispiel INSERT (T,G,5)

 1234567890  Zaehlleiste
    P        P = 4
'einen     ' G vorher
'fueg      ' T
'einfuegen ' G nachher
}
PROCEDURE INSERT (T:STRING; VAR G:STRING; P:INTEGER);
VAR I,J,L:INTEGER;
BEGIN
IF (P<1) THEN P := 1; IF (P>STRLEN) THEN P := STRLEN; {Fehler}
L := STRLEN; WHILE (T[L] = ' ') AND (L>1) DO L := L - 1;
IF (L<>1) OR (T[1] <> ' ') THEN {T als Nullstring ignorieren}
 BEGIN
 J := 1;
 WHILE (P <= STRLEN) AND (J <= L) DO
  BEGIN
   FOR I := STRLEN-1 DOWNTO P DO G[I+1] := G[I];
   G[P] := T[J]; P := P + 1; J := J + 1
  END
 END
END;
```

```
{═══════════════════════════════════════════════════════}
{Demos}

BEGIN
WRITELN ('Length-Demo ------------------------------------------');
     {1234567890 Zaehlleiste}
S := 'ABCDEFGHIJ'; L := LENGTH (S); WRITELN (L); {10}
S := 'A         '; L := LENGTH (S); WRITELN (L); {1}
S := '          '; L := LENGTH (S); WRITELN (L); {0}
S := ' A A A A A'; L := LENGTH (S); WRITELN (L); {10}

WRITELN ('Concat-Demo ------------------------------------------');
      {1234567890 Zaehlleiste}
T1 := 'ABCDE     ';
T2 := 'FGHIJ     '; G := CONCAT (T1,T2); WRITELN (G);
     {'ABCDEFGHIJ'}
T1 := '          ';
T2 := 'FGHIJ     '; G := CONCAT (T1,T2); WRITELN (G);
     {'FGHIJ     '}
T1 := 'ABCDE     ';
T2 := '          '; G := CONCAT (T1,T2); WRITELN (G);
     {'ABCDE     '}
T1 := '         A';
T2 := '         B'; G := CONCAT (T1,T2); WRITELN (G);
     {'         A' Fehler}
T1 := 'ABCDEFGHI ';
T2 := 'ABCDEFGHIJ'; G := CONCAT (T1,T2); WRITELN (G);
     {'ABCDEFGHIA' Fehler}
T1 := '          ';
T2 := '          '; G := CONCAT (T1,T2); WRITELN (G, '*')
    :{'          '}

WRITELN ('Pos-Demo ---------------------------------------------');
     {1234567890 Zaehlleiste}
G := 'ABCDEFGHIJ';
T := 'BC        '; P := POS (T,G); WRITELN (P); {2}
G := 'ABCDEFGHIJ';
T := 'EFGHIJ    '; P := POS (T,G); WRITELN (P); {5}
G := '          ';
T := '          '; P := POS (T,G); WRITELN (P); {1}
G := 'AAAAAAAAAA';
T := 'A         '; P := POS (T,G); WRITELN (P); {1}
G := 'AAAAAAAAAA';
T := 'B         '; P := POS (T,G); WRITELN (P); {0}
G := '         A';
T := '         A'; P := POS (T,G); WRITELN (P); {1}

WRITELN ('Copy-Demo --------------------------------------------');
     {1234567890 Zaehlleiste}
G := 'ABCDEFGHIJ';
T := COPY (G,1,5); WRITELN (T); {'ABCDE'}
T := COPY (G,6,5); WRITELN (T); {'FGHIJ'}
T := COPY (G,9,8); WRITELN (T); {'IJ', Teilkopie}
T := COPY (G,0,8); WRITELN (T, '*'); {fehlerhaft}
T := COPY (G,0,0); WRITELN (T, '*'); {fehlerhaft}

WRITELN ('Delete-Demo ------------------------------------------');
     {1234567890 Zaehlleiste}
G := 'ABCDEFGHIJ'; DELETE (G,1,10); WRITELN (G, '*');
    {'         '}
G := 'ABCDEFGHIJ'; DELETE (G,5,5); WRITELN (G);
    {'ABCDJ     '}
G := 'ABCDEFGHIJ'; DELETE (G,2,2); WRITELN (G);
    {'ADEFGHIJ  '}
G := '         A'; DELETE (G,1,9); WRITELN (G);
    {'A         '}
G := 'A         '; DELETE (G,1,1); WRITELN (G, '*');
    {'          '}

WRITELN ('Insert-Demo ------------------------------------------');
     {1234567890 Zaehlleiste}
G := 'ABCDEJ    ';
T := 'FGHI      '; INSERT (T,G,6); WRITELN (G);
    {'ABCDEFGHIJ'}
G := 'J         ';
T := 'ABCDEFGHI '; INSERT (T,G,1); WRITELN (G);
    {'ABCDEFGHIJ'}
G := '     FGHIJ'; {Fuehrende Leertasten gelten!}
T := 'ABCDE     '; INSERT (T,G,1); WRITELN (G);
    {'ABCDE     '  Fehler}
G := '      AAA ';
T := 'BBB       '; INSERT (T,G,7); WRITELN (G);
    {'      BBBA'  Fehler}
G := '          ';
T := 'ABC       '; INSERT (T,G,5); WRITELN (G);
    {'    ABC   '}
END.
```

FILEDEMOS

```
PROGRAM FILEDEMOS;

CONST
 MAX      = 100;
 DATANAME = '/RAM/DATAFILE';
 TEXTNAME = '/RAM/TEXTFILE';
VAR
 I:         INTEGER;
 ZAHLEN:    ARRAY [0..MAX] OF INTEGER;
 DATAFILE: FILE OF INTEGER;
 TEXTFILE: FILE OF TEXT;
{═══════════════════════════════════════════════════════}
PROCEDURE INITARRAY;
BEGIN FOR I := 0 TO MAX DO ZAHLEN [I] := I; END;

PROCEDURE CLEARARRAY;
BEGIN FOR I := 0 TO MAX DO ZAHLEN [I] := 0; END;

PROCEDURE SHOWARRAY;
BEGIN FOR I := 0 TO MAX DO WRITE (ZAHLEN [I]:5);
WRITELN; END;
{═══════════════════════════════════════════════════════}
BEGIN
WRITELN ('Neuen Datafile speichern: Rewrite, Put');
INITARRAY;
REWRITE (DATAFILE, DATANAME);
FOR I := 0 TO MAX DO
 BEGIN
 DATAFILE↑ := ZAHLEN [I]; PUT (DATAFILE)
 END;
SHOWARRAY;

WRITELN ('Alten Datafile einlesen: Reset, Seek 0, Get');
CLEARARRAY;
RESET (DATAFILE, DATANAME); SEEK (DATAFILE, 0);
FOR I := 0 TO MAX DO
 BEGIN
 GET (DATAFILE); ZAHLEN [I] := DATAFILE↑;
 END;
SHOWARRAY;
{═══════════════════════════════════════════════════════}
WRITELN ('Alten Datafile ueberschreiben: Reset, Seek 0, Put');
INITARRAY;
RESET (DATAFILE, DATANAME); SEEK (DATAFILE, 0);
FOR I := 0 TO MAX DO
 BEGIN
 DATAFILE↑ := ZAHLEN [I]; PUT (DATAFILE);
 END;
SHOWARRAY;

WRITELN ('Alten Datafile at random einlesen: Reset, Seek, Get');
CLEARARRAY;
RESET (DATAFILE, DATANAME);
FOR I := MAX DOWNTO 0 DO {rueckwaerts}
 BEGIN
 SEEK (DATAFILE, I); GET (DATAFILE); ZAHLEN [I] := DATAFILE↑;
 END;
SHOWARRAY;
{═══════════════════════════════════════════════════════}
WRITELN ('Neuen Textfile speichern: Rewrite, Writeln');
INITARRAY;
REWRITE (TEXTFILE, TEXTNAME);
FOR I := 0 TO MAX DO WRITELN (TEXTFILE, ZAHLEN [I]);
SHOWARRAY;

WRITELN ('Alten Textfile einlesen: Reset, Readln');
CLEARARRAY;
RESET (TEXTFILE, TEXTNAME);
FOR I := 0 TO MAX DO READLN (TEXTFILE, ZAHLEN [I]);
SHOWARRAY;
{═══════════════════════════════════════════════════════}
WRITELN ('Datafile unbekannter Laenge mit EOF einlesen');
RESET (DATAFILE, DATANAME);
WHILE NOT EOF (DATAFILE) DO
 {Erst Write, dann Get!}
 BEGIN WRITE (DATAFILE↑:5); GET (DATAFILE); END;
WRITELN;

WRITELN ('Textfile unbekannter Laenge mit EOF einlesen');
RESET (TEXTFILE, TEXTNAME);
WHILE NOT EOF (TEXTFILE) DO
 {Erst Readln, dann Write!}
 BEGIN READLN (TEXTFILE, I); WRITE (I:5); END;
WRITELN;
{═══════════════════════════════════════════════════════}
END.
```

PEEKER Börse

Gelegenheitsanzeigen

Sie können unter dieser Rubrik zu einem besonders günstigen Preis

- Ihre Hardware und Software verkaufen
- Ihre Hard- und Software suchen
- Kontakte knüpfen und vieles mehr

Musteranzeige privat

1 Druckzeile à 32 Buchstaben nur DM 5,50 zuzügl. ges. MwSt. Mindestens 2 Druckzeilen
Beispiel:

Verkaufe neuwertigen Typenraddrucker mit Apple-Interface. Preis auf Anfrage. Tel. 007

nur DM 18,81 inkl. MwSt.

Musteranzeige gewerblich

1 Druckzeile à 32 Buchstaben nur DM 11,– zuzügl. ges. MwSt. Mindestens 2 Druckzeilen
Beispiel:

Neu im Angebot: Professionelle, separate Tastatur für Apple II plus 16 Funktionstasten und separatem Ziffernblock.
Fa. Keyboard & Co.

nur DM 62,70 inkl. MwSt.

PEEKER

Peeker-Börse

AUFTRAG FÜR KLEINANZEIGEN

Bitte veröffentlichen Sie in der nächsterreichbaren Ausgabe nachstehenden Text unter folgender Rubrik:

☐ suche Hardware ☐ suche Software ☐ Verschiedenes
☐ biete Hardware ☐ biete Software ☐ Chiffre

Bitte jeweils 32 Buchstaben pro Zeile – einschließlich Satzzeichen und Wortzwischenräume. Bitte Absender nicht vergessen. Mindestens 2 Zeilen. Chiffregebühr DM 6,– zuzügl. MwSt.

PEEKER

Produkt-Karte

Zu der im **Peeker**, Heft _____, Seite _____ erschienenen

Anzeige über _____

_____

bitte ich um detaillierte Information.

Ich wünsche ☐ Prospekt, Datenblatt ☐ Preisliste ☐ schriftliches Angebot ☐ tel. Rückruf

Menge	Produkt und Bestellnummer	à DM	gesamt DM

gebe ich nebenstehende Bestellung unter Anerkennung Ihrer in der Anzeige genannten Liefer- und Zahlungsbedingungen auf.

_____

Unterschrift (für Jugendliche unter 18 Jahren der Erziehungsberechtigte)

PEEKER

Info-Karte

Schreiben Sie uns, wenn Sie technische Fragen zu Aufsätzen im **Peeker** haben. Beachten Sie jedoch, daß die Redaktion keine Auskünfte über Bezugsquellen erteilt:

_____

_____

_____

_____

_____

_____

Peeker-Börse

Vorname, Name

Firma

Straße

Wohnort

PLZ/Ort

Bitte veröffentlichen Sie den umstehenden Text von _____ Zeilen à _____ DM in der nächsterreichbaren Ausgabe vom **Peeker**

Bei Angeboten: Ich bestätige, daß ich alle Rechte an den angebotenen Sachen besitze

Datum Unterschrift

Produkt-Karte

Karte bitte vollständig ausfüllen

Vorname, Name

Firma

Straße

PLZ/Ort

Telefon mit Vorwahl

Anschrift der Firma angeben, bei der Sie bestellen bzw. von der Sie Informationen wünschen

Info-Karte

Karte bitte vollständig ausfüllen

Vorname, Name

Firma

Straße

PLZ/Ort

Telefon mit Vorwahl

POSTKARTE

Peeker-Börse
Anzeigen-Service
Dr. Alfred Hüthig Verlag
Postfach 10 28 69
6900 Heidelberg 1

POSTKARTE

Inserent

Straße

PLZ/Ort

POSTKARTE

Peeker
Redaktion
Dr. Alfred Hüthig Verlag
Postfach 10 28 69
6900 Heidelberg 1

Produkt-Karte

Wünschen Sie weitere Informationen zu einer der im Heft erschienenen Anzeigen?

Nichts einfacher als das. Produkt-Karte ausfüllen, frankieren und an den Inserenten (nicht an die Peeker-Redaktion senden.

Vorher aber nicht vergessen:
Kreuzen Sie an, welchen Informationswunsch Sie haben.

Damit erleichtern Sie dem Hersteller eine gezielte Beantwortung Ihrer Anfrag

Zum Schluß tragen Sie auf der Rückseite die genaue Anschrift des Inserenten und Ihren Absender ein.

PEEKER

Verpassen Sie Ihrem Apple den richtigen Biß mit der C86 Steckkarte

Die C86-Karte ist ein kompletter 16-Bit Mikrocomputer mit eigenem Speicherbereich (512KB) und Schnittstelle zum Applebus.

Die Erweiterung bietet Ihnen:

- einen leistungsfähigen 16-Bit Mikrocomputer mit Intel 8086CPU, es sind fast alle betriebssystemgestützten IBM PC Programme auf der C86-Karte lauffähig
- serienmäßig 128KB oder 512KB Speicherkapazität
- ein spezielles Applebusinterface für den sicheren Betrieb
- alle notwendigen Teile auf einer Apple-Slotkarte

Mit dieser Erweiterung entspricht Ihr Apple dem neuesten Stand der Technik.

C86 mit 128K **nur 598,– DM**
C86 mit 512K **nur 998,– DM**

D86, Diskettencontroller zur C86-Karte

- ermöglicht das Einlesen aller handelsüblichen Diskettenformate (MS-DOS, CPM, Apple-DOS) in den Apple od. compatible Computer.

D86 298,– DM

IBM AT COMPATIBLER MIKRO-COMPUTER **ab 7500,–DM**
IBM XT COMPATIBLER MIKRO-COMPUTER **ab 1600,–DM**
HANDBUCH FÜR IBM COMPATIBLE COMPUTER 68,–DM

Händleranfragen erwünscht: Anton Peter & Partner
Brüsseler Straße 16
1000 Berlin 65

alle Preise incl. 14%MWST, zuzgl. Verpackung u. Porto
Apple und IBM sind eingetragene Markenzeichen

CSW UPC II (eine universelle Programmierkarte)

für alle APPLE-II- und kompatiblen Rechner

- jetzt auch mit 12,5 V Brennspannung
- Komfortable Bedienungssoftware mit Menüsteuerung DOS 3.3 – keine Umschaltung über DIP-Switches auf der Karte!
- Anzeige und Sockel auf externer Platine – 28poliger Nullkraftsockel – 500-mm-Flachbandkabel
- EPROM-Typen 2516, 2716, 2732, 2732A, 2764, 2764A, 27128, 27128A

DM 580.–

CSW 72 I/O-Port-Card

- 9 programmierbare 8-Bit-I/O-Ports
- lauffähig unter allen Betriebssystemen

DM 350.–

WEISS COMPUTER Dipl.-Psych. Karl-Heinz Weiß
Am Wiesenhof 17, 2940 Wilhelmshaven, Tel. 0 44 21/8 31 79

APPLE & CP/M-80 & MS-DOS SOFTWARE & HARDWARE

z. B. für APPLE II und Kompatible

Wir liefern die RAM-Karte (AE) für den Apple IIe mit max. 3 MB (Appleworks mit mehr als 2 MB)! 64-K-Ausf. . . . . DM 620.–
SpeeDemon 3.56 MHz Coproz. für II+/e (McT) . . . . . DM 980.–
UPC-Programmer-Card 2716-128 komfortabel . . . . . DM 580.–
72 I/O Port Card programmierbar DOS+CP/M . . . . . DM 350.–
AD 16 Ch. 12 Bit, schnell! (Applied Eng.) . . . . . DM 1400.–
PKASO/U-Printer-Karte (IS) . . . . . DM 550.–
CP/M-Plus-Card, 6 MHz, 64 K, CP/M 3.0 (ALS) . . . . . DM 1250.–
NicePrint-Printer-Karte (Spies Loborat.) . . . . . DM 490.–
Timemaster II H. O., die Uhrenkarte! (AE) . . . . . DM 540.–
Softerm 2 II+/e/c (Softronics) . . . . . DM 800.–
ELF kompl. Statistik-Software (TWG) . . . . . DM 800.–
Prime-Plotter-Grafik-Software (Primesoft) . . . . . DM 960.–
TTL-IC-Tester inkl. Software . . . . . DM 399.–
Memory-Tester & Eprom-Writer/Tester . . . . . DM 550.–
Z-RAM 512 K für APPLE IIc (AE) . . . . . DM 1350.–

z. B. für IBM und Kompatible

APPLE Turnover (Vertex) Lesen/Schreiben von Apple Disks im IBM PC & Komp. . . . . DM 1200.–
XENO-COPY plus (Vertex) Lesen/Schreiben div. CP/M & MS-DOS Formate im IBM . . . . . DM 600.–
ELF PC kompl. Statistik Software (TWG) . . . . . DM 800.–
PROM Blaster 28-Pin (Apparat Inc.) . . . . . DM 620.–

z. B. für alle Systeme

Printerchanger 3 paral. Drucker auf 1 Micro inkl. Kabel/Netzteil (Keyzone) . . . . . DM 600.–
Printersharer 3 Micros auf 1 paral. Drucker inkl. Kabel/Netzteil (Keyzone) . . . . . DM 500.–
Shufflebuffer 64 K (IS) . . . . . DM 1350.–

Wir sind Import-Spezialisten und bieten Ihnen eine große Auswahl an Software und Hardware bedeutender Hersteller aus den USA und England. Informationen gegen DM 3.– in Briefmarken.

WEISS COMPUTER Dipl.-Psych. Karl-Heinz Weiß
Am Wiesenhof 17, 2940 Wilhelmshaven, Tel. 0 44 21/8 31 79

Apple II als Datenanalysesystem

Mit der Erweiterungskarte **Metascope** kann der Apple II, II+ und IIe als interaktives Datenanalysesystem eingesetzt werden. Es lassen sich damit serielle Schnittstellen synchron und asynchron bis 19200 Baud analysieren. Sowohl TY- als auch BSC-, SDLC- und HDLC-Übertragung wird unterstützt. Darstellbar sind die Zeichen ASCII, EBCDIC, Baudot und Hexadezimal.

Bitte besuchen Sie uns auf der Hannover Messe **CeBit** Halle 7, Stand 106.

Lange Communications
LANGE & Co. GmbH
Ünninghauser Str. 70, 4780 Lippstadt
Tel. (02945) 5449, Tx. 841952 langed

Apple und IBM kompatible Computer

16K, Z80, Diskcontroler je 75,–
80 Zeichenkarte mit Softswitch 2 Zeichensätze 149,–
IIe-kompatibles Motherboard ohne Firmware 498,–
Erphi-controler mit Autopatch . . . 285,–
TEAC FD-54A mit Apple-Bus 298,–
TEAC FD-55B 2 × 40 Track 375,–
TEAC FD-55F 2 × 80 Track 395,–
FD4 Spezialcontroler für Laufwerke mit bis zu 2 × 80 Track 120,–
OLYMPIA compact NP 1298,–
Monochrome Monitore . . . . ab 375,–
Farbmonitore . . . . . ab 998,–
Tastaturen für IBM und Apple . ab 298,–
512K-RAM-Karte mit 256 K bestückt 298,–
Apple Super-Modem-Karte inkl. dt. Software und dt. Handbuch 348,–

Versand nur per Nachnahme oder Vorkasse.
Weiteres Zubehör für Apple und IBM gegen frankierten Rückumschlag.
128K Karte (Saturn kompatibel) . **248,–**

Ulf Mohwinkel Electronic
Berliner Straße 73 Pf: 250166
5090 Leverkusen Fettehenne
Telefon 02 14/9 37 81 od. 9 50 60

Berlin im Apple IIe 384 x 288 Bildpunkte.

VIDEO-1000

DIGITIZER ZUM APPLE IIE

INTERFACE+ SOFTWARE **295,-DM**

* Auflösung 384x288 Bildpunkte
** für TV, Recorder und Kamera
*** Aufnahmezeit 20 msec
**** Umrechnung auf HIRES-Seite

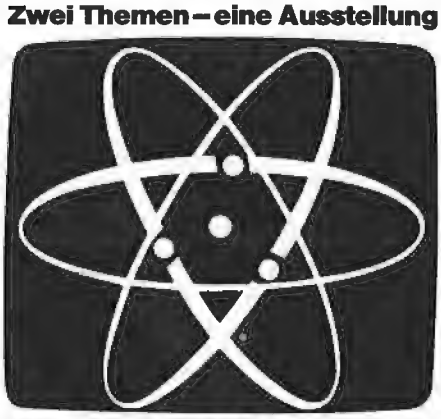
Erweiterte Software z.B. Bilder mit bis zu 300.000 Bildpunkten, 7 Graustufen, Doubble Hires, Kurzfilmerstellung etc auf Anfrage.

256 K RAM-Karte mit Software . . . . . 425,- DM
192K RAM-Karte mit Software . . . . . 395,- DM
64 K AKKU-RAM-Karte mit Software . . . . . 195,- DM
128K AKKU-RAM-Karte mit Software . . . . . 295,- DM

Gewerbl. gegen Einsendung von 10 DM oder V-Scheck. Info gratis. Versand p.NN. oder beim Fachhändler.

Ing.-Büro M.Fricke, Neue Str.13, 1000 Berlin 37
Telefon: 030 / 801 60 62

Zwei Themen – eine Ausstellung



Hobby-tronic

9. Ausstellung für Funk- und Hobby-Elektronik

COMPUTER-SCHAU

2. Ausstellung für Computer, Software und Zubehör

Dortmund
23.–27. April 1986

Die umfassende Marktübersicht für Hobby-Elektroniker und Computer-Anwender, klar gegliedert:

In Halle 5 das Angebot für CB- und Amateurfunker, Videospieler, DX-er, Radio-, Tonband-, Video- und TV-Amateure, für Elektro-Akustik-Bastler und Elektroniker. Mit dem Actions-Center und Laborversuchen, Experimenten, Demonstrationen und vielen Tips.

In Halle 4 das Super-Angebot für Computer-Anwender in Hobby, Beruf und Ausbildung. Dazu die „Computer-Straße", als Aktionsbereich, der Wettbewerb „Jugend programmiert" und der Stand des WDR-Computer-Clubs.

Ausstellungsgelände Westfalenhallen Dortmund täglich 9.00-18.00 Uhr

TIPS UND TRICKS IN PASCAL

Teil 6: Der interne Aufbau von Files

von Dieter Geiß

In dieser und der nächsten Folge soll auf den internen und den externen Aufbau von Files eingegangen werden. Mit internem Aufbau ist gemeint, wie Files im Speicher repräsentiert werden. Der externe Aufbau beschäftigt sich mit der Darstellung der Files auf Diskette. Wegen der Länge des Artikels wird Teil 6 der Serie den internen und Teil 7 den externen Aufbau beschreiben.
Um sowohl den internen wie den externen Aufbau zu verstehen, wollen wir von einem kleinen Programm ausgehen, welches in **Listing 1: FileTest** dargestellt wird.
Anhand dieses Listings soll erklärt werden, was beim Übersetzen und Ausführen des Programms genau geschieht.

In Zeile 1 wird durch die Compiler-Option ein Listfile erzeugt, der hier auf den Drukker ausgegeben wird. Die Änderung des Wortes PRINTER in CONSOLE bewirkt, daß der Listfile auf dem Bildschirm erscheint. Ein Listfile sollte erzeugt werden, wenn man an den Adressen und Längen der einzelnen Variablen interessiert ist.

1. Filetypen

1.1. Text- und Interaktivfiles

Der File **F1** (s. Listing 1) wurde als **Textfile** definiert, d.h. als „file of char", was dasselbe ist. Er benötigt 301 Words (= 602 Bytes) Speicherplatz. Warum? 300 Words werden vom Compiler immer für einen gepufferten File reserviert. Der Inhalt dieser 300 Words wird aus der Definition für einen File-Information-Block (s.u.) ersichtlich. Ein weiteres Word muß für den Typ des Files, im Falle des Textfiles ein Zeichen (char), reserviert werden, was 301 Words ausmacht. Dieses Word beinhaltet den Wert der Variablen F1 ↑, also der sog. Fenstervariablen. Sie wird automatisch mit definiert, d.h.
var F1 : text;
entspricht
var F1 : ↑ char;

Die Fenstervariable stellt immer ein Objekt oder Element des Filetyps zur Verfügung. Über sie kann mit Get und Put ein File gelesen oder geschrieben werden.
Der File **F2**, der als **interaktiv** bezeichnet wird, benötigt ebenfalls 301 Words. Auch er ist im Prinzip ein „file of char". Der Unterschied zwischen Textfiles und Interaktivfiles besteht in der Behandlung beim

Öffnen des Files durch den Reset-Befehl. Während beim Textfile eine automatische Get-Operation durchgeführt wird, wird sie beim Interaktivfile unterdrückt. Dies ist von Bedeutung, wenn ein File für den Zugriff auf Bildschirm und Tastatur geöffnet wird. Betrachten wir das Programmstück

```
reset (F, 'CONSOLE:');
write ('Bitte Taste drücken');
read (F, C);
```

Dabei sei C eine Char-Variable. Man erwartet normalerweise, daß das Programm den File F als Bildschirm mit Tastatur (CONSOLE) öffnet, danach die Meldung „Bitte Taste drücken" ausgibt, und daß schließlich das Programm beim Read-Befehl anhält, um die Taste von der Tastatur einzulesen. Ist F ein Interaktivfile, so läuft dies auch wie erwartet ab. Anders verhält es sich, wenn F ein normaler Textfile ist. Durch die implizite Get-Operation hält das Programm bereits beim Reset-Befehl an und wartet nun auf eine Eingabe eines Zeichens, um dem Aufruf der Get-Operation zu genügen. Man muß also im voraus wissen, ob ein Zeichen oder auch eine Zahl verlangt wird, denn das Programmstück hätte auch

```
reset (F, 'CONSOLE:');
write ('Bitte Zahl eingeben');
read (F, I);
```

lauten können, wobei I eine Integer-Variable ist. Die Benutzung von Interaktivfiles scheint also weniger Probleme mit sich zu bringen, wenn die Eingabe über die Tastatur erfolgt. Bei der Ausgabe spielt es keine Rolle, ob ein Text- oder Interaktivfile benutzt wird. Die vordefinierten Files „input", „output" und „keyboard" sind allesamt Interaktivfiles, die sich so verhalten, wie man es von ihnen erwartet.

1.2. Benutzerdefinierte Files

F3 ist ein File für Paare von Integer-Zahlen, also z.B. Koordinaten. Dazu wurde der Typ „Koord" eingeführt, der als **Verbund** (Record) zwei Integer-Zahlen enthalten kann. Der File F3 benötigt 302 Words. Die ersten 300 Words haben dieselbe Bedeutung wie bei den Textfiles; die beiden letzten Words beinhalten die Fenstervariable F3↑, die mit F3 definiert ist. Da sie dem Typ „Koord" entspricht, hat sie auch dessen Größe. Im Gegensatz zu Text- und Interaktivfiles sind bei Files mit einem anderen Typ als „Char" keine Read- oder Write-Befehle erlaubt, was in Standard-Pascal der Fall ist. Man hilft sich hier mit Get und Put, womit man die fehlenden Operationen ausgleicht. Ein

```
read (F3, K);
```

entspricht somit der Befehlsfolge

```
K := F3↑;
get (F3);
```

während ein

```
write (F3, K);
```

durch

```
F3↑ := K;
put (F3);
```

simuliert wird. Auch für Files mit einem benutzerdefinierten Typ gilt: Nach einem Reset wird ein Get durchgeführt, womit sich die Reihenfolge der beiden zu Read äquivalenten Befehle erklären läßt.

1.3. Typlose Files

F4 wurde als File definiert, der keinem Typ angehört. Daraus folgt, daß es auch keine Fenstervariable gibt, mithin F4↑ undefiniert ist. Außerdem gibt es keinen Puffer, der bei den anderen Filetypen benötigt wird, um einen Block von Diskette zu lesen oder auf Diskette zu schreiben. Als Vorteil ist anzuführen, daß mit Hilfe dieses Filetyps beliebige Datenstrukturen in einen File geschrieben werden oder von einem File gelesen werden können. Diese Datenstrukturen müssen nicht unbedingt ein „packed array of char" sein. Auch eine komplizierte kombinierte Record-Array-Struktur ist möglich. Die einzige Einschränkung ist, daß die Größe der benutzten Datenstruktur ein Vielfaches von 512 sein muß. Dies kann mit Hilfe der Sizeof-Funktion immer getestet werden. Die Variable F4 belegt im Gegensatz zu den anderen Filetypen nur 40 Words im Speicher, was am fehlenden Puffer liegt, der vom Programmierer zur Verfügung gestellt werden muß, wenn er Blockread oder Blockwrite aufruft.

Das Programm FileTest (Listing 1) öffnet vier Files verschiedener Datentypen und schreibt in diese Files bestimmte Werte. Was dann im einzelnen auf Diskette steht, nachdem man das Programm beendet hat, wird im Teil 7 über den externen Aufbau erklärt.

2. Die Tricks des Compilers

Sieht man sich den vom Programm FileTest erzeugten Code genau an, so fällt auf, daß der Compiler mehrere Befehle, die nicht im Programmtext stehen, dazuschmuggelt (vgl. Peeker 11/85, S. 58: Die Extended-File-Option). Für jeden File, der in einer Prozedur definiert wird (auch das Hauptprogramm ist schließlich nur eine Prozedur), werden zwei zusätzliche Anweisungen für den File eingefügt. Die erste Anweisung, die vor alle anderen Befehle in einer Prozedur – mit Ausnahme des Ladens von Segmenten – gesetzt wird, ist ein Aufruf der File-Init-Prozedur (Prozedur 3 des Pascalsystems). Sie wird mit drei Parametern aufgerufen:

1. Adresse der Filevariablen F
2. Adresse der Fenstervariablen F↑
3. Länge eines Datensatzes

Für 3. gilt:
Länge = 0 → Interaktivfile
Länge = -2 → Textfile
Länge = -1 → File ohne Typ
Länge > 0 → Länge in Words

In unserem Beispiel FileTest wird „Finit" mit folgenden Werten aufgerufen:
Für F1
1. Adresse von F1 (3)
2. Adresse von F1↑ (303)
3. Länge des Datensatzes (-2 für Textfile)

Für F2
1. Adresse von F2 (304)
2. Adresse von F2↑ (604)
3. Länge des Datensatzes (0 für Interaktivfile)

Für F3
1. Adresse von F3 (605)
2. Adresse von F3↑ (906)
3. Länge des Datensatzes (2 für Record)

Für F4
1. Adresse von F4 (907)
2. Adresse von F4↑ (0 = nil)
3. Länge des Datensatzes (-1 für File ohne Typ)

Die Adressen sind nicht absolut, sondern relativ zum globalen Datensegment (globale Variablen) zu verstehen. Was im einzelnen bei Finit geschieht, wird nach der Definition des File-Information-Blocks erklärt.

Am Ende einer Prozedur, in der Files definiert wurden, wird ein implizites

```
close (F, normal)
```

für jeden File F eingeführt, bevor eventuell geladene Segmente wieder freigegeben werden. Damit ist gewährleistet, daß ein File nach Ablauf eines Programms immer geschlossen wird, falls der Programmierer es vergessen sollte. Das Einfügen des impliziten Close-Befehls kann durch die Compiler-Option

```
{$E+}
```

verhindert werden, wie bereits in dem früheren Teil 4 dieser Serie dargelegt wurde.

3. Öffnen und Schließen

Über die Befehle „reset", „rewrite" und „close" wird in den Handbüchern genug geschrieben. Was ich hier ausführen möchte, sind Dinge, die vor allem Anfängern Schwierigkeiten machen.
Will man einen File neu öffnen, so benutzt man bekanntlich die Rewrite-Anweisung.

Tabelle 1: Öffnen und Schließen von Files

	RESET	File geschlossen mit !normal	lock	purge	crunch
File war vorher	da nicht da	! 1	1	2	3
		! 4	4	4	4
	REWRITE	**!normal**	**lock**	**purge**	**crunch**
File war vorher	da nicht da	! 5	6	5	7
		! 8	9	8	10

Legende zu Tabelle 1:

(1) File wird normal geschlossen
(2) File wird gelöscht
(3) File wird „gecruncht" geschlossen
(4) Input/Output-Error #10
(5) Alter File unverändert, neuer File gelöscht
(6) Alter File gelöscht, neuer File normal geschlossen
(7) Alter File gelöscht, neuer File „gecruncht" geschlossen
(8) Neuer File gelöscht
(9) Neuer File normal geschlossen
(10) Neuer File „gecruncht" geschlossen

Dazu muß man wissen, daß das Pascalsystem beim Öffnen in Ermangelung des Wissens, wie lange der File werden könnte, den größten zusammenhängenden Platz auf der Diskette reserviert. (Dies gilt nur für Diskettenfiles.) Öffnet man nun noch einen zweiten File, bevor der erste geschlossen worden ist, wird das Pascalsystem keinen Platz mehr vorfinden, wenn sich der File auf derselben Diskette befindet und vor dem Öffnen des ersten Files nur *ein* zusammenhängender Bereich vorhanden war. Es gibt mehrere Möglichkeiten, diesen Nachteil zu beheben:

a) Man schließt den ersten File vor dem Öffnen des zweiten, was jedoch praktisch nicht immer möglich ist.

b) Man benutzt für den zweiten File ein anderes Laufwerk, welches allerdings nicht immer vorhanden ist.

c) Man erwirkt durch geschickte Make- und Crunch-Anweisungen im Filer, daß mehrere zusammenhängende Bereiche auf der Diskette entstehen.

d) Man gibt beim Öffnen des Files eine Längenspezifikation an.

Die Längenspezifikation setzt voraus, daß man vor dem Öffnen des Files weiß, wieviele Blöcke der File ungefähr einnehmen wird. Wenn man einen „file of integer" hat und 500 Zahlen auf Diskette abspeichern möchte, so kann man sich ausrechnen, daß diese 500 Zahlen 500 Words = 1000 Bytes einnehmen werden, was in zwei Blöcken unterzubringen ist. Man öffnet die Datei dann z.B. mit
rewrite (F, 'INTEGER.DATA[2]')
wodurch das Pascalsystem veranlaßt wird, nur zwei Blöcke für diesen File auf der Diskette zu reservieren. Man kann statt dessen auch
rewrite (F, 'INTEGER.DATA[*]')
angeben, womit die Hälfte des größten noch zur Verfügung stehenden Bereiches oder der zweitgrößte Bereich für diesen File auf der Diskette reserviert wird.
Bei mehr als zwei Files, deren Länge vorher nicht geschätzt werden kann, sollte man stets die Spezifikation [*] verwenden.

Beim Schließen von Files gibt es die vier Optionen „normal", „lock", „crunch" und „purge". Je nachdem, ob man einen File mit „reset" oder „rewrite" geöffnet hat und der File schon vorhanden war oder nicht, ergeben sich 16 Kombinationen, die jedoch nicht alle verschieden sind. In **Tabelle 1** sind diese angegeben. Um ein Beispiel für das Verständnis der Tabelle zu geben: Die Sequenz
rewrite (F, 'TESTFILE');
put (F); etc.
close (F, normal);
bei der vorher schon ein File namens TESTFILE existiert hat, läßt den ursprünglichen File unverändert, während der neu geöffnete TESTFILE wieder gelöscht wird.

4. EOLN und EOF

Des öfteren taucht die Frage auf, wie man bei der Eingabe ein End-Of-Line (EOLN) oder ein End-Of-File (EOF) erkennen kann, wenn man von „input" bzw. „keyboard" liest. Angenommen, C sei eine Char-Variable und die Anweisung laute
read (input, C)
oder einfacher
read (C).
Beim Betätigen der Return-Taste wird der Variablen C ein Leerzeichen (ASCII SP) zugewiesen, während gleichzeitig die Funktion
eoln (input)
den Wert „true" ausgibt. Der Wert 13 (ASCII CR) tritt nie als Inhalt von C auf, womit Abfragen wie
if C = chr (13)
niemals das Ende einer Zeile erfassen. Die Funktion „eoln" kann man z.B. durch ein
readln (input)
zurücksetzen. Aber auch beim Einlesen eines neuen Zeichens, welches nicht das End-Of-Line-Zeichen ist, wird „eoln" zurückgesetzt.
Betätigt man die End-Of-Text-Taste Ctrl-C, wird ebenfalls ein Leerzeichen in C gebracht, während sowohl
eoln (input)
als auch
eof (input)
„true" ausgeben. Der Wert 3 (ASCII ETX) wird niemals als Inhalt von C auftreten, womit Abfragen wie
if C = chr (3)
sinnlos erscheinen. Nach Eingabe von Ctrl-C ist der File „input" geschlossen, womit jegliche weitere Eingabe unmöglich wird, was man durch das Statement
if eof (input) then reset (input)
wieder rückgängig machen kann. Wenn man dies nicht tut, überspringt das Programm jede weitere Eingabe von „input". Erst kurz vor Erscheinen der Kommandozeile wird die Ende-Anzeige des Files „input" wieder auf „false" gesetzt.
Etwas anders verhält sich die Sache bei dem File „keyboard". Für End-Of-Line gilt dasselbe wie für „input". Im Gegensatz zu „input" kann aber der File „keyboard" niemals geschlossen werden. Ein Close-Befehl ist wirkungslos; das gleiche gilt für das Einlesen des End-Of-File-Zeichens ETX. Die Funktion
eof (keyboard)
wird immer den Wert „false" liefern. Statt dessen wird aber das eingelesene Zeichen C nicht in ein Leerzeichen umgewandelt, sondern es behält den ASCII-Wert 3, womit die Abfrage
if C = chr (3) ...
sinnvoll ist.

5. File-Information-Block

Wie wir gesehen haben, beträgt die Größe einer Filevariablen im Speicher bei Files ohne Typ 40 Words und bei anderen Files 300 Words + x Words für die Fenstervariable.
Da ein Hexdump des Inhalts einer Fileva-

riablen nicht soviel aussagt wie eine Darstellung mit Hilfe einer korrekten Datenstruktur, seien hier alle Komponenten einer Filevariablen anhand der Struktur Fib = File-Information-Block aufgezeigt. Betrachten Sie dazu **Listing 2: FibTest**. Die Definitionen, die zum Directory gehören, wurden schon im Peeker, 1/1985 genauestens erklärt. Sie seien im Listing für Leser, die nicht über diese Ausgabe verfügen, noch einmal aufgeführt. Der Verbund „Fib" benötigt maximal 290 Words. Die letzten 10 Words werden vom Compiler reserviert, falls die Definition von „Fib" sich noch ändern sollte. Die einzelnen Komponenten haben folgende Bedeutung:

Fwindow enthält die Adresse der Fenstervariablen, die immer 300 Words größer ist als die Adresse der Filevariablen. Fwindow ist als Zeigervariable implementiert. Fwindow ↑ ist die Fenstervariable.

Feof ist „true", wenn der letzte Datensatz des Files gelesen worden oder ein Fehler beim Schreiben aufgetreten ist. Mit der vordefinierten Funktion EOF kann der Wert dieser Variablen abgefragt werden.

Feoln ist „true", wenn das End-Of-Line-Zeichen gelesen wurde, welches beim Apple-Pascal den ASCII-Wert 13 (CR) hat. Mit der vordefinierten Funktion EOLN kann der Wert dieser Variablen abgefragt werden, wenn es sich um einen Text- oder Interaktivfile handelt.

Fstate kann drei Zustände einnehmen, welche Fjandw, Fneedchar oder Fgotchar lauten. Alle Files außer Interaktivfiles haben den Status Fjandw. Bei diesem Status wird bei einem Reset-Befehl gleichzeitig ein erstes Get durchgeführt. Bei Interaktivfiles ist der anfängliche Status Fneedchar, weil noch ein Get durchgeführt werden muß, also ein Zeichen gebraucht wird. Nach einem Get ist der Status Fgotchar. Da die einzelnen Zeichen beim Einlesen einer Integer-Zahl über Get geholt werden, bewirkt ein nachfolgendes „read (C)", wobei C eine Char-Variable ist, daß das Zeichen, mit welchem die Integer-Zahl abgeschlossen wurde, in C eingelesen wird. Das Programm bleibt bei diesem Read-Befehl nicht stehen, was völlig richtig ist, da das abschließende Zeichen noch nicht verarbeitet ist. Die Sequenz

read (I);
read (C);

liest eine Integer-Zahl ein. Die Variable C enthält danach ein Leerzeichen, wenn die Zahl mit der Leer-, der Return- oder der ETX-Taste abgeschlossen wurde. Wurde hingegen die Integer-Zahl mit einem Buchstaben wie z.B. „x" abgeschlossen, so enthält C diesen Buchstaben.

Frecsize gibt die Größe der Fenstervariablen in Bytes an. Für Textfiles und Interaktivfiles gilt die Größe 1, nicht 2, was einem „packed file of char" entsprechen würde. Typlose Files haben die Größe 0.

Flock wird nur in Pascal 4.0, 4.1 usw. benutzt. (Es ist eine Semaphor-Variable, die benötigt wird, um Files vor gleichzeitigem Zugriff mehrerer Prozesse zu schützen. Diese Variable darf in Pascal 1.1 oder 1.2 nicht in die Definition von „Fib" aufgenommen werden.)

Fisopen gibt an, ob der File geöffnet ist oder nicht. Ist der File geöffnet, so existiert eine Variante mit weiteren Komponenten.

Fisblkd ist „true", wenn sich der geöffnete File auf einer Diskette befindet und nicht etwa den Bildschirm oder den Drucker usw. ansteuert.

Funit, die Unitnummer des Geräts, auf welchem sich der File befindet, läßt sich aus **Tabelle 2** entnehmen.

Fvid, der Name des Geräts, auf welchem sich der File befindet, läßt sich ebenfalls aus Tabelle 2 entnehmen.

Freptcnt gibt an, wie oft die Fenstervariable einen gültigen Wert enthält, bevor wieder ein Get-Befehl durchgeführt werden soll. Im praktischen Fall heißt das, daß beim Einlesen eines DLE-Zeichens von einem Textfile der nachfolgende Wert eingelesen wird und dieser Wert, nachdem er um 32 vermindert wurde, als Anzahl der nun folgenden Leerzeichen interpretiert wird, sofern er größer als 0 und kleiner als 127 ist. Der gültige Wert ergibt Freptcnt.

Fnxtblk ist der nächste Block, der von Diskette gelesen werden muß. Er wird hier relativ zum Fileanfang angegeben. Er wird beim Öffnen auf 0 gesetzt, es sei denn, es handelt sich um einen Textfile (Suffix TEXT), der über eine Filevariable, die nicht typlos ist, eingelesen wird. In diesem Fall werden die beiden ersten Blöcke übergangen, hingegen beim Schreiben mit Nullen gefüllt. Fnxtblk hat dann den Wert 2.

Fmaxblk hat den Wert 0, wenn ein neuer File geöffnet wird. Sonst gibt die Variable die Länge des Files in Blöcken an. Wird diese Zahl beim Schreiben überschritten, so versucht das System, den File zu dehnen, wenn auf der Diskette noch Platz dafür ist.

Fmodified wird „true", wenn ein File verändert wurde, damit beim Schließen des Files dieser das aktuelle Datum zugewiesen bekommt.

Fheader enthält eine Kopie des Directory-Eintrags des geöffneten Files.

Fsoftbuf ist „true", wenn der File einen Puffer hat, der vom System benutzt werden darf, um die einzelnen Records des Files nach und nach von Diskette zu laden oder auf Diskette zu schreiben. Typlose Files haben keinen Puffer; er muß vom Programmierer zur Verfügung gestellt werden. Auch die vordefinierten Files „input", „output" und „keyboard" werden vom Pascalsystem ohne Puffer zur Verfügung gestellt, obwohl es Interaktivfiles sind, weil sie nur auf „ungeblockte" Geräte zugreifen (Bildschirm, Tastatur, usw.). Eine Sequenz wie

close (output);
rewrite (output, '#4:AUSGABE.TEXT');
writeln (output, ...);

sollte in einem Programm vermieden werden, da sonst der Heap durcheinandergebracht wird („input", „output" und „keyboard" liegen knapp unterhalb des Heaps).

Fnxtbyte ist das nächste Byte, das im Puffer gelesen oder geschrieben werden muß.

Fmaxbyte ist das letze Byte im letzten Block, das eine gültige Information besitzt.

Fbufchngd wird „true", wenn der Puffer durch einen Schreibbefehl geändert wurde, was das System dazu veranlaßt, diesen Puffer auf Diskette zu retten, wenn der File geschlossen wird oder der Puffer durch ein Get oder Put neu gelesen oder geschrieben wird.

Fbuffer ist der Puffer, der Platz für einen Diskettenblock zur Verfügung stellt. Dies ist nötig, da immer nur ein bestimmter Block auf Diskette adressiert werden kann. Einzelne betimmte Bytes können nicht gelesen oder geschrieben werden.

Tabelle 2: Unitnummern (Funit) und Namen (Fvid)

Nummer	Name	Bedeutung
1	CONSOLE:	Tastatur und Bildschirm (Mit Eingabe-Echo)
2	SYSTERM:	Bildschirm und Tastatur (Ohne Eingabe-Echo)
3	GRAPHIC:	Graphic Terminal wird nicht auf dem Apple benutzt
4	Disk	Laufwerk 1 (Slot 6, Drive 1)
5	Disk	Laufwerk 2 (Slot 6, Drive 2)
6	PRINTER:	Drucker
7	REMIN:	Externe Eingabe (Terminal, Modem, etc.)
8	REMOUT:	Externe Ausgabe (Terminal, Modem, etc.)
9	Disk	Laufwerk 5 (Slot 4, Drive 1)
10	Disk	Laufwerk 6 (Slot 4, Drive 2)
11	Disk	Laufwerk 3 (Slot 5, Drive 1)
12	Disk	Laufwerk 4 (Slot 5, Drive 2)

Hinweis zur Sammeldisk

Um nun die einzelnen Informationen eines File-Information-Blocks zu analysieren, konvertieren Sie die Dateien FILETEST-.TEXT und FIBTEST.TEXT von der DOS-3.3-Peeker-Sammeldisk auf Ihre Pascal-Diskette mittels GETDOS, oder geben Sie Listing 1 als Programm FileTest und Listing 2 als Programm FibTest ein. Speichern Sie Listing 1 als
FILETEST.TEXT
und Listing 2 als
FIBTEST.TEXT.
Lassen Sie beide Quelltexte compilieren und führen Sie zuerst FILETEST.CODE aus. Das Programm erzeugt die Files
TEXTFILE.TEXT,
INTFILE.TEXT,
DATAFILE.DATA und
CODEFILE.CODE.
Nun starten Sie FIBTEST.CODE, das die Files einliest und die Inhalte der File-Information-Blocks ausgibt.

Finit

Nachdem wir nun verstanden haben, wie die Files im Speicher aufgebaut sind, können wir uns noch einmal der Prozedur Finit zuwenden und uns ansehen, welche Komponenten eines Files initialisiert werden. Für jeden File, der der Prozedur Finit übergebenen wird, werden folgende Werte übergeben:

Fstate wird auf Fjandw gesetzt.
Fisopen wird auf „false" gesetzt.
Feof und Feoln werden auf „true" gesetzt.
Fwindow wird auf die Adresse der Fenstervariablen gesetzt (normalerweise Anfang der Filevariablen + 300 Words, bei typlosen Files „nil").

Handelt es sich um einen Interaktiv- oder Textfile, so wird das höherwertige Byte der Fenstervariablen, die eine Char-Variable ist, auf Null gesetzt, damit Vergleiche mit der Fenstervariablen ordnungsgemäß funktionieren. Des weiteren wird Frecsize auf 1 und bei Interaktivfiles Fstate auf Fneedchar gesetzt (nach Reset muß Get folgen).
Bei typlosen Files werden, wie schon erwähnt, Fwindow auf „nil" und Frecsize auf Null gesetzt.
Bei allen anderen Files wird Frecsize auf 2 · Frecwords gesetzt, also auf die Anzahl der Bytes, die ein File-Element, wie z.B. die Fenstervariable, belegt.
Wenn man den File-Information-Block kennt, kann man gezielt dessen Werte ändern, um damit bestimmte Effekte zu erzielen. Ein Ändern des Wertes Funit von 1 auf 6 würde bewirken, daß die Ausgabe des Files auf einmal auf den Drucker umgelenkt wird. Außerdem kann mit Leichtigkeit festgestellt werden, ob ein File auf Diskette oder als Gerät geöffnet wurde (Fisblkd). Im Zuge weiterer Maßnahmen kann dann an die Stelle eines langsamen
writeln (F, Zeichenkette)
bei geblockten Files ein schnelles
unitwrite (Funit, Zeichenkette [1], length (Zeichenkette))
treten.

Soviel zum internen Aufbau von Files. In Teil 7 soll dann auf den externen Aufbau eingegangen werden.

Kurzhinweise

Die Peeker-Sammeldisk enthält die Programme
FILETEST.TEXT
FIBTEST.TEXT
Nähere Erläuterungen zur Konvertierung und Anwendung finden Sie im Aufsatz unter „Hinweise zur Sammeldisk".

Listing 1: FileTest

```
{$L PRINTER:}
program FileTest;

type Typ = record
             X : integer;
             Y : integer
           end;

var  F1  : text;
     F2  : interactive;
     F3  : file of Typ;
     F4  : file;
     F5  : file;
     I   : integer;
     A   : packed array [0..511] of char;

begin
  rewrite (F1, 'TEXTFILE.TEXT');
  writeln (F1, 'Text');
  close (F1, lock);

  rewrite (F2, 'INTFILE.TEXT');
  writeln (F2, 'Interactive');
  close (F2, lock);

  rewrite (F3, 'DATAFILE.DATA');
  with F3↑ do
  begin
    X := 513;
    Y := 1027
  end;
  put (F3);
  close (F3, lock);

  rewrite (F4, 'CODEFILE.CODE');
  reset (F5, 'FILETEST.CODE');
  for I := 0 to 1 do
    if blockread (F5, A, 1) <> blockwrite (F4, A, 1)
    then close (F4, normal);
  close (F4, lock);
  close (F5, normal)
end.
```

Listing 2: FibTest

```
{$I-}
{$R-}
program FibTest (input, output);

const Maxunit     =    12;
      Maxdir      =    77;
      Vidleng     =     7;
      Tidleng     =    15;
      Fidleng     =    23;
      Fblksize    =   512;
      Fblkmax     =   511;

type  Daterec     = packed record
                      Month : 0..12;
                      Day   : 0..31;
                      Year  : 0..100
                    end; {Daterec}

      Unitnum     = 0..Maxunit;
      Vid         = string [Vidleng];

      Dirrange    = 0..Maxdir;
      Tid         = string [Tidleng];
      Fid         = string [Fidleng];

      Filekind    = (Untypedfile, Xdskfile, Codefile, Textfile,
                     Infofile, Datafile, Graffile, Fotofile,
                     Securedir, Subsvol);

      Direntry    = packed record
                      Dfirstblk : integer;
                      Dlastblk  : integer;
                      case Dfkind : Filekind of
                        Securedir,
                        Untypedfile : (Filler_1  : 0..4095;
                                       Dvid      : Vid;
                                       Deovblk   : integer;
                                       Dnumfiles : Dirrange;
                                       Dloadtime : integer;
                                       Dlastboot : Daterec);
                        Xdskfile,
                        Codefile,
                        Textfile,
                        Infofile,
                        Datafile,
                        Graffile,
                        Fotofile,
                        Subsvol     : (Filler_2  : 0..2047;
                                       Dstatus   : boolean;
                                       Dtid      : Tid;
                                       Dlastbyte : 1..Fblksize;
                                       Daccess   : Daterec)
                    end; {Direntry}

      Dirp        = ↑Directory;
      Directory   = array [Dirrange] of Direntry;

      Windowp     = ↑Window;
      Window      = packed array [0..0] of char;

      Fibp        = ↑Fib;
      Fib         = record
                      Fwindow   : Windowp;
                      Feof,
                      Feoln     : boolean;
                      Fstate    : (Fjandw, Fneedchar, Fgotchar);
                      Frecsize  : integer;
                      case Fisopen : boolean of
                        true : (Fisblkd   : boolean;
                                Funit     : Unitnum;
                                Fvid      : Vid;
                                Freptcnt,
                                Fnxtblk,
                                Fmaxblk   : integer;
                                Fmodified : boolean;
                                Fheader   : Direntry;
                                case Fsoftbuf : boolean of
                                  true : (Fnxtbyte,
                                          Fmaxbyte  : integer;
                                          Fbufchngd : boolean;
                                          Fbuffer   : packed
                                            array [0..Fblksize]
                                            of char))
                    end; {Fib}

      Typ         = record
                      X : integer;
                      Y : integer
                    end; {Typ}

var   F1          : text;
      F2          : interactive;
      F3          : file of Typ;
      F4          : file;
      Fileinfo    : Fib;
      S           : string;

{-------------------------------------------------------------}

procedure writelnBool (B : boolean);

begin {writelnBool}
  if B
  then write ('true')
  else write ('false');
  writeln
end; {writelnBool}

{-------------------------------------------------------------}

procedure writelnFstate (State : integer);

begin {writelnFstate}
  write ('Fstate      : ');
  case State of
    0 : write ('Fjandw');
    1 : write ('Fneedchar');
    2 : write ('Fgotchar')
  end; {case}
  writeln
end; {writelnFstate}

{-------------------------------------------------------------}

procedure writeFheader (Header : Direntry);

{-------------------------------------------------------------}

  procedure writelnKind (Kind : Filekind);

  begin {writelnKind}
    write ('Filekind    : ');
    case Kind of
      Untypedfile : write ('Untypedfile');
      Xdskfile    : write ('Xdskfile');
```

```
        Codefile    : write ('Codefile');
        Textfile    : write ('Textfile');
        Infofile    : write ('Infofile');
        Datafile    : write ('Datafile');
        Graffile    : write ('Graffile');
        Fotofile    : write ('Fotofile');
        Securedir   : write ('Securedir');
        Subsvol     : write ('Subsvol')
      end; {case}
      writeln
  end; {writelnKind}

{-------------------------------------------------------------------}

  procedure writelnTheDate (Date : Daterec);

  begin {writelnDate}
    with Date do writeln (Day, '-', Month, '-', Year)
  end; {writelnDate}

{-------------------------------------------------------------------}

  procedure writelnDir;

  begin {writelnDir}
    with Header do
    begin
      writeln ('Directory:');
      writeln ('Dvid        : ', Dvid);
      writeln ('Deovblk     : ', Deovblk);
      writeln ('Dnumfiles   : ', Dnumfiles);
      writeln ('Dloadtime   : ', Dloadtime);
      write   ('Dlastboot   : ');
      writelnTheDate (Dlastboot)
    end {with}
  end; {writelnDir}

{-------------------------------------------------------------------}

  procedure writelnFile;

  begin {writelnFile}
    with Header do
    begin
      write   ('Dstatus     : ');
      writelnBool (Dstatus);
      writeln ('Dtid        : ', Dtid);
      writeln ('Dlastbyte   : ', Dlastbyte);
      write   ('Daccess     : ');
      writelnTheDate (Daccess)
    end {with}
  end; {writelnFile}

{-------------------------------------------------------------------}

begin {writeFheader}
  with Header do
  begin
    writeln ('Dfirstblk   : ', Dfirstblk);
    writeln ('Dlastblk    : ', Dlastblk);
    writelnKind (Dfkind);
    case Dfkind of
      Securedir,
      Untypedfile : writelnDir;
      Xdskfile,
      Codefile,
      Textfile,
      Infofile,
      Datafile,
      Graffile,
      Fotofile,
      Subsvol     : writelnFile
    end {case}
  end {with}
end; {writeFheader}

{-------------------------------------------------------------------}

procedure Lookfile (F : Fib);

var I     : integer;
    C     : char;
    Ready : boolean;

begin {Lookfile}
  with F do
  begin
    writeln ('Fwindow     : ', ord (Fwindow));
    write   ('Feof        : ');
    writelnBool (Feof);
    write   ('Feoln       : ');
    writelnBool (Feoln);
    writelnFstate (ord (Fstate));
    writeln ('Frecsize    : ', Frecsize);
    write   ('Fisopen     : ');
    writelnBool (Fisopen);
    if Fisopen then
    begin
      write   ('Fisblkd     : ');
      writelnBool (Fisblkd);
      writeln ('Fvid        : ', Fvid);
      writeln ('Freptcnt    : ', Freptcnt);
      writeln ('Fnxtblk     : ', Fnxtblk);
      writeln ('Fmaxblk     : ', Fmaxblk);
      write   ('Fmodified   : ');
      writelnBool (Fmodified);
      writeln ('Fheader:');
      writeFheader (Fheader);
      write   ('Fsoftbuf    : ');
      writelnBool (Fsoftbuf);
      if Fsoftbuf then
      begin
        writeln ('Fnxtbyte    : ', Fnxtbyte);
        writeln ('Fmaxbyte    : ', Fmaxbyte);
        write   ('Fbufchngd   : ');
        writelnBool (Fbufchngd);
        writeln ('Fbuffer:');
        I := 0;
        with Fheader do
        while (I <= Fblkmax) and not Ready do
        begin
          if I mod 16 = 0 then writeln;
          C := Fbuffer [I];
          Ready := (Dfkind = Textfile) and (C = chr (0));
          if Dfkind = Textfile
          then write (C : 4)
          else write (ord (C) : 4);
          I := I + 1
        end; {while}
        writeln
      end {if}
    end; {if}
    writeln
  end {with}
end; {Lookfile}

{-------------------------------------------------------------------}

begin {FibTest}
  reset (F1, 'TEXTFILE.TEXT');
  moveleft (F1, Fileinfo, size_of (Fileinfo));
  writeln ('TEXTFILE.TEXT');
  Lookfile (Fileinfo);
  close (F1);

  reset (F2, 'INTFILE.TEXT');
  read (F2, S);
  moveleft (F2, Fileinfo, size_of (Fileinfo));
  writeln ('INTFILE.TEXT');
  Lookfile (Fileinfo);
  close (F2);

  reset (F3, 'DATAFILE.DATA');
  moveleft (F3, Fileinfo, size_of (Fileinfo));
  writeln ('DATAFILE.DATA');
  Lookfile (Fileinfo);
  close (F3);

  reset (F4, 'CODEFILE.CODE');
  moveleft (F4, Fileinfo, size_of (Fileinfo));
  writeln ('CODEFILE.CODE');
  Lookfile (Fileinfo);
  close (F4)
end {FibTest}.
```

Weitere Wordstar-Modifikationen

Patches für Epson FX-80 mit DDT

von Dieter Conrad

Die Beiträge von H. A. Rohrbacher zum Thema Wordstar und FX-80 (Peeker 7/85, S. 57 und 8/85, S. 45) zeigten auch dem unbedarften Wordstar-Anwender, wie er sein WS-Systen nach eigenen Vorstellungen patchen kann. Dabei wurde vorausgesetzt, daß der Anwender das entsprechende Programm INSTALL.COM besitzt. Wer nicht über INSTALL.COM verfügt, kann den Patch trotzdem durchführen. Benötigt wird dann das CP/M-Programm DDT.COM. Es ist aber auch für den INSTALL-Besitzer sinnvoll, der sein System mit Einzel-Patches modifizieren will, da er sonst den gesamten Install-Vorgang immer wieder neu durchführen muß.
Tabelle 1 zeigt die erweiterten Patches, die sich bewährt haben. (Apple-Anwender, die auf der 80-Zeichenkarte keinen zweiten Zeichensatz installiert haben, ignorieren TRMINI: 0292H bzw. setzen diesen Bereich auf Null.)

Die Ctrl-Codes für die Druckersteuerung werden mit ↑ Px aufgerufen (der Hochpfeil „↑" steht für Ctrl, so daß z. B. ↑ C Ctrl-C entspricht).

Einige Adressen im einzelnen

– ↑ **Tn ROLUP:** erhält ein Suffix 1 für hochstellen „ein" und 0 für tiefstellen „ein". ↑ Tn und ↑ T sind immer paarweise anzuwenden.
– Mit **DEFDSK:** = 02/03 können alle „.OVR"-Dateien auf das Laufwerk B: bzw. C: ausgelagert werden. Das Laufwerk A: verfügt danach über 64K (oder mehr) Speicherkapazität zusätzlich. Wordstar ruft dann automatisch das Laufwerk B: oder C: auf, wenn die Dateien auf dem Laufwerk A: nicht vorhanden sind.
– **SCRLSZ:** legt den maximalen horizontalen Screen Scroll fest.
– **PSINIT:** der Drucker wird in Grundstellung gebracht, der deutsche Zeichensatz angewählt und Doppeldruck initialisiert.
– Die schon erfahrenen Wordstar-Anwender können mit den Adressen **ITHELP:** und **NITHLF:** den Bildschirm von den Hilfemeldungen befreien und so den gesamten Schirm zur Textverarbeitung nutzen.
– Mit **ITITOG:** und **INITWF:** kann nach eigenen Bedürfnissen, nach der Initialisierung, auf Einfügen „ein" und Blocksatz „ein" verzichtet werden.
– **NONDOC:** ist für System-Programmierer geeignet, die WS im N-Modus direkt mit einem Quellcode oder mit einer neuen Programm-Datei aufrufen wollen.

Vorgehensweise beim Patchen

Vor dem Patchen kopiert man auf eine neuformatierte Diskette zuerst das CP/M-Betriebssystem und dann eine WS.COM-Datei. Danach kann DDT.COM gestartet werden.
Zwei-Drive-Besitzer legen hierzu die CP/M-Master-Diskette mit FORMAT.COM, DDT.COM bzw. WS.COM in Laufwerk A: und eine unbenutzte Diskette in Laufwerk B:. Beim Wechsel von Master- und WS-Diskette muß diese ggf. durch ↑ C angemeldet werden. Die Befehle lauten wie folgt:

```
FORMAT
(Frage nach Laufwerk mit B: beantworten)
COPY B:=A:/S
PIP B:=A:WS.COM
DDT B:WS.COM
```

Wer nur ein Laufwerk besitzt, muß die entsprechenden Befehle benutzen und außerdem auch DDT.COM auf die neue Diskette übertragen:

```
FORMAT
(Frage nach Laufwerk mit A: beantworten)
COPY A:=A:/S
FILECOPY WS.COM
FILECOPY DDT.COM
DDT WS.COM
```

DDT meldet sich, wenn die Wordstar-Datei hochgeladen ist, mit dem nachfolgenden Eintrag:

```
DDT VERS 2.2
NEXT  PC
4280  0100
```

Wird eine bereits gepatchte Wordstar-Version benutzt, kann die Adresse unter NEXT auch 4300 ausweisen.

DDT gibt ein „-“ als Prompt-Zeichen aus. Der Patch wird dann wie folgt eingegeben (hier am Beispiele von TRMINI:, das auf Null gesetzt wird):

```
-S0292
-0292 04 00
-0293 14 00
-0294 48 00
-0295 1A 00
-0296 33 00
-0297 00 .
-
```

Der oben dargestellte Dialog zeigt die komplette Bildschirmausgabe. Der Befehl „S0292“ weist DDT an, Änderungen ab Adresse 0292H vorzunehmen. DDT gibt dann die Adresse und den darin enthaltenen Wert aus. Die Änderung erfolgt nun durch Eingabe des neuen Wertes, der mit Return abgeschlossen wird. DDT gibt dann die darauffolgende Adresse aus. Um den Patch abzuschließen, antwortet man mit einem „.“ gefolgt von Return. Zum Überschreiben einer anderen Stelle gibt man einfach wieder „S<adr>“ ein, wobei <adr> für die zu ändernde(n) Adresse(n) steht. Ansonsten wird DDT durch Betätigen von ↑C verlassen. Der Speicherinhalt im RAM bleibt dabei erhalten!
Nun wird die neu erstellte Diskette in das Laufwerk A: eingelegt und mit ↑C angemeldet. Danach muß die neue und geänderte WS.COM-RAM-Datei mit

```
SAVE 66 WSP.COM
```

auf die Diskette geschrieben werden (CP/M zeigt unter STAT immer 128-Byte-Blocks, während SAVE 256-Byte-Blocks verwendet). Nachdem die gepatchte Wordstar-Version ausgetestet ist, muß WS.COM gelöscht und WSP.COM in WS.COM umbenannt werden. Wordstar erwartet bei internen Aufrufen seinen eigenen Dateinamen „WS.COM“ auf dem angemeldeten Laufwerk.

Auf der neuen Diskette dürfen die WS.OVR-Dateien natürlich nicht fehlen, es sei denn, die Dateien wurden mit DEFDSK: verlagert.

Das Löschen und Umbenennen geschieht wie folgt:

```
STAT WS.COM $R/W    (Schreibschutz aufheben)
ERA WS.COM          (alte Datei löschen)
REN WS.COM=WSP.COM  (neue Datei umbenennen)
STAT WS.COM $R/O    (Schreibschutz anlegen)
```

Bei „SAVE nn XXX.xxx“ (nn = Anzahl der 256-Byte-Blocks, XXX.xxx = Dateiname) treten bei Unachtsamkeiten Fehlermeldungen auf, die einfach umgangen werden können:

Tabelle 1

↑P	Adresse	Adr. #	Inhalt (Hex)	Bedeutung
	TRMINI:	0292H	04 14 48 1A 33	Ctrl-Z3 deutsche Zeichen
	DEL1:	02CFH	01	Cursor-Blinkzeit an
	DEL2:	02D0H	01	Cursor-Blinkzeit aus
	DEL3:	02D1H	01	Ctrl-delay Help-Menue
	DEL4:	02D2H	01	Ctrl-delay Copyright
	DEL5:	02D3H	01	Ctrl-refresch Screen
	APLFLG:	02D7H	05	Apple-Flag = 05
	DEFDSK:	02DCH	01/02/03	.OVR = A:/B:/C:
	SCRLSZ:	02DDH	17/14	Horizonal Screen Scroll
	ITHELP:	0360H	01/00/02/03	Hilfestufe Bildschirm
	NITHLF:	0361H	FF/00	Hilfestufe Anzeige aus/ein
	ITITOG:	0362H	FF/00	Einfügen ein/aus
	ITDSDR:	0363H	FF/00	Inhaltsverzeichnis ein/aus
	INITPF:	0366H	08	Zeilenabstand in 1/48"
	INITPF:+1	0367H	48 40 02 08	72 Zeilen/Blatt
	INITPF:+5	036BH	00 00 00 08	Null Leerschritte/Kopf
	INITPF:+9	036FH	03 18 00 08	Drei Kopfzeilen frei
	INITPF:+13	0373H	0A 50 00 08	Zehn Leerzeilen/Fuß
	INITPF:+17	0377H	01 08 00 08	Eine Leerzeile Seiten #
	INITPF:+24	037EH	08	Linker Rand = 8 Spalten
	INITLM:	037FH	00	Randbegrenzung links
	INITRM:	0380H	43	68 Spalten/Zeile
	INITWF:+1	0386H	FF/00	Blocksatz ein/aus
	INITWF:+4	0389H	FF/00	Trennhilfe ein/aus
	NONDOC:	0392H	00/FF	Direktaufruf D/N-Mode
↑B	BLDSTR:	0691H	04	Fettdruck 4fach
↑D	DBLSTR:	0692H	02	Doppeldruck 2fach
↑A	PALT:	06B5H	02 1B 0E	Breitschrift ein
↑N	PSTD:	06BAH	01 14	Breitschrift aus
↑Tn	ROLUP:	06BFH	02 1B 53	tief (1)/hoch (0) ein
↑T	ROLDOW:	06C4H	02 1B 54	tief-/hochstellen aus
↑Q	USR1:	06C9H	02 1B 4D	Elite-Schrift ein
↑W	USR2:	06CEH	02 1B 50	Elite-Schrift aus
↑E	USR3:	06D3H	03 1B 70 01	Proportional ein
↑R	USR4:	06D8H	03 1B 70 00	Proportional aus
↑Y	RIBBON:	06DDH	02 1B 0F	Schmal-Schrift ein
↑Y	RIBOFF:	06E2H	01 12	Schmal-Schrift aus
	PSINIT:	06E7H	08 1B 40 1B 52 02 1B 47 07	Drucker Init D
	PSFINI:	06F8H	03 1B 40 07	Drucker Ende-Reset

Patch-Tabelle für WS.COM VERS. 3.0

```
Schrifttest <^B^A>WSP.COM <^N^B>

      Ein Beispiel in Doppeldruck-Matrix
<^B>  Ein Test in Vierfach-Druck      <^B>
<^A>  Breitschrift                    <^N>
      Nun kommt <^T0>  HOCHSTELLEN    <^T>
      und  dann <^T1>  TIEFSTELLEN    <^T>
<^Q>  Ein Versuch mit der ELITE - Schrift <^W>
<^E>  Jetzt noch PROPORTIONAL-Schrift <^R>
<^Y>  Hier ist jetzt noch Schmalschrift eingeschaltet. <^Y>

      Durch den Patch - PSINIT: 1B 52 02
      ist Einfach-Druck nicht moeglich !
```

1. „No Space“: Die Diskette ist voll oder wurde nicht angemeldet.

Abhilfe: Warmstart mit ↑C durchführen oder eine Diskette mit ausreichend freien Sektoren einlegen und ↑C ausführen. Die CP/M-Befehle wie ERA und REN können, ohne die TPA (RAM-Daten-Bereich) zu zerstören, angewendet werden.

2. „Bdos Err On A: File R/O“: Der Dateiname befindet sich schon auf der Diskette und ist schreibgeschützt.

Abhilfe: Neuen Dateinamen wählen oder mit REN die Datei auf der Diskette umbenennen. „STAT XXX.xxx $R/W“ darf nicht benutzt werden, da hierdurch der Speicherinhalt der TPA überschrieben wird.

Es wurde hiermit aufgezeigt, daß Wordstar nach eigenem Ermessen in jeder Hinsicht entsprechend der Hauptanwendung installierbar ist. Für weitere Hintergrundinformationen wird auf die CP/M- und WS-Handbücher sowie auf Peeker 4/85, S. 59 (CP/M für Einsteiger) und die bereits oben genannten Artikel verwiesen.

Meßwertverarbeitung

Praktikumsauswertung mit dem Apple

von Volker Herrmann

Dieses Programm wurde von mir entwikkelt, als ich in der Uni Karlsruhe immer wieder mit Praktika konfrontiert wurde, die es mit sich brachten, daß ganze Meßwertkolonnen ausgewertet werden mußten. Nicht selten gestaltete sich die Behandlung dieser Meßreihen trotz programmierbarem Taschenrechner ausgesprochen umständlich. Ich brauchte ein Programm, das verschiedensten Aufgaben und Anforderungen gerecht werden konnte:

1. Eingabe von Meßwertkolonnen
2. Kurze Übersicht der Namen der Kolonnen
3. Auflisten der Werte (auch Ausdrucken)
4. Mathematische Verknüpfung der Kolonnen
5. Grafische Auftragung
6. Lineare Regression
7. Abspeichern auf und Einlesen von Diskette
8. Korrekturen
9. Statistik

Das im folgenden beschriebene Programm **MESSWERT** erfüllt diese Anforderungen. Wegen des ungewöhnlichen Umfangs und der spezifischen Aufgabenstellung wurde es jedoch nur auf die Peeker-Sammeldiskette aufgenommen. Daher beschränkt sich der vorliegende Beitrag auf die Beschreibung der Optionen, ohne allzusehr auf die Programminterna einzugehen.

1. Programmstart

Auf der Sammeldiskette befinden sich die Files MESSWERT.START, MESSWERT, MESSWERT.TITEL und MW.ZEICHENSATZ. Das Programm wird durch „RUN MESSWERT.START" aufgerufen. Die Anzahl und Länge der Meßwertkolonnen richtet sich nach dem zur Verfügung stehenden Speicherplatz. Aus diesem Grund empfiehlt es sich, bei umfangreicheren Messungen das DOS in die LC zu schieben (siehe DOSMOVER, Peeker 2/85).
Nach dem Start erfolgt zunächst die Aufforderung, das Datum einzugeben. Dieses Datum wird bei allen folgenden Druckerausgaben mit ausgedruckt. Bei dem verwendeten Drucker handelt es sich um einen Epson FX-80 mit grafikfähigem Interface. Für andere Drucker und Interfaces müssen die im Programm verwendeten Steuerzeichen entsprechend verändert werden.
Nach dem Erscheinen des Titels gibt man die Anzahl der Werte der zu bearbeitenden Kolonnen ein. Eine Längenänderung ist während oder nach der Bearbeitung nicht mehr möglich! Anschließend zeigt der Bildschirm das Hauptmenü, aus dem durch Tippen der entsprechenden Taste die einzelnen Optionen aufgerufen werden. In der Kopfzeile sind Anzahl und Länge der möglichen Kolonnen ausgewiesen. Eine positive Beantwortung der in den Optionen z.T. auftretenden Fragen kann in den meisten Fällen mit Return erfolgen.

2. Die Optinen im einzelnen

2.1. Eingabe von Kolonnen

Jede Kolonne wird durch ihre Nummer beschrieben. Darüber hinaus kann jeder Kolonne ein Name zugeordnet werden.
Zur Werteeingabe werden die einzelnen Felder der Reihe nach abgefragt. Dabei sind alle Tasten zugelassen, die aber als „0" interpretiert werden, sollten sie sich von den Zifferntasten unterscheiden (um die „REENTER?"-Ausgabe des Rechners zu unterdrücken).

2.2. Namenliste

Hier erhält man eine Ausgabe der ersten zehn Zeichen aller eingegebenen Kolonnennamen in der Reihenfolge ihrer Nummern.

2.3. Kolonnen-Listing

Maximal lassen sich drei Kolonnen gleichzeitig auflisten. Dieser Punkt kann durch Drücken jeder anderen Taste, außer Return, vorzeitig verlassen werden.
Wünscht man die Ausgabe über den Drukker, liegt die Grenze bei acht Kolonnen. Da Meßwerte immer einer gewissen Ungenauigkeit unterliegen, ist es meistens nicht sinnvoll, alle vom Computer berechneten Stellen auszudrucken. Man kann nun von einer Rundungsroutine Gebrauch machen, indem die Kolonnennummer und die dazugehörige Anzahl der zu rundenden Stellen eingegeben werden. Dazu ein paar Beispiele, jeweils auf 4 Stellen gerundet:
1347635 ergibt 1348000;
1.347635E22 ergibt 1.348E22;
1.347635E-19 ergibt 1.348E-19;
0.00134763 ergibt 0.0013.
Als Überschrift oder erklärender Text ist es schließlich noch möglich, vier Zeilen Text über die Tabelle zu stellen.

2.4. Kolonnenoperationen

Nach Wahl dieser Nummer erscheint ein sieben Punkte umfassendes Menü.

1. Zurück: Rücksprung zum Hauptmenü.
2. Anleitung: Kurzanleitung zur Eingabe einer Befehlsfolge.
3. Konstanten: Liste der vordefinierten Konstanten (z.B. P = Pi, E = Eulersche Zahl e).
4. Steigung: Hier legt man zwei Kolonnen als Abszisse und Ordinate fest und erhält in einer dritten Kolonne die Steigung.
5. Invertieren: In einer Kolonne wird die Reihenfolge der Werte umgekehrt.
6. Ordnen: Für die Sortierroutine muß man ein klein wenig Geduld aufbringen. Hierbei werden Kolonne 0 und die letzte Kolonne als Werteablage benutzt und sollten nicht anderweitig verwendet werden.
7. Andere Operationen: Unter diesem Punkt bietet sich die Möglichkeit, Kolonnen untereinander und mit Zahlen mittels mathematischer Operatoren und Funktionen zu verknüpfen. Diese Verknüpfungsanweisung wird in Form einer Gleichung angegeben. Dabei müssen verschiedene Vorschriften beachtet werden:

a) Kolonnennummern in „<“ und „>“ einbetten.
b) Die Befehlsfolge beginnt mit „<n> =“, wobei „n“ für die zu berechnende Kolonne steht.
c) Zwischen allen Operatoren, Zahlen, eingebetteten Kolonnennummern und Funktion-Tokens steht ein Leerzeichen.
d) Zahlen und Konstanten sind direkt davor mit einem „#“ zu versehen. Das Vorzeichen gehört nicht zur Zahl. Das „E“ des Exponenten und, falls positiv, der Exponent selbst gehören zur Zahl.
e) Die Befehlsfolge endet nicht mit einem Leerzeichen.

Um dieses mächtige Werkzeug noch einmal zu veranschaulichen, noch einige Beispiele:

<1> = #J
J ist eine Laufvariable. Der Kolonne <1> werden also die Werte 1, 2, 3, ... zugeordnet.
<2> = <1>
Kolonne <2> erhält die Werte von Kolonne <1>.
<3> = TAN (#P / <1>) ↑ #4
Kolonne <3> erhält die Funktionswerte der Funktion y = TAN (Pi/x) ↑ 4, wobei die x-Werte der Kolonne <1> entnommen werden.
<4> = RND (#1) ∗ #E - #1.349E - #4
Kolonne <4> wird mit Zufallswerten im Bereich [0, e - (1.349 ∗ 10 ↑ -4)] gefüllt.
<5> = - #4 / <3> + SQR (ABS (COS (#J)))
Kolonne <5> erhält die Funtionswerte der Funktion y = -4/x + SQR (ABS (COS (n))), wobei n von 1 bis zur Länge der Kolonne läuft.
<6> = EXP (<1> - #2.22E3)
Kolonne <6> erhält die Werte y = e ↑ (x - 2.23 ∗ 10 ↑ 3)

2.5. Grafische Auftragung

Wieder werden zwei Kolonnen Abszisse und Ordinate zugewiesen. Legt man Wert auf im Verhältnis gleiche Punktabstände auf den Koordinatenachsen (z.B. beim Zeichnen eines Kreises), wählt man zweckmäßigerweise „Bildschirm proportional“. Wird danach eine beliebige Taste gedrückt, sieht man die hochauflösende Grafikseite 2 und kann das Zeichnen der Kurve beobachten, wobei x = 0 und y = 0 durch gestrichelte Linien wiedergegeben werden. ESC schaltet zwischen Text und Grafik hin und her.

Mit „D“ schaltet man zunächst auf Text um. Anschließend ist es möglich, die Grafik für den Druck zu beschriften. Dafür lassen sich sämtliche Zeichen verwenden (wegen INALL, Peeker 4/85, S. 70). Sollte die Beschriftung falsch sein, wird die Grafik gelöscht, und die Prozedur beginnt wieder mit dem Drücken von „D“. Der eigentliche Ausdruck (siehe **Abb. 1**) beginnt nach Beantwortung der Frage nach der Druckgröße. Wird hier weder „1“ noch „2“ getippt, findet auch kein Ausdrucken statt und es kann weiter beschriftet werden.

2.6. Lineare Regression

Unter diesem Punkt ist es erstmals von größerem Interesse, den Bildschirm auszudrucken (siehe **Abb. 2**). Hat man für zwei Kolonnen die Regression durchgeführt, drückt man „P“ und erhält dann den Ausdruck. Diese Möglichkeit der Druckausgabe bietet sich an vielen Stellen des Programms.
Um „Ausreißer“ abzufangen, können beliebige Wertepaare unberücksichtigt bleiben. Näheres zur Mathematik der linearen Regression entnehme man den entsprechenden Büchern.

2.7. Diskettenoperationen

Speichern: Gefragt wird nach der Anzahl der Kolonnen ab Nummer 1, die man speichern möchte, und nach dem Namen, den diese Tabelle erhalten soll. Die Textfiles erhalten immer das Präfix „TABELLE.“ und werden gesperrt (LOCK).
Laden: Bereits im Speicher stehende Tabellen gehen verloren, da vor dem Laden der CLEAR-Befehl ausgeführt wird. Tabellen lassen sich also nicht mit anderen mischen.

2.8. Korrekturen

Es erscheint folgendes kurze Menü:

1. Zurück: Rücksprung zum Hauptmenü.
2. Austausch von Kolonnen: Zwei Kolonnen können in ihrer Reihenfolge (siehe Namenliste) gegeneinander ausgetauscht werden.
3. Einzelwertkorrektur: Falsch eingegebene Werte lassen sich unter Angabe ihrer Position in der Kolonne verbessern. Es werden maximal 3 Reihen zu je 16 Werten aufgelistet.
4. Kolonnennamen ändern: Als Hilfe wird der alte Name mitangezeigt.

2.9. Statistik

Hier gilt im wesentlichen das gleiche wie bei der linearen Regression, nur daß die Statistik allein für eine Kolonne durchgeführt werden kann. „Für P = 68.3%“ bedeutet: für eine Wahrscheinlichkeit von 68,3% (siehe **Abb. 3**).

Abb. 1: Grafische Auftragung zweier Kolonnen

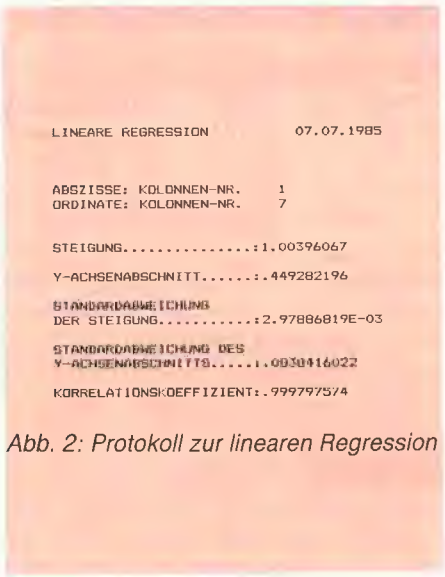


Abb. 2: Protokoll zur linearen Regression

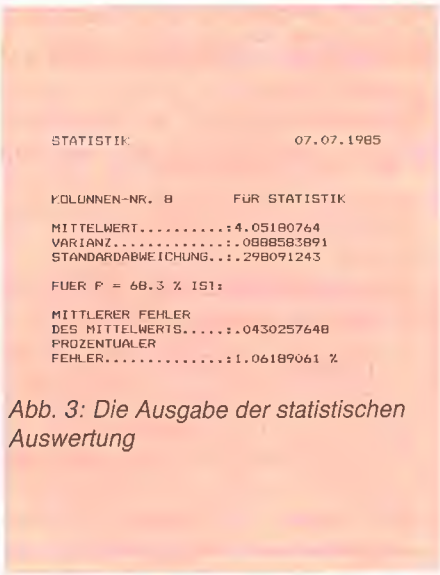


Abb. 3: Die Ausgabe der statistischen Auswertung

2.10. Werte löschen – Neue Eingabe

Wählt man aus dem Hauptmenü diese Option, wird das Programm nach nochmaliger Abfrage, die mit „J“ zu bestätigen ist, neu gestartet.

2.11. Programm verlassen

Mit „E“ aus dem Hauptmenü verläßt man das Programm und befindet sich im Applesoft-BASIC. Ein Drücken der Reset-Taste bewirkt, daß der Rechner einen Kaltstart durchführt, da das DOS gepatcht wurde (es werden nur noch gesperrte Dateien beim Catalog ausgegeben).

3. Fehlerbehandlung

Auftretende Fehler werden mit einer speziellen Routine abgefangen. Eine Übersicht zeigt, welcher Fehler in welcher Zeile auftrat und wieviel freier Speicherplatz noch zur Verfügung steht. Mit jeder beliebigen Taste (Ausdruck mit „P“) gelangt man wieder ins Hauptmenü.

Abschließend noch ein kurzer Hinweis zur Speicheraufteilung: Das Programm belegt den im allgemeinen dafür vorgesehenen Bereich ab $0800 bis $3CDC. Unter der HGR2 liegt die Shape-Table ($3CE6-$3FFA) zur Beschriftung der Grafik (aus Roy E. Myers – Mikrocomputer Grafik, S. 76 ff.). Die HGR2 wird zur grafischen Auftragung benutzt. LOMEM: wurde auf 24580 ($6004) gesetzt, damit Grafik und Variablen nicht kollidieren.

Kurzhinweise

1. Zweck:
Programm zur Erfassung und Auswertung von Meßwertkolonnen
2. Konfiguration:
II+, IIe oder IIc;
Ggf. Epson-Drucker FX-80 mit Grafik-Interface;
DOS 3.3, ggf. in LC geschoben
3. Test:
RUN MESSWERT.START
4. Sammeldisk:
MESSWERT.START
(Applesoft-Startprogramm)
MESSWERT
(Applesoft-Hauptprogramm)
MESSWERT.TITEL
(Titel-Bidschirm als Binärdatei)
MW.ZEICHENSATZ
(Shape-Tabelle)

High Fidelity Adaptor

getestet von Thomas Bühner

Wenn der Lautsprecher des Apple nur bei Spielen zum Einsatz kommt, wird man mit seiner Klangqualität zufrieden sein. Hat man sich jedoch ein Programm gekauft, das Musik macht oder mit dessen Hilfe man selbst komponieren

kann, spielt man bald mit dem Gedanken, sich einen Synthesizer wie das Mockingboard anzuschaffen. Diese Erweiterungskarten können meist an die Stereo-Anlage angeschlossen werden und erzeugen so einen Klang, der dem des eingebauten Lautsprechers weit überlegen ist. Statt viel Geld für einen Synthesizer auszugeben, kann man mit dem „Happ Hifi Adaptor" den Apple direkt an einen Verstärker anschließen.

Montage

Um den Hifi Adaptor im Apple II+ oder IIe anbringen zu können, muß man das Gehäuse des Computers entfernen. Mit einer Schraube befestigt man den Adaptor an einem der Ventilation-Schlitze, so daß von außen nur der Schraubenkopf und ein kleiner stählerner Kippschalter zu sehen sind. Da beim Apple II+ und IIe der Lautsprecher unterschiedlich plaziert ist und die Adaptor-Kabel recht kurz sind, muß man darauf achten, daß man den Hifi Adaptor an der richtigen Stelle anbringt.

Wenn die Schraube gut angezogen ist, löst man den kleinen Stekker, der die Hauptplatine des Apple mit dem Lautsprecher verbindet, und steckt ihn in die entsprechende Buchse des Adaptors. Dann verbindet man Hauptplatine und Adaptor und schraubt das Gehäuse des Computers wieder zu. Das einzige Werkzeug, das während der Montage gebraucht wird, ist ein Schraubenzieher.

Anschluß

Man verbindet nun die Cinch-Buchse des Hifi Adaptors mit dem Auxiliary-Eingang des Verstärkers oder Receivers der Stereo-Anlage. Wenn man jetzt den Apple eine Melodie spielen läßt, kann an einem Rändelrädchen des Adaptors die Lautstärke eingestellt werden, die zu den anderen angeschlossenen Geräten, wie Plattenspieler und Radio, paßt. Legt man den Kippschalter des Adaptors um, verstummt die Stereo-Anlage und der Apple-Lautsprecher nimmt seine Tätigkeit wieder auf.

Klangqualität

Man bemerkt, daß die Töne wesentlich klarer sind, wenn sie durch die Lautsprecher der Stereo-Anlage erklingen. Unreinheiten, die durch das musikerzeugende Programm hervorgerufen werden, treten aber auch deutlicher hervor. Bei dem Programm „Electric Duet" etwa, das zweistimmig spielen kann, indem ein Trägerton von 10.000 Hertz moduliert wird, ist dieser Trägerton jetzt störend hörbar. Das „Music Construction Set" erzeugt jedoch einwandfreie Tonfolgen.

Ein-Blick

Name:	Happ Hifi Adaptor
Einsatz:	Anschluß des Apple II+ oder IIe an einen Verstärker
Gesamtwertung:	****
Dokumentation:	*** (1 Seite)
Zweckdienlichkeit:	*****
Klangqualität:	****
Optischer Eindruck:	*****
Verarbeitung:	****
Allgemeine Bedienbarkeit:	****
Preis/Leistung-Verhältnis:	*****
Preis:	$ 25,-
Hersteller:	Happ Electronics, Inc., Oshkosh, USA

(Die beste Bewertung entspricht 5 Sternen)

Aus postalischen Gründen dürfen wir bei Produktberichten keine Firmenanschriften mitteilen.

Peeker-Redaktion

Mockingboard C

getestet von Roland Maisch

Das über einschlägige Importeure auch in Deutschland erhältliche Mockingboard C ist ein Interface für den Apple II/II+/IIe, das zur Erzeugung von Sprache und Tönen/Geräuschen dient. Die Handhabung der Karte ist durch die mitgelieferte Software recht einfach; die mit dem Mockingboard erstellte Sprache bzw. Sounds können problemlos in eigene Applesoft-Programme eingebunden werden.

1. Allgemeines

1.1. Systemanforderungen

Apple II/II+/IIe, 48K RAM, 1 Disk-Drive (5,25 Zoll), 2 Lautsprecher (8 Ohm) oder ein externer Verstärker mit Lautsprechern.

1.2. Lieferumfang

– Interface mit 2 Sound- und 2 Speech-Chips, 2 Verstärker mit jeweils 0,5 Watt Leistung und getrennter Lautstärkeregelung;
– ausführliches Manual (75 Seiten) in englischer Sprache;
– Kabel mit RCA-Steckern zum Anschluß der Lautsprecher oder des Verstärkers;
– eine doppelseitig beschriebene Diskette mit Demo-Software, Hilfsprogrammen zur Sound- und Spracherstellung sowie einem „Invader-Spiel" mit eindrucksvollen Soundeffekten.

1.3. Anschluß

Das Mockingboard (MB) kann in Slot 1 bis 7 betrieben werden. Software, die die Sound- und Sprachfähigkeiten des MB unterstützt, nimmt Slot 4 als Standardplatz an. Für den Anschluß an Lautsprecher oder Verstärker muß man sich evtl. noch einen entsprechenden Adapter (z.B. RCA -> DIN) besorgen.

2. Betrieb

2.1. Sprache

Die Basis der Spracherzeugung bilden (bei Mensch und Computer) sogenannte „Phone". Dies sind die kleinsten voneinander zu unterscheidenden Laute, deren Art und Anzahl von Sprache zu Sprache variiert. Die Speech-Chips des MB sind in der Lage, 64 verschiedene Phone der amerikanischen Sprache zu erzeugen, aus denen dann jeweils die Aussprache eines bestimmten Wortes zusammengesetzt wird. Da jedes Phon vom MB in vier verschiedenen Längen erzeugt werden kann, stehen eigentlich 256 unterschiedliche Laute zum Sprechen zur Verfügung.
(*Anm. d. Red.:* Strenggenommen gibt es nur 57 Phone, von denen nicht alle in jeder Sprache vorkommen. Da es mit einfachen technischen Mitteln nicht möglich ist, alle natürlichen Phone zu erzeugen, geht man bei der synthetischen Spracherzeugung dazu über, einige weitere Laute aufzunehmen, um der menschlichen Sprache gerecht zu werden. Dadurch unterscheidet sich auch die phonetische Lautschrift bei einem Sprachprozessor von dem durch die API [Association Phonétique Internationale] erstellten phonetischen Alphabet. Auch führt die Länge [Quantität] im engeren Sinn nicht zur Unterscheidung von Phonen, sondern ist wie die Stärke [Intensität] und Tonhöhe [Intonation] nur Attribut eines Lautes.)

Die Aussprache eines bestimmten Buchstabens kann in verschiedenen Worten durchaus unterschiedlich sein. Beispielsweise wird das „e" in „evil" vollkommen anders ausgesprochen als in „yes" oder „silent". Vor der eigentlichen Spracherzeugung im Speech-Chip muß der Computer daher durch die mitgelieferten Hilfsprogramme die richtigen Phone für den jeweiligen Buchstaben auswählen. Diese Auswahl richtet sich nach den in einer Rule-Table über Software festgelegten Aussprachеregeln (die mitgelieferte Rule-Table enthält ca. 870 verschiedene, nach Buchstaben geordnete Regeln). Wem die Aussprache bestimmter Worte nach dieser Standard-Table nicht zusagt, kann die vorhandenen Regeln problemlos verändern oder neue Regeln hinzufügen. Im Editiermodus werden dann per Tastatur Worte oder Sätze (bis max. 239 Zeichen) eingegeben, die sofort nach den momentan im Sprecher befindlichen Regeln in Sprache umgesetzt werden. Es können beliebig viele verschiedene Rule-Tables erzeugt und zur Benutzung in verschiedenen eigenen Programmen auf Diskette gespeichert werden. Trotz dieser Editiermöglichkeiten bleibt der Aussprache jedoch fast immer ein amerikanischer Akzent erhalten, denn die 64 Phone entstammen – wie bereits erwähnt – der amerikanischen Sprache und können einfach nicht alle natürlichen Laute des Deutschen vertreten.

Den letzten Schliff bekommt die Sprache dann noch durch das Hinzufügen einer Satzmelodie, d.h. durch Heben und Senken der Stimme am Satzende, Pausen vor einem Interpunktionszeichen oder die Betonung einzelner Silben. All diese Effekte sind durch einfaches Einfügen von Markern in den zu sprechenden Text zu steuern bzw. werden vom MB automatisch berücksichtigt.
Schließlich ist es noch möglich, den Stimmtypus und die Sprechweise des MB über vier Parameter zu verändern:

– 12 verschiedene Lautstärken,
– 14 verschiedene Sprechgeschwindigkeiten,
– 26 verschiedene Stimmlagen und
– 254 verschiedene „Frequenzfilter"

ermöglichen z.B. die Simulation der Sprache eines beleibten Opernsängers, eines Kleinkindes oder gar eines Marsmenschen. Trotz der Vielfalt der Möglichkeiten wird man viele Kombinationen der Parameter kaum verwenden, da die resultierenden Stimmen nicht mehr verständlich klingen. In einem Testmodus können diese Parameter so lange geändert werden, bis einem die Aussprache des gewünschten Textes zusagt. Dieser Text (max. 239 Zeichen) kann dann mit allen vorgenommenen Einstellungen auf Diskette gespeichert und so später in eigene Applesoft-Programme eingebunden werden. Will man an den eingestellten Parametern nichts mehr ändern, kann man auch mittels eines weiteren Hilfsprogrammes (TEXT TO SPEECH) einen beliebigen Text über die Tastatur eingeben, der dann direkt als sog. „Composite-Textfile" auf Diskette geschrieben wird. Derartige Files bilden dann den Ausgangspunkt zur Spracherzeugung durch das MB in eigenen Applesoft-Programmen.

2.2. Sound

Mindestens ebenso vielfältig wie die Möglichkeiten der Spracherzeugung sind die Möglichkeiten zur Produktion aller nur denkbaren Geräusche: angefangen mit einfachen, reinen Tönen einer Tonleiter bis hin zu komplexen Geräuschen, wie z.B. Meeresbrandung, Glokkenklang, Helikopter, Dampflokomotive oder Sphärenmusik.
Pro Sound-Chip verfügt das MB über drei Übertragungskanäle. Jeder Kanal kann an- oder ausgeschaltet sein, und über jeden dieser Kanäle kann ein Ton, ein Geräusch oder beides zusammen übertragen werden. Es können also 6 verschiedene Töne/Geräusche gleichzeitig produziert und gespielt werden (3 pro Sound-Chip), oder aber auch zu einem Helikoptergeräusch auf dem linken Lautsprecher ein C-Dur-Akkord im rechten Lautsprecher erklingen. Insgesamt ergeben sich also pro Sound-Chip 64 Kombinationsmöglichkeiten zwischen Tönen und Geräuschen.
Was das MB dann tatsächlich an Sounds produziert, wird durch die menügesteuerte Einstellung von verschiedenen Parametern festgelegt. Diese Parametereinstellungen werden für jeden Sound-Chip getrennt vorgenommen. Man legt z.B. fest, welche der drei Kanäle benutzt werden, bzw. ob ein Ton, ein Geräusch oder beides gleichzeitig erzeugt werden soll.

– Um einen Ton zu erzeugen, legt man zunächst dessen Frequenz und dann dessen Lautstärke fest. Im Anhang des Manuals sind die Frequenzen-Einstellungen für alle Halbtöne aus acht Oktaven vermerkt – 4080 verschiedene Einstellungen sind möglich.
– Die Lautstärke kann aus einer

von 16 festen und einer variablen Einstellung gewählt werden. Diese variable Einstellung ermöglicht die Änderung der Lautstärke nach einer von acht vorgegebenen „Hüllkurven". Der Ton kann z.B. leise anfangen, stetig lauter werden und dann plötzlich wieder auf die Anfangslautstärke abfallen; eine andere Möglichkeit ist das regelmäßige An- und Abschwellen des Tons. – Die Zeit, in der das Programm einer solchen Hüllkurve einmal abläuft, ist ebenfalls variabel. Es gibt hierfür 65535 verschiedene Einstellungen, die z.B. aus dem Sound einer Dampflokomotive einen Kanonenschuß machen!

Die für die reinen Töne beschriebenen Manipulationen können ebenso mit Geräuschen durchgeführt werden. Einem Geräusch kann jedoch nicht eine bestimmte Frequenz zugeordnet werden, da es aus verschiedenen Schwingungen aufgebaut ist. Ein Geräusch läßt sich dafür jedoch in 32 Stufen dehnen bzw. komprimieren, was den Grundcharakter sehr deutlich beeinflußt. Alle für gut befundenen Sounds lassen sich natürlich zur späteren Verwendung in Programmen auf Diskette speichern.

3. Programmieren mit dem Mockingboard

3.1. Sound

Um in einem Applesoft-Programm an einer bestimmten Stelle z.B. eine Glocke klingen zu lassen, müssen zunächst zwei Maschinenprogramme (auf der mitgelieferten Disk) in den Speicher geladen werden. Im Applesoft-Programm selbst liegen die Informationen zur Erzeugung des Glockenklangs in Form von 16 DATA-Statements vor (jeder Sound-Chip braucht zur Sounderzeugung immer exakt die Werte für 16 Parameter). Um den Sound erzeugen zu können, werden die Werte dann nacheinander in einen freien Speicherbereich geschrieben, die Startadresse dieser Daten wird dem Maschinenprogramm mitgeteilt und die Sound-Routine mit einem CALL-Befehl aufgerufen. Die Dauer des Sounds wird durch eine Zählschleife bestimmt, an deren Ende durch einen weiteren CALL die Sound-Routine wieder abgeschaltet wird. So etwas könnte dann wie folgt aussehen:

```
100 DATA 145,0,0,0,0,0,0,62,15,
    0,0,0,0,0,0,0
110 A = 33024
120 FOR X = 0 TO 15 : READ D
130 POKE A + X,D
140 NEXT X
150 POKE 8,0:POKE 9,129:
    REM Adresse von A
160 CALL 32768
170 FOR T = 1 TO 4000
180 NEXT T
190 CALL 36897
200 END
```

Auf diese Weise können auch mehrere Sounds hintereinander produziert werden, bzw. die beiden Sound-Chips mit unterschiedlichen Geräuschen beschickt werden.

3.2. Speech

Die Einbindung von Sprache in ein Applesoft-Programm ist der Sounderzeugung recht ähnlich, jedoch noch einfacher. Aufgrund der komplexeren Daten, die zum Sprechen notwendig sind, werden die zuvor als Composite-Textfiles gespeicherten Worte/Sätze als Datenquelle benutzt. Um diese Daten lesen zu können, muß wiederum ein Maschinenprogramm im Speicher vorhanden sein: der sog. COMPOSITE-DRIVER. Als nächstes lädt man dann den Composite-Textfile in einen freien Speicherbereich, übergibt diesem Driver die Adresse der Daten und ruft dann die Speech-Routine mit einem CALL auf. Ein Abschalten dieser Routine ist nicht notwendig. Beispiel:

```
10 PRINT CHR$(4);
   "BLOAD TESTSATZ,A35072"
20 POKE 249,0:POKE 250,137
30 CALL 27904
40 END
```

Um einem Programm einen „unbegrenzten Wortschatz" zu verschaffen (z.B. um eingetippte Antworten eines Benutzers vom Computer sprechen zu lassen), kann man noch zusätzlich eine Rule-Table mit den erforderlichen Aussprachregeln, sowie die zur Umsetzung der Antworten in Sprache benötigten vier weiteren Maschinenprogramme in den freien Speicherplatz laden (sofern freier Speicherplatz überhaupt noch vorhanden ist !).

Übrigens gibt es inzwischen ca. 50 Programme, die ein vorhandenes Mockingboard unterstützen: MUSIC CONSTRUCTION SET, ZAXXON, ADVENTURE CONSTRUCTION SET, ...
Der Hersteller des MB bietet eine ganze Reihe käuflicher Zusatzprogramme/Programmierhilfen an, die es z.B. ermöglichen sollen, vernünftige deutsche, französische oder englische Sprache zu erzeugen. Leider standen bis dato diese Programme nicht zu einem Test zur Verfügung.

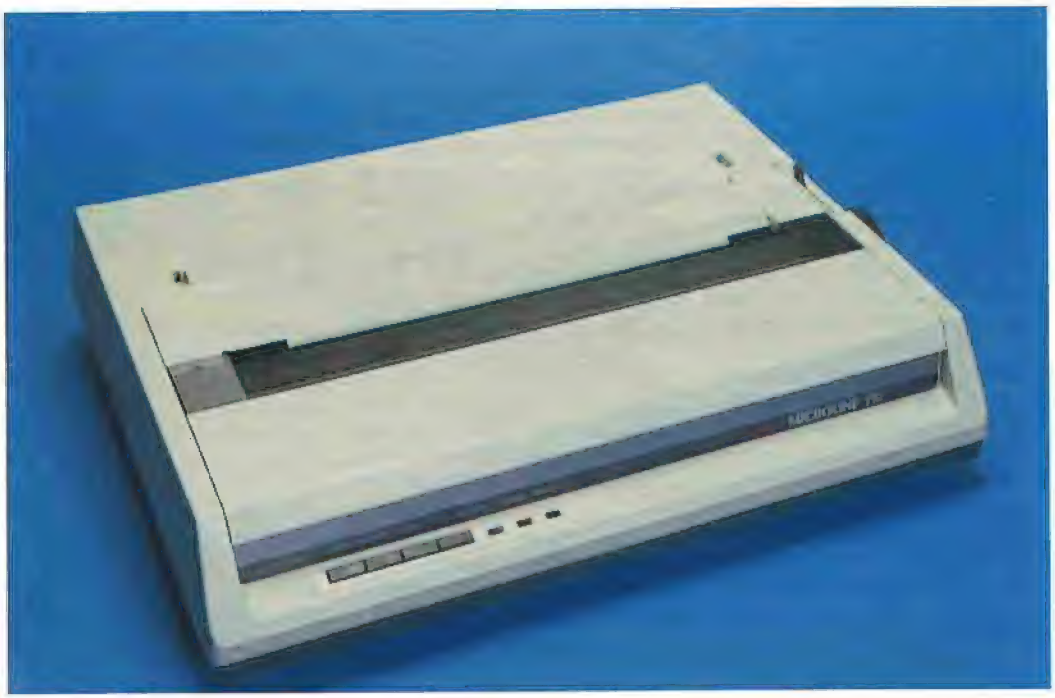
4. Fazit

Mockingboard C ist ein sehr vielfältiges, leicht zu bedienendes „Gerät" zur Sound- und Spracherzeugung. Die kreativen Möglichkeiten bei der Entwicklung immer neuer Geräusche, Töne oder auch Musikstücke sind (fast) unbegrenzt. Die Einbindung in Applesoft-Programme ist an sich leicht, wenn auch aufgrund von (Speicher-)Platzproblemen manchmal unmöglich. Um Musikstücke mit dem MB selbst zu schreiben, ist meines Erachtens zusätzliche Software wie z.B. das MUSIC CONSTRUCTION SET unverzichtbar, wenn man für zehn Takte nicht fünf Stunden am Computer sitzen will.
Ein deutlicher Wermutstropfen ist, so finde ich, auch der nicht gerade niedrige Preis: In Deutschland wechselt das Mockingboard C für ca. DM 660,- den Besitzer. Bei Software-Discountern in den USA ist es etwa für $124,- plus 15-20% Versandgebühren zu erhalten.

Microline 192 von Okidata

getestet von Harald Grumser

Mit dem Microline 192 (der Microline 193 unterscheidet sich nur durch die breitere Bauweise) stellt Okidata einen Imagewriter-kompatiblen Drucker vor, der neben dem günstigeren Preis auch noch andere Vorzüge aufweist.



Das Handbuch

Die Handhabung eines Druckers ist nicht minder anspruchsvoll wie die Bedienung eines größeren Software-Pakets. Daher ist das Handbuch besonders wichtig, da es den effizienten Einsatz entscheidend bestimmt.
Das mitgelieferte Handuch umfaßt knapp 100 Seiten und läßt zunächst etwas an der Professionalität des ganzen Produkts zweifeln. Auch die Tatsache, daß es sich um ein eigens für Apple-Benutzer geschriebenes Handbuch handelt, hätte nicht Anlaß sein müssen, lieblos beschnittene Fotokopien in Leimbindung zu liefern.
Abgesehen davon bietet es jedoch übersichtlich alle benötigten Informationen – sofern man Englisch spricht. In zahlreichen Tabellen und Kästen findet man beim späteren Nachschlagen schnell die gewünschten Steuercodes. Hier hätten nur einige Beispielausdrucke die Verständlichkeit noch erleichtern können (wie z.B. in den Handbüchern zu Epson-Druckern).

Inbetriebnahme

Der Drucker kann nach dem Auspacken schnell in Betrieb genommen werden. Außer dem Anschluß des Interfaces und der Netzverbindung muß nur noch Papier und Farbband eingespannt werden. Das Farbband ist wie bei fast allen Druckerherstellern ein spezielles (Okidata-)Farbband. Der Nachkauf wird somit zwar erschwert, der Einbau ist jedoch in Sekundenschnelle vorgenommen (mancher Star-Druckerbesitzer hätte sich angesichts seiner schwarzen Finger dann doch für diese Lösung entschieden).

Die Anpassung an unterschiedliche serielle Interfaces erfolgt durch Umschalten von leicht zugänglichen DIP-Schaltern. Die Einstellung anderer Parameter, wie Zeichensatz (7 nationale Zeichensätze), Papierende-Erkennung oder automatisches LF nach CR erfolgt durch ein Firmware-Menü (!) des Druckers. Dabei erfolgt die Ausgabe des Menüs auf dem Papier und die Eingabe durch die Druckertasten (Line Feed, Form Feed, TOF Set, Select). Diese Parameter bleiben selbst bei abgeschaltetem Drucker erhalten und ersetzen so die bei anderen Geräten üblichen DIP-Schalter, zu deren Betätigung z.T. der halbe Drukker zerlegt werden muß (z.B. Epson RX-80).

Handhabung

Die Handhabung unterscheidet sich grundsätzlich wenig von anderen Druckern. Es sind die Kleinigkeiten, die den Wert eines solchen Geräts steigern.

Microline 192 von Okidata

Der Microline 192 kennt 6 verschiedene Laufweiten:
9 cpi: Dieser Satz ist nicht immer gleich lang.
10 cpi: Dieser Satz ist nicht immer gleich lang.
12 cpi: Dieser Satz ist nicht immer gleich lang.
13,4 cpi: Dieser Satz ist nicht immer gleich lang.
15 cpi: Dieser Satz ist nicht immer gleich lang.
17,1 cpi: Dieser Satz ist nicht immer gleich lang.

Pica-Schrift läßt sich auch im Proportional-Modus ausgeben.
Genauso wie die Elite-Schrift mit 12 cpi (Chracters per Inch).

Für die Korrespondez schaltet man am besten in den NLQ- und Proportional-Modus:
Die Ausgabe wird zwar langsamer, dafür erhält man aber ein besseres Schriftbild.

Natürlich gibt es die Möglichkeit des
Doppeldrucks mit horizontaler Verschiebung oder des
Doppeldrucks mit vertikaler Verschiebung.
Beides gemischt ergibt dann **halbfetten** Druck.
Das Hochstellen und Tiefstellen läßt sich, ebenso wie das Unterstreichen, in verschiedenen Laufweiten und auch im Proportional-Modus realisieren.

Abb. 1: Schriftmuster

– Der Traktor fällt mit der Friktionswalze zusammen und liegt auf der Höhe des Druckerkopfes. Somit kann sowohl im Einzelblatt- als auch im Lochrand-Endlos-Format jedes Blatt von oben beschrieben werden, ohne ein Blatt als Vorspann opfern zu müssen.
– Ein Schlitz im Boden erlaubt das Einführen des Papiers von unten, was sich bei entsprechendem Druckerständer als praktischer erweist als die Zufuhr von hinten.
– Die Temperatur des Druckerkopfes wird überwacht und die Druckgeschwindigkeit ggf. reduziert, wenn bei kilometerlangen Printouts die Gefahr einer Überhitzung besteht.

Schriftarten

Ein Teil der Schriftarten kann aus **Abb. 1** entnommen werden. Die Überschrift wurde in einem speziellen Modus gedruckt, der alle Zeichen in doppelter Breite ausgibt.
In jedem Fall erwähnenswert ist die Möglichkeit der Schönschrift (NLQ = Near Letter Quality). Wie der Name bereits sagt, kann ein Teil der Korrespondenz durchaus mit diesem Druck-Modus erstellt werden und bietet somit eine Alternative zu den reinen Schönschreibdruckern oder Schreibmaschinen mit Schnittstelle, die für die Ausgabe von Datenkolonnen zu langsam sind und somit als Zweitgerät zugelegt werden müssen.
Die Vermischung der Schriftarten wird, wie oben bereits erwähnt, im Handbuch nicht durch Beispiele erläutert, und es bedarf einiger Versuche, um die vollen Möglichkeiten dieses Druckers kennenzulernen.

Kompatibilität

Der Microline 192 ist weitgehend steuercode-kompatibel zum Apple-Imagewriter. Das Programm Superdump läuft ohne jegliche Anpassung, wenn die Imagewriter-Parameter verwendet werden. Die Realisierung von Indizes (Tiefstellen) und Exponenten (Hochstellen) wurde sogar vereinfacht; ein Steuerzeichen genügt nun. Der von Steuercodes wimmelnde Text in Abb. 1 wird vom Imagewriter (abgesehen von der zweiten Möglichkeit des Doppeldrucks und des Hoch- und Tiefstellens) genauso wiedergegeben.

Geschwindigkeit

Die Angabe der Druckgeschwindigkeit erfolgt oft nach theoretischen Werten. („Wenn die Walze zwei Meter hätte, würde der neue Imagewriter II glatte 250 Zeichen machen.") Wir haben daher die Netto-Geschwindigkeit ermittelt und verglichen. Hierzu wurde ein String mit 79 verschiedenen Zeichen, gefolgt von einem Return, mehrmals gedruckt. Damit wird auch der Papiertransport mitberücksichtigt. Im normalen Betrieb können nochmals schlechtere Werte auftreten, da selbst bei bidirektionalem Druck der Kopf z.T. Leerwege zurücklegen muß. Die gestoppten Zeiten ergaben folgende Werte:

– Microline 192: 93 Zeichen/sec
– Epson FX-80: 97,5 Zeichen/sec
– Imagewriter: 67 Zeichen/sec

Gedruckt wurde jeweils bidirektional in Standardschrift mit Druckerpuffer (im Microline 2K).

Nachteile

Der Traktor erlaubt im Gegensatz zu Geräten der gleichen Preisklasse nur Papier mit Breiten zwischen 9,5 und 10 Inches. Zum Druck von Lochrand-Etiketten muß ein zusätzlicher Traktor-Antrieb erworben werden.
Die Steuercodes bleiben auch nach dem Ausschalten erhalten. Demnach kann nach dem Einschalten nicht von einer definierten Grundeinstellung ausgegangen werden. Hat man abends also das letzte Wort halbfett gedruckt, so sollte man diese Option wieder aufheben, um am nächsten Morgen nicht ein böses Erwachen zu haben.

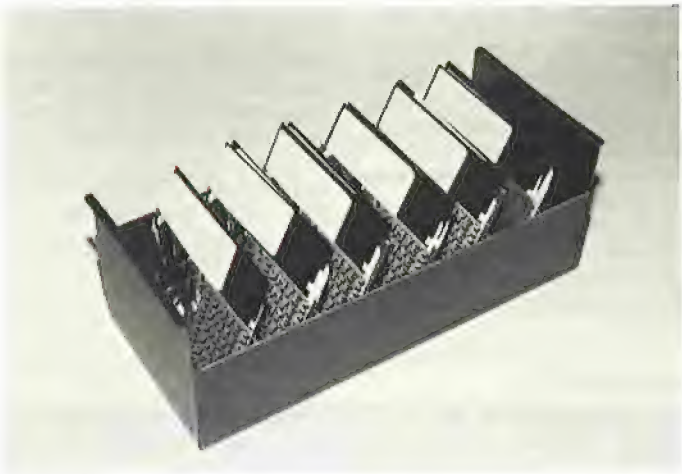
Fazit

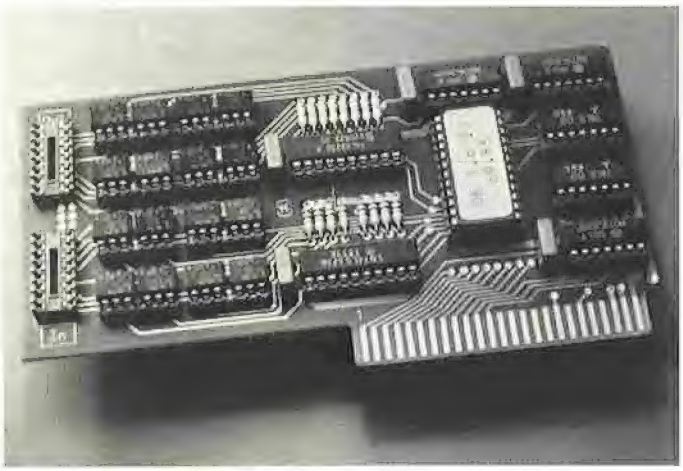
Wer an der Schwelle zum Kauf eines Druckers steht und der Möglichkeit des Schönschreibdrucks hohen Stellenwert einräumt, findet mit dem Microline 192 ein passendes Gerät für sein Anliegen, das darüber hinaus mit einem Großteil an Software läuft, die speziell für den Imagewriter geschrieben wurde. Diese Vorzüge rechtfertigen durchaus den Preis um DM 1800,-, der im Angebot der Billig-Drucker zunächst etwas hoch erscheinen mag.

Diskettenkasten für Hängeregistratur

ComputerLinks GmbH, München hat ein neues praktisches Zubehör im Angebot. Der Diskettensafe ROLLI-DISK paßt in jede Hängeregistratur und ist so konstruiert, daß die 5,25"-Disketten in Schubladen geschoben werden können. Jede Box hat unterteilte Fächer und ist für bis zu 80 Disketten ausgelegt. In eine Hängeregistratur passen zwei ROLLI-DISK. Der Anwender spart sich die platzraubende Diskettenbox auf dem Schreibtisch, die Disketten werden griffbereit und diebstahlsicher, wie z.B. in einem Schreibtischunterschrank, aufbewahrt.

Bezugsquelle: ComputerLinks, München





▲

Optokoppler-Karte

Die Optokoppler-Karte für den Apple II ermöglicht die galvanische Trennung von Computer und Peripherie bei Steuerungsaufgaben. Je nach Ausbaustufe der Karte werden 8 parallele Eingabe- und/oder 8 parallele Ausgabeleitungen zur Verfügung gestellt. Damit eignet sich diese Interface-Karte zur Datenübertragung in Stromschleifentechnik oder für alle Anwendungen, bei denen der Computer vor Störungen durch die angeschlossene Peripherie geschützt werden soll, z.B. bei Steuerung induktiver Lasten. Besonders hervorzuheben ist, daß die Optokoppler-Karte auch mit Software im EPROM geliefert weren kann. Die Software ist als &-Erweiterung des Applesoft-BASIC-Interpreters ausgelegt und deshalb nur in Verbindung mit diesem Betriebssystem verwendbar. Die &-Software stellt dem BASIC-Programmierer leistungsfähige Befehle, z.B. das direkte Ansprechen einer einzelnen Leitung, zur Verfügung. Wenn der Anwender eigene Treiber-Software für die Optokoppler-Karte einsetzen möchte, besteht die Möglichkeit, die &-Software auf Diskette zu kopieren und das EPROM durch ein RAM zu ersetzen. Damit stehen insgesamt 2048 Bytes Speicherbereich zur Verfügung, in denen auch umfangreichere Treiberroutinen bequem untergebracht werden können. Durch den Einsatz eines RAMs kann die &-Software alternativ weiterverwendet werden. Die Optokoppler-Karte ist in verschiedenen Ausbaustufen lieferbar und kostet DM 85,- (nur Ein- oder Ausgabe) oder DM 155,- (Ein- und Ausgabe, &-Software). Eine ausführliche Beschreibung liegt jeder Optokoppler-Karte bei. Interessenten erhalten auf Anfrage eine kostenlose Kurzbeschreibung.

Bezugsquelle: Kurt Stimmler, Tübingen

Typenrad-Drucker Typeterm

Neben dem bereits etablierten Direkt-Interface Typeterm zum Betrieb der Brother-Typenradschreibmaschinen CE-51 bis EM-250 als Ein- und Ausgabemedium am Apple II/IIe bietet die Firma interkom electronic jetzt das Typeterm Junior an. Es handelt sich um das steckfertige Paket aus Typenradschreibmaschine AX-10 und Direkt-Interface zur Ausgabe am Apple II/IIe. Auch hier wird kein zusätzliches Interface benötigt, da es sich um eine Slot-Karte handelt. In Verbindung mit dem Interface wird die AX-10 zum vollwertigen Typenraddrucker und bietet weitergehende Möglichkeiten, wie z.B. 3 verschiedene Schriftstärken, automatisches Unterstreichen, deutscher und vollständiger ASCII-Zeichensatz (mit allen Brother-Typenrädern). Alle Betriebssysteme DOS, ProDOS, Pascal, CP/M werden von der 2K-Software auf der Karte unterstützt. Ein Handbuch gehört zum Lieferumfang.

Bezugsquelle: interkom electronic, Isernhagen

Satzprogramm Pagemaker

Das neue Programm „Pagemaker" von der Aldus Corporation, Seattle/USA, macht den Macintosh auch für das Druckgewerbe zu einer kostensparenden Alternative zu bisherigen Druckpraktiken. „Die elektronische Bearbeitung einer Vielzahl von kommerziellen Text- und Grafikanwendungen und deren Ausdruck in Satzmaschinenqualität läßt sich künftig zu etwa einem Viertel der bisher dazu erforderlichen Investitionen durchführen", erklärte Ralph M. Deja, Geschäftsführer der Apple Computer GmbH, am 26. November 1985 in München. Bisher beschränkten sich diese qualitativ hochwertigen Anwendungen auf Computersysteme der 100.000-Dollar-Klasse. „Pagemaker" ermöglicht professionellen Anwendern ohne EDV-Kenntnisse den elektronischen Umbruch und das grafische Gestalten direkt am Bildschirm eines Macintosh. Pressemappen, Informationsbriefe, Geschäftsberichte, Datenblätter, Preislisten, Handbücher, Bedienungsanleitungen mit technischen Grafiken, Angebotsschreiben, Werbebriefe, Werbeprospekte und -broschüren, Präsentations- und Schulungsunterlagen und viele weitere Geschäftsdokumente lassen sich einfach auf dem Schreibtisch gestalten und „umbrechen". „Pagemaker" läuft auf dem 512K-Macintosh und dem Macintosh XL (Lisa). Benötigt wird ein externes Diskettenlaufwerk bzw. die eingebaute Festplatte beim Macintosh XL. „Pagemaker" arbeitet sowohl mit dem Apple-Matrixdrucker „Imagewriter" als auch mit dem neuen intelligenten Apple-Laserdrucker „Laserwriter". „Pagemaker" kostet in der deutschen Version ca. 2.200,- DM.

Bezugsquelle: Apple Computer GmbH, München

Disketten von Mitsubishi

Die Mitsubishi Corporation, Tokio, startet ab Oktober 1985 europaweit mit dem Vertrieb von Disketten. Die Vertriebsaktivitäten werden von Distributoren übernommen. Für die Bundesrepublik Deutschland werden diese Aktivitäten von microscan GmbH wahrgenommen. Das Produktionsprogramm umfaßt 8-Zoll- 5,25-Zoll- und 3,5-Zoll-Disketten mit unterschiedlichen Speicherdichten. Qualitativ erfüllen die Disketten, die komplett von Mitsubishi Chemical Industries Ltd. (MCI), hergestellt werden, höchste Ansprüche.

Bezugsquelle: microscan GmbH, Hamburg

Synthesizersystem Juice digital

Der Juice digital ist ein vollständiges, polyphones Synthesizersystem mit ungeheuren Möglichkeiten. Das Zeitalter der vielen Knöpfe und Mini-LCD-Anzeigen ist damit vorbei. Dieser Keyboard-Computer „unterhält" sich mit Ihnen über einen großen Bildschirm und eine Schreibmaschinentastatur.

Die Hardware: 9"-Bildschirm, wahlweise ein oder zwei Floppy-Laufwerke, Eingabe-Tastatur und 5-Oktaven-Keyborad – alles in einem Gehäuse. Der empfindlichen Computertechnik wurde durch eine aufwendige und stabile Konstruktion Rechnung getragen. Anschlüsse für einen großen Monitor

und externe Eingabe-Tastatur sind vorhanden. Weitere Peripheriegeräte lassen sich problemlos anschließen. Er besitzt 16 digitale Oszillatoren, die in zwei Gruppen zu je 8 unterteilt sind. Für die Oszillatoren steht je eine programmierbare ADSR und ein programmierbarer Lautstärkesteller sowie ein programmierbarer LFD zur Verfügung. Damit werden Filter unnötig. Wenn die Oszillatoren unterschiedliche ADSR-Programmierung und Obertongehalt aufweisen, ergibt sich durch Summierung und Auslöschung der Oberwellen der gleiche Effekt. Die Tonerzeugung erfolgt durch additive Synthese. Ausgänge für Ihren Stereo-Verstärker sind selbstverständlich. Die Software: 2 Disketten mit ausführlichem Handbuch sind im Lieferumfang enthalten. Diskette 1 enthält das Betriebssystem und die 5 Anwenderprogramme. Auf der 2. Diskette befinden sich massenweise vordefinierte Sounds.

Bezugsquelle: Digitanalog – Ewald Balfer, Krombach

▼

OSP-2-Matrixdrucker

Einen äußerst vielseitigen und leistungsfähigen 12-Nadel-Drucker für PC-Anwendungen stellt jetzt die Firma Data Recording vor. Der Drucker wird höchsten Anforderungen gerecht und bietet u.a. integrierten Einzelblatteinzug (einstellbar bis DIN A4 quer), 7 Farben, Schnell- (200 cps) und Korrespondenzschrift (100 cps), Grafik, steckbare Interface-Karten (seriell RS232C und Centronics parallel) sowie 12 eingebaute Zeichensätze incl. IBM-1- und 2-Zeichensatz. Der Drucker ist Epson-Code-kompatibel und unterstützt zahlreiche Sonderfunktionen, wie z.B. ein programmierbares Interface. Er arbeitet mit den gängigen IBM-Software-Paketen zusammen und kann deren Grafiken auch in Farbe zu Papier bringen. Die zahlreichen Funktionen des Druckers können wahlweise über das Interface oder ein eingebautes Operator-Panel selektiert werden.

Bezugsquelle: Data Recording Instrument GmbH, München

▼

RAMWORKS und Z-RAM

RAMWORKS heißt die Speichererweiterungskarte von Applied Engineering für Apple-IIe-Rechner. Sie wird in den Auxiliary-Slot eingesteckt und ersetzt dabei die 64K-Karte. Darüber hinaus kann sie je nach Aufrüstung mit bis zu 1M als schnelle RAM-Disk unter DOS, ProDOS, CP/M und Apple-Pascal benutzt werden. Das Besondere aber ist die mitgelieferte Software zum Anpassen des deutschen Appleworks 1.2 an die RAMWORKS. Ergebnis ist dann nicht nur ein Arbeitsspeicher von bis zu 758K, sondern auch eine wesentliche Verbesserung von Appleworks. So besteht die Möglichkeit, alle Programm-Module in die Karte zu laden, einen Teil des Speichers als Druckerpuffer zu nutzen und die Timemaster II automatisch beim Start mit Datum und Uhrzeit im deutschen Anzeigeformat einzubinden. Die maximale Record-Anzahl in der Datenbank wird ebenso erhöht wie die maximale Anzahl von Zeilen in der Textverarbeitung.

Die Z-RAM kann bis zu 512K fassen, wird in den APPLE IIc eingebaut und stellt neben einem schnellen Z80B mit 6 MHz, lizenziertem CP/M 4.0B (entspricht CP/M 2.23) und 256K bzw. 512K RAM-Disk die bereits für die RAMWORKS genannten Appleworks-Erweiterungsmöglichkeiten bereit, maximale Erweiterung der „Schreibtischfläche" hier bis 379K.

Bezugsquelle: Weiß Computer, Wilhelmshaven

CSW-EPROM-Karte für Apple II

Bei der CSW-Burnerkarte handelt es sich um ein komfortables und universelles EPROM-Programmier-Gerät. Es besteht aus der eigentlichen Brennerkarte und einer kleinen Extenderkarte mit einem 28poligem Nullkraftsockel. Die beiden Karten sind über ein 40poliges Flachbandkabel miteinander verbunden. Neben dem Nullkraftsokkel sind auf der Extenderkarte noch Leuchtdioden für die Anzeige des Betriebszustandes und der EPROM-Typen (16-128K) untergebracht.

Die Trennung von Brennerkarte und Nullkraftsockel ermöglicht in Verbindung mit der menügeführten Softwaresteuerung aller Funktionen ein bequemes Arbeiten auch bei geschlossenem Rechner. Die folgenden Ausstattungen erhöhen den Bedienungskomfort zusätzlich.

- keine zusätzlichen externen Versorgungsspannungen,
- vier separat anzusteuernde Brennspannungen,
- Anzeige des EPROM-Typs und des Betriebszustandes auf der Slotkarte und der Extenderkarte,
- alle Funktionen werden per Software geschaltet,
- im nicht-aktiven Zustand wird die Burnerkarte abgeschaltet.

Bezugsquelle: Weiß Computer, Wilhelmshaven

Daten-Diagnosesystem

Lange Communications vertreibt jetzt die Erweiterungskarte „Metascope" der Fa. Metatek Inc. für den Apple II, II+ und IIe. Der Apple II wird damit zu einem interaktiven Datenanalysator aufgerüstet. Durch die preiswerte Hardware des Mikrocomputers kann dieser im Vergleich zu den bisher verfügbaren Systemen als sehr preisgünstiges Daten-Diagnosesystem eingesetzt werden. Der Apple II läßt sich damit zur Lösung von Problemen im Bereich der Datenübertragung, zur Analyse von Protokollen und Netzwerken einsetzen. Er kann zur Entwicklung von Protokollen und entsprechenden Schnittstellen für Systeme aller Art verwendet werden. Dabei können z.B. Netzwerke, Front-End-Prozessoren oder Terminal-Steuereinheiten emuliert werden. Die Metascope-Karte ermöglicht sowohl asynchrone als auch synchrone Datenströme bis 19.200 Baud zu analysieren. Die Daten werden dabei in jeweils einer Richtung invers dargestellt. Darstellbar sind die Zeichen in ASCII, EBCDIC, Baudot und Hexadezimal. Auf der Diskettenstation können die Daten gespeichert oder über einen angeschlossenen Drucker ausgedruckt werden. Dies erlaubt z.B. Technikern vor Ort, die auftretenden Probleme aufzuzeichnen und in der Zentrale von entsprechenden Spezialisten analysieren zu lassen. Die Einstellung des Systems ist menügesteuert und daher auch bei weniger häufigem Einsatz leicht zu handhaben. Einmal optimierte Konfigurationen können auf Diskette gespeichert werden und sind so problemlos später wieder einsetzbar. Um spezielle Problemfälle lösen zu können, kann die Aufzeichnung der Daten auch in Abhängigkeit von einer vom Anwender definierten Trigger-Sequenz erfolgen. Zum Beispiel kann die Aufzeichnung beginnen oder enden, nachdem eine bestimmte Zeichenfolge 40mal im Datenstrom erkannt wurde. Darüber hinaus ist das System frei programmierbar, um auch sehr komplexe Problemstellungen zu bearbeiten. So lassen sich Test-Sequenzen senden oder spezielle Endgeräte, wie z.B. Terminal-Steuereinheiten, simulieren. Die Funktionstasten können dabei je nach Bedarf belegt werden, so daß der Anwender das laufende Simulationsprogramm entsprechend der jeweiligen Situation steuern kann. Mehrere Zeitnehmer und Zähler stehen außerdem für die Programmierung zur Verfügung. Die erstellten Programme können anschließend auf einem Diskettenlaufwerk abgelegt werden.

Bezugsquelle: Lange & Co. GmbH, Lippstadt

Preissenkung für Apple-Computer

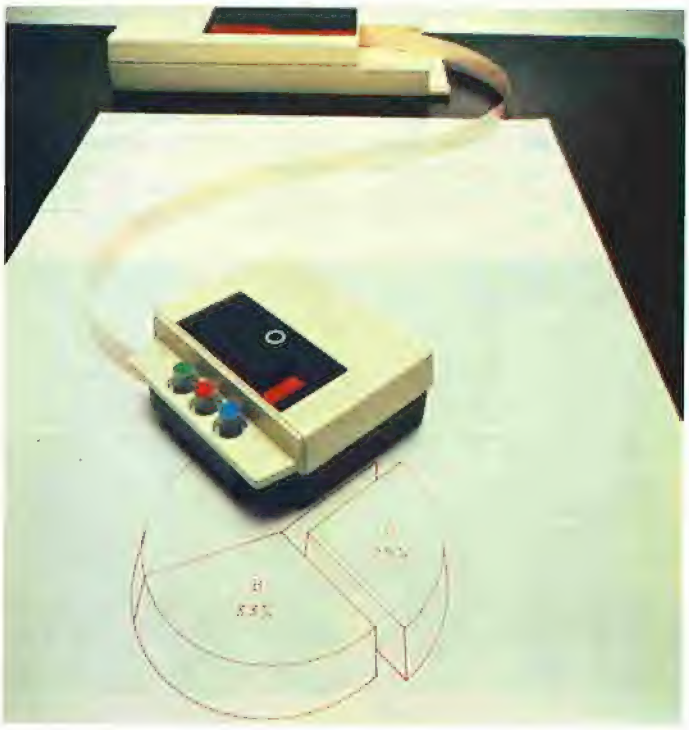
Der Preis des Macintosh 128K wird von DM 8.470,- auf DM 7.980,- gesenkt. Der Preis für den Macintosh 512K verringert sich von DM 10.500,- auf DM 8.949,-. Außer den Preisreduzierungen sind für die Apple-II-Modelle neu im Programm-Angebot ein „Apple-IIe-Ausbau-Paket I" (DM 2.565,-) und „Apple-IIe-Ausbau-Paket II" (DM 2.850,-) sowie ein „Apple-IIc-Startpaket" (DM 3.990,-). Das Apple-IIe-Ausbau-Paket I umfaßt einen Monitor IIe, ein externes Laufwerk, eine Speichererweiterung auf 128K und 80-Zeichen-Darstellung und das integrierte kommerzielle Programm „Appleworks". Das Ausbau-Paket II beinhaltet einen Monitor II, ein Doppel-Laufwerk, eine Speichererweiterung auf 128K und 80-Zeichen-Darstellung sowie die Programme „Applewriter" und „Quickfile". Achtung: Die Ausbaupakete umfassen nicht das Grundgerät (Platine)!
Zum Startpaket gehören ein Apple-IIc-Grundgerät, ein Monitor IIc mit Ständer, eine Tragetasche und das integrierte kommerzielle Programm „Appleworks".

Bezugsquelle: Apple Computer GmbH, München

Roboter-Plotter Penman

Neu im Vertriebsprogramm der Compucon Deutschland GmbH ist ab sofort „Penman", ein vollkommen neu entwickelter und unkonventioneller „Roboter-Plotter". „Penman", mit dem Titel „Peripheriegerät des Jahres 1985" (British Microcomputing Awards) ausgezeichnet, plottet vielfarbig und ist frei programmierbar. Der Roboter-Plotter bewegt sich mit nur zwei Steuerrädern frei auf dem Papier. Der Vorteil: Kreiszeichnungen sind echte Kreiszeichnungen, und keine „Zickzack"-Geraden, unterschiedliche Papiergrößen können benutzt werden. „Penman" kann direkt durch Software-Pakete wie AUTOCARD, MACPLOT und DOODLE gesteuert werden, spezielle Software liegt vor für IBM-PCs und Kompatible, Apple II und Apricot. Geliefert wird auf Wunsch ein „Utility Pack Guide" zur individuellen Programmierung in Microsoft-BASIC, mit Subroutinen, Listings und Driver. „Penman" ist die ideale Ergänzung für jeden CAD-Arbeitsplatz: In der Schule, beim Architekten, in der Konstruktion, als Erst- oder Zweitplotter. Der „Penman" ist im Fachhandel oder direkt bei Compucon zu einem Preis von ca. DM 1596,- erhältlich.

Bezugsquelle: Compucon Deutschland GmbH, Germering



Plotter MP 2000

Die DIN-A3-Plotter der Serie MP – die mit dem Prädikat „Die gute Industrieform" ausgezeichnet wurde – ist durch den Plotter MP 2000 erweitert worden. Der MP 2000 setzt mit seinen Leistungen – die sonst nur Geräte der gehobenen Preisklasse aufweisen – neue Maßstäbe auf dem Gebiet der Low-Cost-Plotter. So besitzt der MP 2000 standardmäßig drei verschiedene, durch Schalter wählbare Befehlssätze: die des MP 1000 (27 Befehle), des FP 5301 (53 Befehle) und eine HP/GL-Emulation. Das Stiftangebot besteht aus Faser- und Keramikstiften, Kugelschreiber und Tuschespitzen für die verschiedenen Zeichenträger. Seine einsetzbaren 8 Federn mit automatischer Stiftspitzenabdekkung arbeiten mit 250 mm/s Plottgeschwindigkeit und einer Schrittgröße von 0,1 mm. Klare Buchstaben durch Verwendung eines Kreis-/Kurven-Generators, Digitalisierfunktion, Vergrößerung/Verkleinerung, Schraffur beliebiger Flächen sowie Fond-Kommandos für Sonderzeichen verschiedener Länder und ein Selbsttestprogramm sind selbstverständlich. Ein 6K-Speicher ist serienmäßig vorhanden. Der Plotter MP 2000 ist in 2 Versionen lieferbar: MP 2000-51 mit Centronics und RS-232-C und als MP 2000-11 mit GP-IB-Interfaces. Wie auf alle GRAPHTEC-Plotter wird eine 24monatige Garantie übernommen.
Bezugsquelle: GRAPHTEC GmbH, Herrsching

8086-Prozessorkarte

Die Prozessorkarte C86 ist ein kompletter 16-Bit-Mikrocomputer mit eigenem Speicherbereich (512K) und Schnittstelle zum Apple-Bus. Die Erweiterung bietet Ihnen:
– einen leistungsfähigen 16-Bit-Mikrocomputer mit Intel-8086-CPU,
– mindestens 128K oder serienmäßig 512K Speicherkapazität,
– ein spezielles Apple-Businterface für den sicheren Betrieb,
– alle notwendigen Teile auf einer Apple-Slotkarte.
Sie ist vorbereitet für CP/M 86 und MS-DOS und als RAM-Floppy unter allen gängigen Betriebssystemen geeignet und äußerst preisgünstig. Es sind fast alle betriebssystem-gestützten IBM-PC-Programme auf der C86 lauffähig, und das bei ca. doppelter Verarbeitungsgeschwindigkeit! Mit dieser Erweiterung entspricht Ihr Apple dem neuesten Stand der Technik.
Bezugsquelle: Anton Peter, Berlin

(Gekürzte Firmenmitteilungen. Bitte rufen Sie die Peeker-Redaktion nicht wegen der Firmenanschriften an, die wir aus postalischen Gründen nicht veröffentlichen dürfen.)

CAD-System SYNCAD

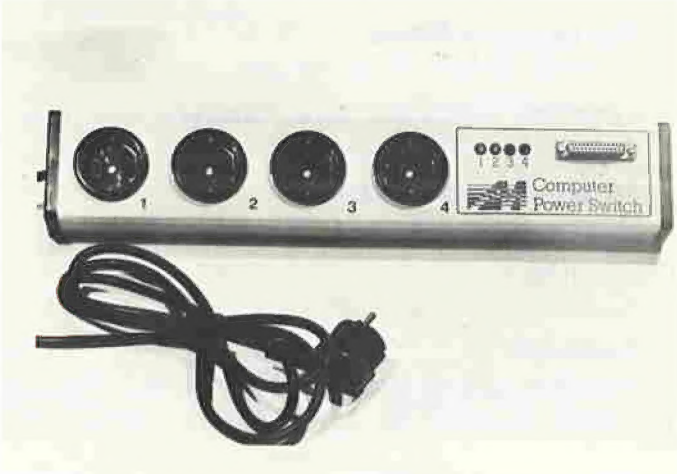
SYNCAD ist ein eigenständiges CAD-System für den rechnergestützten Entwurf von Schalt- bzw. Stromlaufplänen, Bestückungsplänen, Layouts (Interaktiv und Autorouting) und Bauteilbibliotheken. Hinsichtlich der Layout-Erstellung ist das Programm in drei Ausbaustufen lieferbar:

– Interaktive Entflechtung
– Halbautomatische Entflechtung,
– Vollautomatische Entflechtung.

Die einzelnen Stufen können jederzeit nachträglich aufgerüstet werden. Das System ist sehr bedienerfreundlich durch konsequente Anwendung der Maus-Technologie und farbige Darstellung auf dem Grafik-Bildschirm. Der Dialog erfolgt in deutscher Sprache auf einem gesonderten Textbildschirm. Mit dem Programm wird eine Bauteilbibliothek mitgeliefert. Sie kann vom Anwender jederzeit erweitert bzw. geändert werden.
Bezugsquelle: syntax EDV-Beratung und Mikrocomputervertrieb GmbH, Oldenburg

220V-Steckdosenleiste

Eine Vierfach-Steckdosenleiste, die sich über eine Schnittstelle von Computern aus ein- und ausschalten läßt, bietet die Firma MICHAEL-Datentechnik unter der Bezeichnung „Computer Power Switch" an. Die Steckdosenleiste ist entweder mit einer V24-, einer Centronics- oder einer C64-Schnittstelle ausgerüstet und kann von nahezu allen auf dem Markt befindlichen Home- und Personal-Computern angesteuert werden. Die vier Steckdosen lassen sich einzeln problemlos über einfache Drucker-Befehle ein- und ausschalten. Mit dem Computer Power Switch können Haushaltsgeräte, Heizungspumpen, Lichtanlagen, Werkzeugmaschinen etc. mit einer Stromaufnahme bis 8 Ampere betrieben werden.
Bezugsquelle: Michael Datentechnik, Konstanz



Vorschau Heft 4/86

(Änderungen vorbehalten)

Grundlagen

Binäres Rechnen mit Papier und Bleistift
Teil 2: Multiplikation und Division

Technik

Harddisk-Controller
Aufbau und Funktion

Assembler

Assemblierung auf Diskette
Big-Mac-Filer

Applesoft

Haushaltsverwaltung
auf dem Apple IIc/IIe

UCSD

Superschnelle Bildschirmausgabe
in Apple-Pascal

Kyan

Kyan-Pascal und Assembler

CP/M

Premium-Softcard
Speicherverwaltung unter CP/M

Hobby

Kreiszahl Pi auf 15 000 Stellen

Zweistimmige Melodien
Intelligentes Tonprogramm für den Apple

Produkte

Bildschirmdump auf Tastendruck
Fingerprint Plus

Die bessere Maus
Maus für Apple-II-Rechner

Joystick einfacher anschließen
Game-Socket-Extender

68 000-Board DTACK
Erfahrungsbericht

DCODE
Applesoft-Hilfsprogramme

Sidekick
Macintosh Office Manager

Inserentenverzeichnis Peeker 3/86

Erscheinungs- und Anzeigenschlußtermine für Peeker

Ausgabe	Erstverkaufstag	Anzeigenschluß
4	24.03.86	03.03.86
5	21.04.86	01.04.86
6	26.05.86	05.05.86
7	23.06.86	02.06.86
8	21.07.86	01.07.86
9	25.08.86	04.08.86
10	22.09.86	01.09.86
11	20.10.86	01.10.86
12	24.11.86	03.11.86

Computerbücher
die gehen, für Computer die kommen.

Arne Schäpers
ProDOS-Analyse
Versionen 1.0.1, 1.0.2, 1.1.1
1985, 470 S., kart., DM 68,–
ISBN 3-7785-1134-3

Arne Schäpers
Bewegte Apple-Grafik
1985, 305 S., 6 Abb., kart.,
DM 58,—
ISBN 3-7785-1150-5

Frank Bühler
Applesoft Basic
Tips und Tricks
1985, 241 S., 40 Abb., kart.,
DM 38,—
ISBN 3-7785-1094-0

Jürgen Kehrel
Assembler lernen
Band 1: Einführung in die Assembler-Programmierung des 6502
1985, ca. 200 S., kart.,
DM 38,—
ISBN 3-7785-1151-3

Ulrich Stiehl
Apple Assembler
1984, 200 S., 3 Abb., kart.,
DM 34,—
ISBN 3-7785-1047-9

Ulrich Stiehl
Apple DOS 3.3
Tips und Tricks
3., völlig überarbeitete Ausgabe erscheint Anfang 1986

Ulrich Stiehl
ProDOS für Aufsteiger
Band 1
2., geänderte Auflage 1985,
208 S., kart., DM 28,—
ISBN 3-7785-1098-3

Ulrich Stiehl
ProDOS für Aufsteiger
Band 2
1985, 207 S., kart., DM 30,—
ISBN 3-7785-1036-3

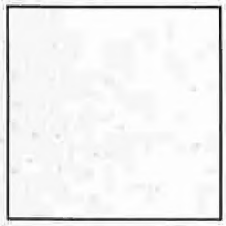
Weitere Titel und Informationen finden Sie in unserem Computerbuch-Katalog:
Dr. Alfred Hüthig Verlag, Postfach 10 28 69, 6900 Heidelberg 1

Hüthig

Ampersand Verlag

Teltower Damm 168
D-1000 Berlin 37
(030) 815 80 69

Apple II Assembler Programmierung
Wagner



Das Assembler-Lehrbuch für BASIC-Kenner!
Roger Wagner, Autor vieler bekannter Software-Pakete, schrieb eine monatliche Kolumne über Assembler-Programmierung in der Zeitschrift SOFTALK. Der vorliegende Band faßt diese Reihe, korrigiert und erweitert, zusammen. Eine stufenweise Einführung in die Befehle und Strukturen der 6502 Assemblersprache, mit vielen Beispielen von der einfachen Tongenerierung bis zum Diskettenzugriff.
277 Seiten.
3-89058-003-3 DM 48,--
komplett mit Disk
3-89058-005-X DM 89,--

Apple II Raster Grafik
Stanton

Die Qualität kommerzieller Arcadespiele läßt sich mit Applesoft BASIC alleine nicht erreichen. Das Buch führt in die Eigenarten der hochauflösenden Apple-Grafik ein und präsentiert schließlich eine Reihe extrem schneller Assembler-Routinen, mit denen Sie viele Effekte bekannter Spiele selbst programmieren können.
Gute BASIC-Kenntnisse werden vorausgesetzt, eine kurze Einführung in Assembler-Programmierung wird gegeben.
299 Seiten.
3-89058-006-8 DM 49,--
komplett mit Diskette:
3-89058-008-4 DM 89,--

Visible Computer

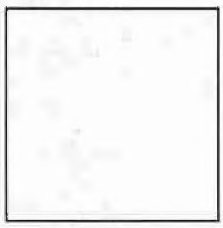
Ein Simulationsprogramm, das Sie in das Innere des 6502-Mikroprozessors führt.
Sie sehen auf dem Bildschirm, wie die einzelnen Instruktionen in Zeitlupe ausgeführt werden, wie sich Register und Flags verändern. Ein unverzichtbares Hilfsmittel beim Erlernen der Assemblerprogrammierung, danach ein wertvolles Werkzeug beim Testen Ihrer Programme.
Komplett mit einem 6502 Editor/Assembler und einem Lehrbuch zur Maschinenprogrammierung (deutsch, 150 Seiten).
3-89058-019-X DM 129,--

MERLIN

Ein professioneller Macro-Assembler für die Apple II-Familie. Neben allen Standard-Features bietet MERLIN u.a.:
- Editor mit globalen Such- und Ersetzfunktionen
- liest und schreibt DOS 3.3 Text und Binärfiles
- Unterstützt 65C02-Opcodes
- enthält einen Disassembler
- kompatibel mit vielen 80-Zeichenkarten und natürlich mit Apple IIe und Apple IIc.

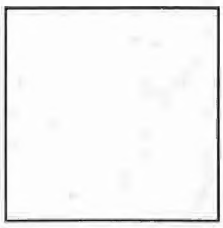
Deutsches Handbuch (170 S.)
3-89058-024-6 DM 198,--

Apple II Schaltpläne
Gayler



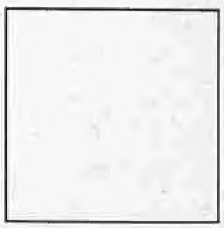
Eine detaillierte Beschreibung der Schaltungen des Apple II und Apple IIplus.
Wenn Sie Ihren Apple selbst reparieren, Interface-Karten oder Schaltungserweiterungen entwerfen oder einfach nur besser über das Innenleben Ihres Apple Bescheid wissen wollen - dieses Buch bietet Ihnen eine Fülle an Informationen, Schaltpläne und Zeitdiagramme, Theorie und praktische Tips.
215 Seiten DIN A4.
3-89058-012-2 DM 64,--

Apple Pascal Trickkiste
DeGroat



Eine Sammlung von Utilities für den Pascal-Programmierer: P-Code-Decodierer, Systemadressen, File-Konverter (von DOS nach Pascal und zurück), DIsk-Zapper, verbesserte Ein/-Ausgabe-Prozeduren, Grafik-Routinen, Textformatierer und viele wichtige Informationen, Tips und Tricks.
248 Seiten.
3-89058-030-0 DM 48,--
komplett mit Disk:
3-89058-032-7 DM 89,--

Mikrocomputer Grafik
Myers



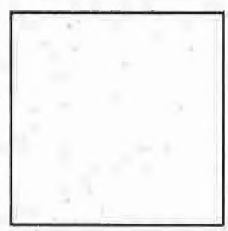
Endlich anspruchsvolle Apple-Grafik Für BASIC-Programmierer. Mikrocomputer Grafik
- enthält fast 80 lauffertige Programme, die die beschriebenen Konzepte illustrieren.
- beschreibt Hidden Line- und Hidden Surface-Techniken, Skalierung, Rotation und Translation von Grafiken.
- bietet eine Einführung in die Animationstechnik.

292 Seiten.
3-89058-000-9 DM 49,--
Komplett mit Diskette:
3-89058-002-5 DM 89,--

Rückgaberecht

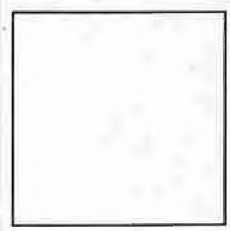
Wenn Ihnen ein Buch wider Erwarten nicht gefällt, dann können Sie es innerhalb von 10 Tagen zurückschicken und bekommen, sofern das Buch unbeschädigt ist, den Kaufpreis erstattet.
Dieses Recht gilt nur bei Direktbestellung beim Verlag und nicht für Software oder Begleitdisketten.

Apple Pascal Grafik
Swan



22 Pascal-Programme, mit denen Sie die Grafikmöglichkeiten Ihres Apple voll ausschöpfen: DESIGNER ermöglicht Ihnen den Entwurf eigener Zeichensätze, mit GREDIT erstellen Sie komplexe Bildschirm-Grafiken, PRINTFOTO bringt Ihre Entwürfe aufs Papier.
Darüberhinaus bietet das Buch Fülle fertiger Prozeduren, die Sie zeitsparend in Ihre eigenen Programme einbauen können.
280 Seiten.
3-89058-009-2 DM 49,--
komplett mit Disk:
3-89058-011-4 DM 89,--

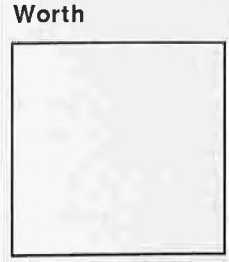
Applesoft Trickkiste
Golding



Alles, was Sie bei Applesoft bisher vermißt haben:
- Eine INPUT-Routine, mit der Sie auch Kommas und Doppelpunkte eingeben können
- Ein PRINT USING Kommando für formatierte Ausgabe
- eine schnelle GARBAGE COLLECTION

und viele andere nützliche Utilities, dazu detaillierte Informationen über die Arbeitsweise des BASIC-Interpreters, Einsprungadressen, Systemvariablen.
304 Seitem.
3-89058-033-5 DM 44,--
komplett mit Disk
3-89058-035-1 DM 89,--

Apple ProDOS Handbuch
Worth



Don Worth und Peter Lechner sind die Autoren von "Beneath Apple DOS", der Bibel aller DOS 3.3-Benutzer. In ihrem neuen Buch haben Sie sich ProDOS vorgenommen und mit der ihnen eigenen Gründlichkeit zerlegt. Ausführliche Beschreibung der Arbeitsweise, des "MLI" und des BASIC.SYSTEM. Das Standardwerk für jeden ProDOS-Anwender.
270 Seiten.
3-89058-036-X DM 46,--
komplett mit Disk
3-89058-038-6 DM 89,--

Bestellcoupon

Ampersand Verlag, Teltower Damm 168, 1000 Berlin 37

Name: .....

Anschrift: .....

.....

O *V-Scheck liegt bei (spesenfreie Lieferung)*
O *per Nachnahme (zzgl. DM 3,-- Versandspesen)*

Anz	Titel oder ISBN-Nummer	Preis

Datum & Unterschrift