Nr. 12/85 Dezember
DM 6.50, sfr 6.50, öS 50, Lit 5900, hfl 7.50

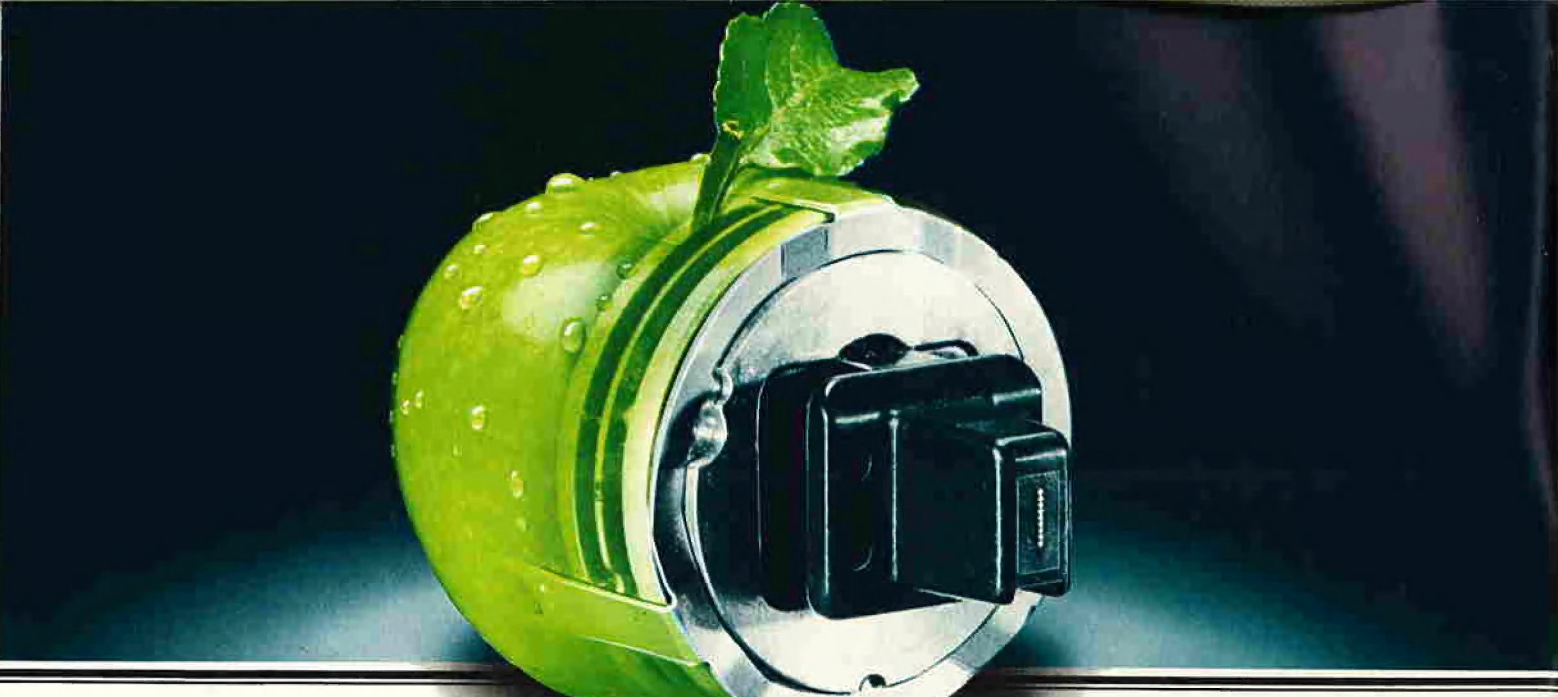
PEEKER

MAGAZIN FÜR APPLE-COMPUTER

Hires-Spiel
Imagewriter
Applesoft intern
Turbo-RAM-Disk
Kyan-Pascal
Mikro-Chronik

Pascal-Kompaktkurs
UCSD- und Turbo-Pascal

Hüthig
PUBLIKATION



NEUERDINGS HAT DER APPEL* EINE BESSERE HÄLFTE

*Düsseldorfer Mundart für einen bekannt hochwertigen Computer

OKI IST O.K.

Rechtzeitig zur Äpfel-und-Nüsse-Zeit eine gute Nachricht für alle Appel-Freunde und die es noch werden wollen: Die Zeit ist reif für einen OKI-Drucker! Denn sowohl den OKIMATE 20 als auch die Hochleistungsdrucker MICROLINE 192/193 gibt es jetzt in einer Appel-kompatiblen Version. Lassen Sie sich von den appetitlich-leuchtenden Farbmöglichkeiten des OKIMATE 20 faszinieren. Natürlich in einer voll Appel-Scribe-kompatiblen Version. Für Grafik, Illustration und Text bis hin zur Schönschrift-Qualität werden Ihrer Phantasie durch den OKIMATE 20 kaum Grenzen gesetzt. Zu einem Preis, der ihn zu einer der begehrtesten Früchte unter den Farbdruckern macht, bietet er über 100 Farbabstufungen, ausgedruckt auf Normal- oder Thermopapier und sogar auf Klarsichtfolie für die Overhead-Projektion.

Mit dem MICROLINE 192/193 bringen wir Sie auf den süßen Geschmack von High-Technologie: Ein ultraslim-Design, neuartige Flat-Pack-Modul-Bauweise, Einzelblatteinzug, hohe Speicherkapazität im Puffer, und das Ganze extrem leise – all dies sind auch für **Ihren** Appel paradiesische Aussichten.

OKTOBER 1985
STIFTUNG WARENTEST
test
MIT TESTERGEBNIS
SEHR GUT

Zusätzlich beim MICROLINE 193: Druckbreite 345 mm, Schallschluckhaube und serienmäßiger Traktor zur Verarbeitung von EDV-Endlos-Papier.
Für Ihren Appel nur das Beste! Natürlich gibt's diese Köstlichkeiten von OKIDATA auch für alle anderen PC's.

Sie wünschen so kurz vor Weihnachten detailliertere Informationen? Dann trennen Sie sich von dem Coupon, oder machen Sie sich schnurstracks auf den Weg zu Ihrem Fachhändler. Er weiß sehr gut, wie man Äpfel von Birnen unterscheidet und wird Ihnen schon deshalb gerne alles Gute über die Drucker von OKIDATA erzählen.

OKIDATA

OKIDATA GmbH · Abt. 17A
Emanuel-Leutze-Straße 8 · 4000 Düsseldorf 11
Telefon 0211-59794-01 · Telex 8587218
Telefax 0211-593345

Coupon
Schicken Sie mir/uns mehr Informationen über

- ☐ MICROLINE 82A/83A
- ☐ MICROLINE 84
- ☐ MICROLINE 92/93
- ☐ PACEMARK 2350/2410
- ☐ OKIMATE 20
- ☐ MICROLINE 182
- ☐ MICROLINE 192
- ☐ MICROLINE 193

Name
Straße
PLZ Wohnort

EDITORIAL

10 Jahre Apple

Man glaubt es kaum, aber vor exakt zehn Jahren wurde der erste Apple konzipiert. Wie Sie in der Mikrocomputer-Chronik in diesem Peeker nachlesen können, begann Steve Wozniak Ende 1975 mit der Konstruktion des später als „Apple I" bezeichneten Bausatzes, der ab März 1976 erhältlich war. Nach der Gründung der Firma Apple im April 1976 ging es im Herbst mit der Entwicklung des „Apple II" weiter, der im April 1977 auf den Markt kam. In Deutschland dürften die ersten Apple-II-Geräte allerdings erst ab Mitte 1978 erhältlich gewesen sein.

Aus Anlaß dieses zehnjährigen Jubiläums suchen wir im deutschsprachigen Raum die ältesten Apple-II-Geräte. Wer besitzt noch einen möglichst funktionsfähigen Apple aus dem Jahre 1978 oder 1979 und schickt uns ein Foto mit einigen Anmerkungen zu technischen Besonderheiten? Wenn Sie nicht sicher sind, ob Sie Besitzer eines der ältesten Geräte sind, so teilen Sie uns bitte telefonisch (Tel. 0 62 21 / 48 93 52) oder per Postkarte das ungefähre Kaufdatum und/oder die Gerätenummer mit. Wir freuen uns auf eine rege Beteiligung.

Kurzumfrage

Nach der Auswertung unserer Kurzumfrage aus dem September-Heft hat sich die prozentuale Geräte-Verteilung kaum verschoben. Lediglich der relative Anteil der Apple-IIc-Besitzer hat merklich zugenommen. Die Zahl der Macintosh-Besitzer ist jedoch immer noch sehr klein. Dies bekam auch die Zeitschrift „Apple's" zu spüren, die dem Macintosh einen sehr großen Raum widmete und nunmehr zum Jahresende ihr periodisches Erscheinen einstellte.

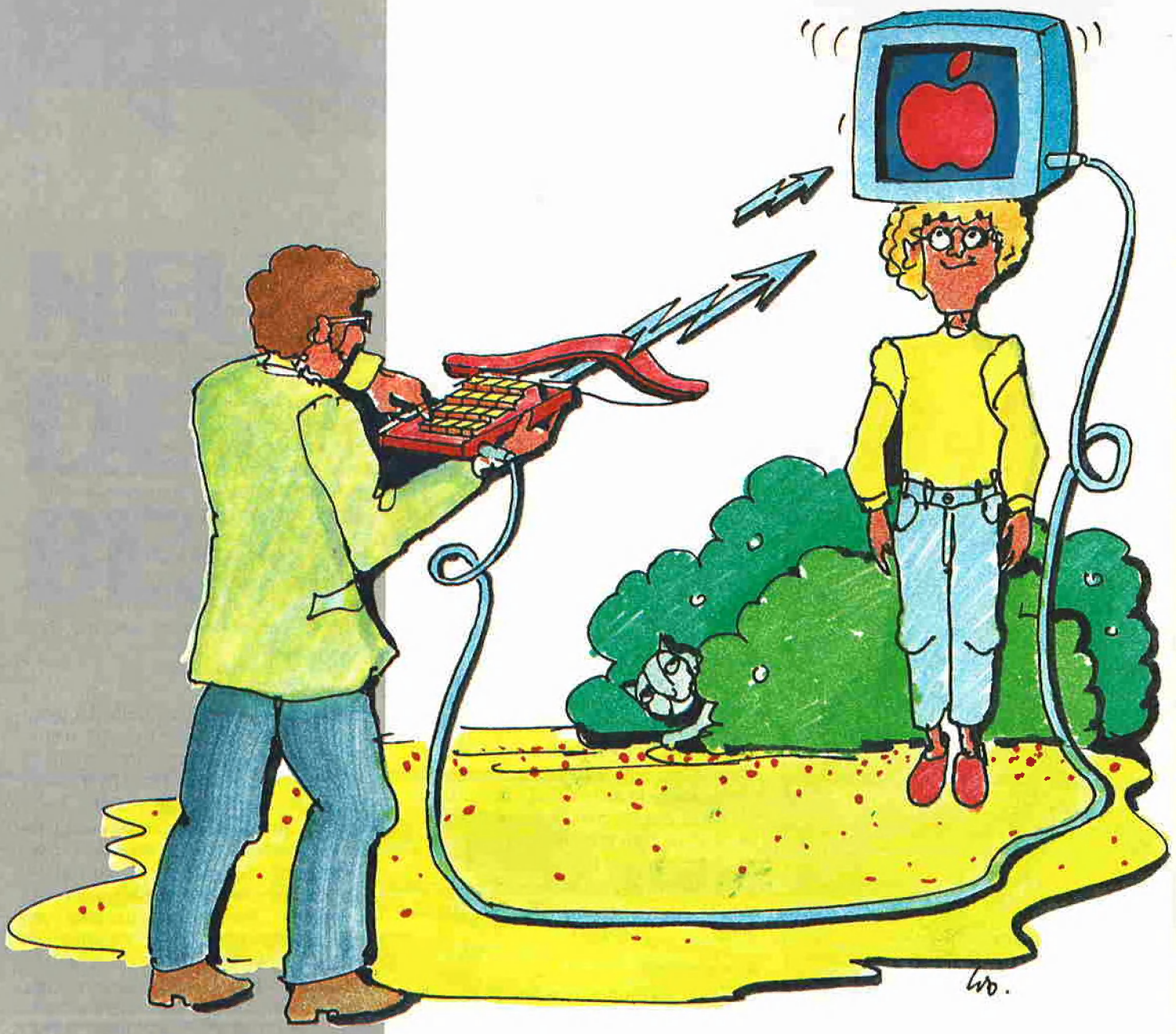
Interessant ist in diesem Zusammenhang auch die „Bewegungsstatistik": Soweit die Kurzumfrage eine Generalisierung zuläßt, scheinen ehemalige Apple-II-Besitzer eher zu einem IBM-PC denn zu einem Macintosh zu tendieren. Der Grund ist evident: Der Apple-II-Markt wird nicht nur in bezug auf die Macintosh-Werbung, sondern auch in bezug auf die Zusatzprodukte zum Apple II selbst völlig ausgespart. Man muß sich allen Ernstes fragen, wie man eigentlich diese Erweiterungen zu verkaufen gedenkt, wenn nicht einmal auf deren Existenz hingewiesen wird. Die Zeiten von 1976, als sich die ersten Freaks noch „selbst geworben" haben, sind mit Gewißheit vorbei.

Neue Rubriken

Auch wenn der Peeker im nächsten Jahr das bewährte redaktionelle Konzept in bezug auf die Fachaufsätze beibehalten wird, werden wir in Zukunft den Anfänger mehr berücksichtigen. Die neue Rubrik „Training" wird Anfängerübungen zu verschiedenen Bereichen bringen, während in der neuen Rubrik „Grundlagen" kompliziertere Grundlagenthemen didaktisch aufbereitet werden. Für das Januar-Heft ist beispielsweise eine sehr anschauliche Darstellung zum Thema „Wie funktioniert Quicksort?" vorgesehen, während im Februar-Heft das „Binäre Rechnen mit Papier und Bleistift" in einer leicht nachvollziehbaren Art behandelt wird.

Ulrich Stiehl

INHALT



Impressum

Peeker
Magazin für Apple-Computer
2. Jahrgang 1985
ISSN 0176–9200

Verleger und Herausgeber:
Dipl.-Kfm. Holger Hüthig
Geschäftsführung Zeitschriften:
Heinz Melcher

Chefredakteur:
Ulrich Stiehl (us) Tel. (0 62 21) 48 92 31
(Bitte nur in redaktionellen Angelegenheiten anrufen)

Anzeigenleitung:
Jürgen Maurer, Tel. (0 62 21) 48 92 18
z. Zt. gilt Anzeigenpreisliste Nr. 3
Vertriebsleitung:
Walter Menzel, Tel. (0 62 21) 48 92 80
Produktionsleitung: Gunter Sokollek
Gestaltung: Rainer Schmitt

Hüthig PUBLIKATION

PEEKER

Heft 12/1985

Verlag:
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.

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

Zahlungen: an den Dr. Alfred Hüthig Verlag GmbH, D-6900 Heidelberg 1: Postscheckkonten: BRD: Karlsruhe 485 45-753; Österreich: Wien 7555 88; Schweiz: Basel 40-24417; Niederlande: Den Haag 1 457 28; Italien: Mailand 47718; Belgien: Brüssel 723 026; 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 0 216 5 041; BLZ 672 700 03; Bezirkssparkasse Heidelberg 204 51, BLZ 672 500 20.

Herstellung: Heidelberger Verlagsanstalt
Printed in Germany

Cosmo-Crumble

Ein Weltraum-Action-Spiel

von Oliver Steinmeier und Alexander Niemeyer

Rechtzeitig zu Weihnachten bringen wir wieder ein Action-Spiel für jung und alt (vgl. auch „Pyramid Pitty" von Peeker 7/85). Das neue Spiel „Cosmo-Crumble" der Autoren O. Steinmeier und A. Niemeyer befindet sich wegen der ungewöhnlichen Länge – wie seinerzeit „Pyramid Pitty" – nur auf der Peeker-Sammeldisk, denn 10 Seiten Hex-Dumps können wir keinem Peeker-Leser zum Abtippen zumuten.
Nach dem Booten von DOS 3.3 (oder besser Diversi-DOS) wird das Action-Spiel mit
RUN COSMO CRUMBLE
gestartet. Die Diskette muß danach noch im Laufwerk bleiben, weil aus speichertechnischen Gründen verschiedene Module nachgeladen werden.

1. Die Spielidee

Sie befinden sich auf einem kosmischen Schlachtfeld und sind in einen Hinterhalt der Cosmo-Crumbles geraten, die Sie auf zwei verschiedene Arten gleichzeitig attackieren:
Die **Bouncer** versuchen in kamikaze-ähnlicher Manier Ihren mit einer Schnellfeuer-Laserkanone ausgerüsteten Raumkreuzer zu zerstören. Glücklicherweise sind sie unbewaffnet und somit nur bei Zusammenstößen gefährlich.
Der **Hurrier** hat sich eine sicherere Position ausgesucht. Er läuft unterhalb des Schlachtfeldes und schießt mit seiner Bordkanone quer über das ganze Feld. Dabei kann es vorkommen, daß der Hurrier einen Bouncer trifft. Die Überreste des Bouncers verglühen langsam und hinterlassen, wenn Sie sie nicht vorher abschießen, ein gefährliches „Loch" im Schlachtfeld, das von nichts durchdrungen werden kann.
Ihre Aufgabe ist es, sämtliche Bouncers zu zerstören. Haben Sie eine Ebene bewältigt, so erwartet Sie sofort eine neue Crumble-Raumflotte in der nächsten Spielebene. Leider sind die Cosmo-Crumbles sehr gute Kämpfer. Es passiert praktisch nie, daß sich einige Bouncers kampflos ergeben und man die nächste Ebene erreicht, ohne sämtliche Gegner zu zerstören.
Ihr Raumgleiter wird durch Treffer des Hurriers, Zuammenstöße mit Bouncers sowie Berührungen mit „Löchern" und verglühten Resten beschädigt, so daß Sie und Ihr Schiff zwecks „Reparatur" von einem kosmischen Krankenwagen abgeholt werden müssen. Nach fünf Aufenthalte im Krankenhaus wird Ihr Raumschiff verschrottet, und die Schlacht ist beendet.
Für jeden abgeschossenen Bouncer erhalten Sie 100 Punkte, für jede Spielebene einen mit dem Schwierigkeitsgrad ansteigenden Punkte-Bonus von mindestens 200 Punkten.
Ein Bonus-Schiff erhält man nach 10.000 Punkten, ebenso nach 20.000, 30.000 usw.
Das Spiel selbst hat 25 Ebenen, die aus unterschiedlichen Formationen mit steigender Schwierigkeit aufgebaut sind. Falls man alle 25 Levels bewältigt, beginnt das Spiel erneut mit Ebene 1, jedoch mit stark gesteigerter Geschwindigkeit.

2. Das Titelbild

Mit der Taste **T** (großes T) können Sie sich die TOP-FIVE-Highscore-Tabelle ansehen. Nach einem Tastendruck kehren Sie zum Titelbild zurück. Wenn Sie die Taste **H** (Help) drücken, erscheint die erste Seite der sogenannten Help-Screens. Diese Informationsseiten informieren Sie über wichtige Tasten und Funktionen des Spiels, um Ihnen das Nachschlagen in diesem Aufsatz zu ersparen, wenn Sie einmal eine Funktion vergessen haben sollten.
Mit der **Leertaste** (Space) können Sie sich die weiteren Seiten anschauen. Nach der letzten Seite kehren Sie automatisch zum Titelbild zurück.

3. Definieren der Steuerungstasten

Wenn Sie Cosmo-Crumble mit der Apple-Tastatur spielen wollen, können Sie vor dem Spiel die Tastenfunktionen selbst belegen. Drücken Sie, wenn das Titelbild auf dem Bildschirm erscheint, die Taste **K**. Nun können Sie die Steuerungstasten für oben, unten, rechts, links, Feuer und Stopp selbst definieren. Nach der letzten Taste kehren Sie zum Titelbild zurück.

4. Spielbeginn

Um das Spiel zu starten, gibt es verschiedene Möglichkeiten. Wenn Sie mit der Tastatur spielen wollen, sollten Sie zunächst die Tastenbelegung wählen. Nach dem Einladen von COSMO CRUMBLE sind folgende Tasten vordefiniert:
F – Feuer,
I – aufwärts,
J – links,
K – rechts,
M – abwärts,
Leertaste – anhalten.
Um das eigentliche Spiel zu starten, wählen Sie nun eine der folgenden Möglichkeiten:
1. **Leertaste**: Das Spiel beginnt, gespielt wird mit der Steuerungseinheit, die zuletzt verwendet wurde (nach Programmstart stets die Tastatur).
2. **Ctrl-J**: Das Spiel beginnt, es wird jedoch automatisch auf Joystick umgestellt.
3. **Ctrl-K**: Nach der Umstellung auf Tastatur (Keyboard) beginnt auch hier das Spiel.

5. Möglichkeiten während des Spiels

Natürlich kann man auch während des Spiels von Joystick- auf Tastatursteuerung und umgekehrt umschalten (wieder mit Ctrl-J und Ctrl-K).
Mit der Taste **ESC** kann das gesamte Spiel angehalten werden. Nach Drücken einer weiteren Taste wird das Spiel fortgesetzt.
Mit **Ctrl-S** kann man die Tonausgabe (Sound) ein- und ausschalten, und mit **Ctrl-R** wird das Spiel beendet.

6. Spiel mit Joystick

Wenn man im Joystick-Modus spielt, kann man mit **Pushbutton 0** einen Schuß abfeuern. Mit **Pushbutton 1** kann wie mit ESC das Spiel unterbrochen werden.

7. Spiel-Ende und Highscore-Tabelle

Nach dem Spiel-Ende erscheint auf dem Bildschirm „GAME OVER". Nach **Return** oder Drücken von Pushbutton 1 wird die Highscore-Tabelle geladen und angezeigt. Die Highscore-Tabelle enthält die fünf bisher besten Ergebnisse und die Namen der Spieler.
Die Eingabe erfolgt stets mit der Tastatur. Bei Eingabefehlern kann mit Hilfe der ESC-Taste die gesamte Eingabe wiederholt werden. Nach dem letzten der 10 Zeichen wird die Eingabe in die Highscore-Liste aufgenommen und auf Diskette abgespeichert (CC.LEVELS).
Deshalb müssen Sie sich das Spiel von der Peeker-Sammeldisk auf eine eigene Diskette kopieren oder den Schreibschutz entfernen.
Nach Tastendruck kehrt man zum Titelbild zurück, und ein neues Spiel kann beginnen.

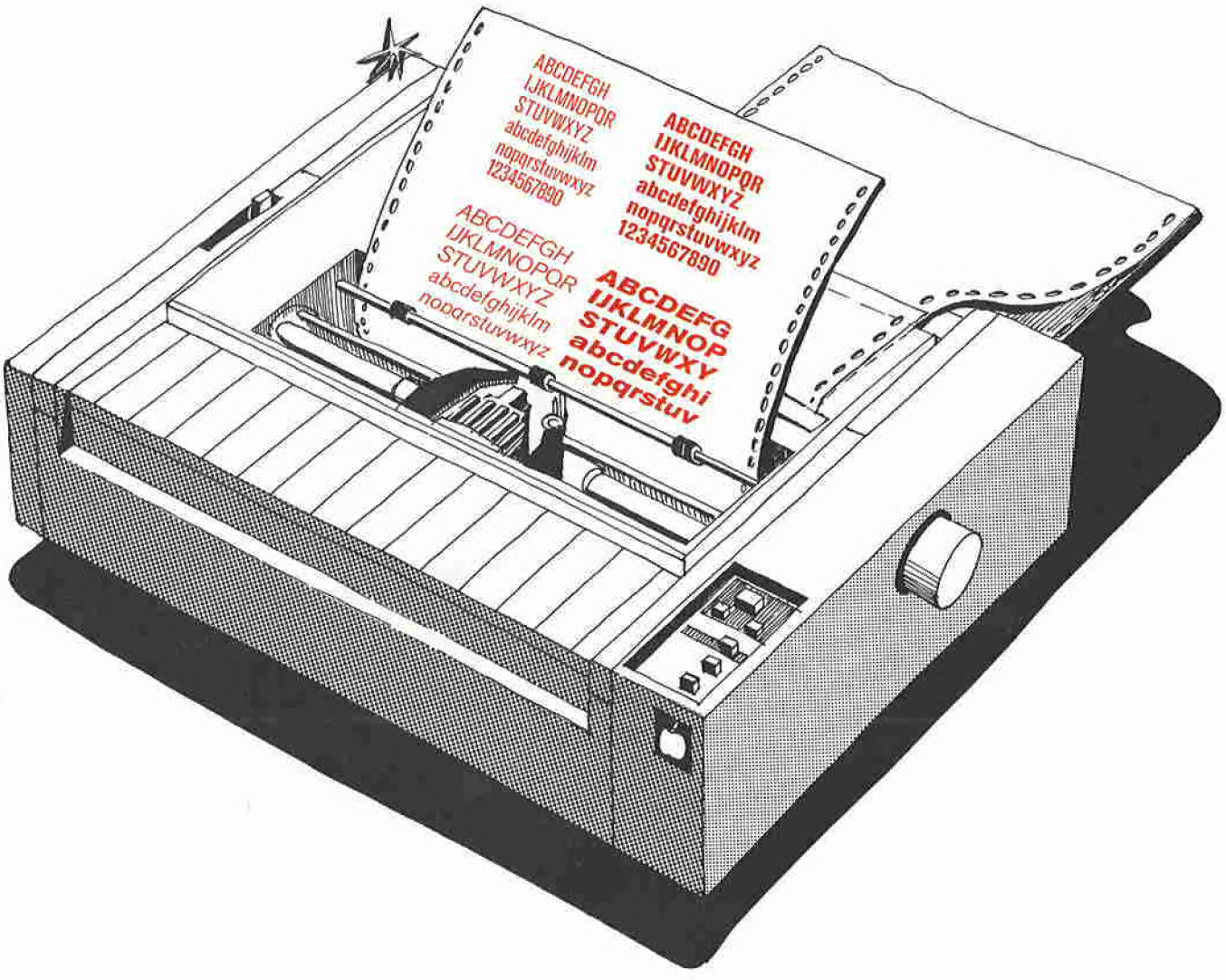
Kurzhinweise

1. Zweck:
Reaktionsspiel für Tastatur oder Joystick.
2. Konfiguration:
Apple II+, IIe oder IIc (wahlweise Joystick); DOS 3.3.
3. Test:
RUN COSMO CRUMBLE
4. Sammeldisk:
COSMO CRUMBLE
(Applesoft-Startprogramm)
T. CC2 – T.CC6
T.CC8
(Big-Mac-Quelltexte)
CC2 – CC8
CC.LEVELS
(Maschinenprogramme und Tabellen)

Imagewriter kurz und bündig

Die wichtigsten Steuerbefehle

von Thorsten Schunk



Auf den folgenden Seiten sollen der Grund-Befehlssatz des Imagewriter-Druckers vorgestellt und Fehler des Handbuches berichtigt werden. Der Artikel beschreibt ein Programm, das es ermöglicht, dem Drucker Kommandos zu übermitteln. Alles hier Aufgeführte gilt wahrscheinlich auch für den Scribe-Drucker, wenn eine Super-Serial-Card verwendet wird. Man sollte sich nicht wundern, daß in diesem Artikel zum Teil etwas völlig anderes als im Handbuch steht, aber leider scheint sich der Verfasser des Handbuches nicht genug mit diesem Drucker beschäftigt zu haben.

1. Anfängerübungen

Wenn Sie nur geringe Vorkenntnisse haben, so sollten Sie zunächst die folgenden Übungen Schritt für Schritt nachvollziehen:
1) Booten Sie mit PR#6 eine beliebige DOS-3.3-Systemdiskette, z.B. die System-Master-Diskette.
2) Nun befinden Sie sich im Applesoft-Modus. Legen Sie nun die Peeker-Sammeldisk #12 ein und starten Sie das Imagewriter-Demo, indem Sie
RUN IW.DEMO
(gefolgt von Return) eingeben.
3) Es erscheint nun ein kurzer Erläuterungstext, den Sie zunächst durch Tippen einer beliebigen Taste ignorieren können. Nun sehen Sie ein Menü mit 3 Optionen:
1: Steuer-Sequenz eingeben
2: Steuer-Sequenz speichern
3: Ende
4) Schalten Sie nun den Imagewriter ein (Power-Lämpchen muß leuchten).
5) Tippen Sie „1" für Steuer-Sequenz eingeben. Es erscheint die Zeile „DRUKKER:". Sie sehen jedoch jetzt keinen blinkenden Cursor, denn jedes Zeichen, das Sie nach „DRUCKER:" tippen, wird als Dezimalzahl und nicht als sichtbares Zeichen angezeigt. Tippen Sie nun erstens auf die ESC-Taste und danach zweitens auf die n-Taste. Nun müßten Sie am Bildschirm die zwei Dezimalzahlen „27, 110" sehen. Wenn dies der Fall ist, so schließen Sie die Eingabe durch gleichzeitiges Drükken auf die Offene Apfeltaste und die Return-Taste ab. Wenn Sie sich vertippt haben, können Sie durch Drücken auf die Del-Taste die jeweils letzte Dezimalzahl löschen.
Nach dem Betätigen von Offener-Apfel-Return „ruckt" der Imagewriter, d.h. er hat die Zeichenfolge „ESC n", die für „weiten Zeichenabstand" steht, empfangen und „sich bis auf Widerruf gemerkt". Gleichzeitig sind Sie wieder im Hauptmenü mit „1 Steuer-Sequenz eingeben" usw. gelandet.
6) Tippen Sie nur wieder „1" und dann einige beliebige „normale" Buchstaben, z.B. „AAAAA". Danach wiederum Offener-Apfel-Return. Merken Sie sich bitte diesen Zyklus. Nachdem Sie „AAAAA" getippt haben, druckt der Imagewriter „AAAAA" in „weitem Zeichenabstand".
7) Geben Sie nun in einem erneuten Zyklus die Steuer-Sequenz „ESC N" ein, dann wieder z.B. „AAAAA", und Sie werden sehen, daß „AAAAA" jetzt im „normalen Pica-Abstand" gedruckt wird.
8) Wiederholen Sie den Zyklus mit „ESC Z Ctrl-G Ctrl-§" und dann mit „ÄÖÜ". Für „ESC" tippen Sie auf die ESC-Taste, für „Z" auf die Z-Taste, für Ctrl-G auf die Ctrl-Taste *und* G-Taste *gleichzeitig*, für Ctrl-§ auf die Ctrl- und §-Taste (Shift-Taste + 3-Taste). Nach „ÄÖÜ" werden Sie statt „ÄÖÜ" die amerikanischen Zeichen „[|]" sehen, d.h. Sie haben mit „ESC Z Ctrl-G Ctrl-§" auf den amerikanischen Zeichensatz umgeschaltet.
9) Wiederholen Sie den Ihnen nunmehr bereits bekannten Zyklus mit „ESC D Ctrl-§ Ctrl-A", gefolgt von „00000". Sie werden sehen, daß die Nullen von „00000" jetzt durchgestrichen sind.
10) Zurück im Hauptmenü, legen Sie jetzt eine eigene Diskette ein und wählen Sie statt „1" die Option „2" für „Steuer-Sequenz speichern". Danach wird das Disketteninhaltsverzeichnis angezeigt, und Sie geben der Datei den Namen „NULLSTRICH".
11) Verlassen Sie jetzt das Programm über „3 Ende", schalten Sie den Drucker aus und wieder ein, und starten Sie die Datei NULLSTRICH mit
BRUN NULLSTRICH.
Schalten Sie den Drucker mit PR#1 ein und tippen Sie
PRINT „10000"
Sie werden sehen, daß die Nullen jetzt durchgestrichen werden.

Was lernen wir aus diesen Übungen? Mit IW.DEMO können Sie beliebige Steuer-Sequenzen für den Imagewriter ausprobieren und als mit BRUN startfähige Initialisierungsprogramme auf der Diskette speichern. Ein solches Programm enthält jedoch immer nur die *zuletzt* eingegebene Steuer-Sequenz. Sie können jedoch insgesamt max. 170 Steuerzeichen hintereinander eingeben, was für alle praktischen Fälle ausreicht. Die Steuerzeichen selbst können Sie der Tabelle entnehmen. Merken Sie sich bitte, daß eine Steuer-Sequenz so lange vom Imagewriter „befolgt" wird, bis Sie einen gegenteiligen Befehl erteilen oder den Drucker ausschalten. Falls fremde (und meist geschützte) Anwenderprogramme keinen „reinen Tisch" machen, bleiben die mit IW.DEMO initialisierten Werte bestehen.

2. Technisches zur Tabelle

Wie im Handbuch erwähnt wird, gibt es zwei Arten von Ctrl-Codes: Steuerzeichen und Steuer-Sequenzen. Letztere sind Zeichenfolgen, die meist mit Escape (ESC) beginnen, wobei die Resetfunktion eine besondere Stellung einnimmt: Sie stellt alle Standardeinstellungen, also den Einschaltzustand, wieder her. Sie sollte nicht in eine Kommandofolge eingebaut werden. Wenn dies aber unvermeidbar scheint, ist ihr ein Nullstring anzufügen, da sonst Schwierigkeiten auftreten könnten. Steuer-Sequenzen, die mit ESC D oder ESC Z beginnen, sind „Schalterfunktionen"; für einige können statt dessen die entsprechenden DIP-Schalter im Imagewriter selbst eingestellt werden.
Der Übersicht und Kürze wegen werden die Kommandos im folgenden stichpunktartig behandelt und in einer Tabelle zusammengefaßt:

Seitenlänge: Seltsamerweise steht selbst im deutschen Handbuch eine Standardeinstellung für 66 Zeilen je Seite, was der amerikanischen Norm entspricht. Um mit DIN-Papier klarzukommen, sollte der Drucker jedoch auf 72 Zeilen je Seite eingestellt werden (SW 1-4).

Zeilenabstand: Der Abstand aufeinanderfolgender Zeilen kann in Schritten von 144stel Zoll eingestellt werden. Zwei Standard-Abstände sind als ESC-Sequenzen verfügbar. Mit dem Zeilenabstand ändert man auch die Blattlänge. Nur für 6 Zeilen/Zoll ergibt sich eine Normlänge.

Zeilenvorschub (LF): Die durch Befehl eingestellte LF-Richtung bleibt zwar bis zum gegenteiligen Kommando bestehen, gilt aber – im Gegensatz zu Änderungen beim Zeilenabstand für FF – nicht für die LF-Taste. Übrigens wird das Kommando für automatischen Zeilenvorschub nur anerkannt, wenn vorher der Drucker mit entsprechendem Code für High-ASCII (8-Bit-Code) vorbereitet wird, sonst passiert nichts. Nachher sollte man nicht vergessen, ihn wieder auf 7 Bits zurückzustellen.

Wagenrücklauf (CR): Das Zeichen für Wagenrücklauf ist normalerweise nicht nur, wie im Handbuch behauptet, Carriage Return (CR), sondern es werden auch LF, VT und FF als Wagenrücklauf behandelt. Schaltet man auf den anderen Modus, so werden LF, VT und FF innerhalb eines Textes ignoriert.

Puffer-Überlauf: Hierbei ist nicht der Zwischenpuffer des Druckers gemeint, sondern der Zeilenpuffer. Man kann damit das Verhalten des Druckers am Anschlag (Zeilenende) einstellen. LF bei Pufferüberlauf bedeutet, daß er einen Zeilenvorschub

ausführt. Anders als im Handbuch beschrieben, tut er dies nicht automatisch. Wählt man einen Zeichenabstand, der weniger Zeichen in einer Zeile erlaubt, als die Druckerkarte zuläßt, so überschreibt der Drucker einen Teil der Zeile.

Nullen mit Schrägstrich: Der im Handbuch angegebene Code schaltet den italienischen oder französischen Zeichensatz ein. In der Tabelle ist das berichtigte Kommando aufgelistet.

Rückschritt (BS): Es ist immer nur ein Rückschritt erlaubt, dem ein druckbares Zeichen, also kein Ctrl-Zeichen, folgen muß.

Da es zu weit führen würde, alle Fehler zu besprechen, sollte man die Kommandos (auch den dezimalen und hexadezimalen Code) im Handbuch vor der Benutzung mit der Tabelle vergleichen, die hoffentlich nunmehr richtig ist.

Eine Übermittelung der Kommandos an den Drucker mit PRINT in Applesoft erfolgt oft fehlerhaft, da der String verschlungene Wege zurücklegen muß. Enthält ein String Ctrl-D, so fühlt sich DOS oder ProDOS angesprochen und „raubt" alle Zeichen bis zum nächsten CR. Die Interface-Karte läßt ebenfalls einige Zeichen nicht passieren. Aus diesen und anderen Gründen ist es ratsam, Befehlssequenzen nicht auf dem üblichen Weg zu senden. Eine Möglichkeit bietet das Programm IW.DEMO, das schon oben in vereinfachter Form erläutert wurde.

3. Technisches zu IW.DEMO

Das Programm enthält drei Unterprogramme, die selbständige kleine Programme sind.

Die **Identifikationsroutine** (Zeilen 63000-63220) ermittelt die Gerätekonfiguration, und damit paßt sich das Programm der Hardware, auf der es gestartet wurde, an. So können alle Besonderheiten des spezifischen Rechners ausgenutzt werden, wobei man allerdings auf dem Apple II oder II+ leider auf einige Optionen verzichten muß. Die erstellten, BRUN-fähigen Steuercode-Dateien sind dadurch ebenfalls hardwarespezifisch, d.h. sie funktionieren meist nur auf der Rechnerkonfiguration, auf der sie erstellt wurden.

Die **Input-Routine** (Zeilen 12000-12210) erlaubt jede Eingabe. Auf neueren Apples sind auch Low- und High-ASCII möglich. Auf dem Bildschirm werden die dezimalen ASCII-Werte angezeigt. Es wurde weniger Wert auf gute Editiermöglichkeiten gelegt als darauf, daß alle Zeichen akzeptiert werden.

Die **Output-Routine** (Zeilen 13000-13240) sendet Kommando-Strings direkt an den Drucker. Natürlich kann man auch Text senden. Ihr Vorteil ist, daß andere Karten eingeschaltet bleiben können. Da normalerweise nach dem String CR und LF gesendet werden, fügt die Routine diese an. Ist das nicht erwünscht, kann der zusätzliche Einsprungpunkt (Zeile 13060) verwendet werden. Außer auf dem Apple IIc wird aus technischen Gründen vor jedem String ein Nullstring übertragen. Stellt man Kommando-Sequenzen auf, ist zu beachten, daß Applesoft beim Addieren von Strings leere Anführungszeichen ("") nicht beachtet. Als Nullstring sollte man CHR$(0) benutzen.

Die Input-Routine ist als T.IW.IN und die Output-Routine als T.IW.OUT zusätzlich im Assembler-Quelltext gelistet, doch sind die Maschinenprogramme entbehrlich, weil sie vom Applesoft-Programm IW.DEMO automatisch gepokt werden. Die BRUN-fähige Steuer-Sequenz-Datei, die mit IW.DEMO erzeugt wird, liegt im Bereich $0300 bis maximal $03CF.

Kurzhinweise

1. Zweck:
Initialisierung des Imagewriters durch mit IW.DEMO generierbare, BRUN-fähige Steuer-Sequenz-Programme.
2. Konfiguration:
Apple IIe und IIc (auch II+, jedoch mit gewissen Einschränkungen wegen der Tastatur); Imagewriter, bei IIe/+ mit Super-Serial-Card; DOS 3.3 oder ProDOS
4. Test:
RUN IW.DEMO
4. Sammeldisk:
IW.DEMO
Applesoft-Programm
T.IW.IN
IW.IN
T.IW.OUT
IW.OUT
(Big-Mac-Quell- und Objektdateien zu den gepokten Routinen)
ECHO
(Mit RUN ECHO starten zum Erzeugen eines 40-Z/Z-Bildschirmechos während des Druckens)

Apple IIe Compatible

jetzt mit 80 Zeichen Karte und 128 KB RAM

Technische Beschreibungen der Geräte siehe Testbericht „Peeker 10/85"
Neben bestem Fachservice bieten wir Ihnen ebenfalls
Apple IIe + II+ Interface Karten + Zubehör in bester High-Tech Qualität.

Händleranfragen erwünscht unter Tel. 02191/50041

Es beraten Sie gerne Ihre Fachhändler:

Electronic-Shop
Friedhelm Mayer
1000 Berlin 10, Spielhagenstraße 10
☎ 030 / 3422156

Menkens Elektronik
Ulrich Menkens
2870 Delmenhorst, Grünestraße 70
☎ 04221 / 14591

Holzinger Elektronik
4000 Düsseldorf, Kölner Straße 16–18
☎ 0211 / 353969

Electronic Shop Chr. Kaup
4400 Münster, Bremer Platz 42–46
☎ 0251 / 665887

Müller – Elektronik
4410 Warendorf, Freckenhorster Straße 31
☎ 02581 / 1767

Gelsen Electronic
4650 Gelsenkirchen,
Kurt-Schumacher-Straße 124
☎ 0209 / 83033

Heitmann Elektronik
4750 Unna, Gerhard-Hauptmann-Straße 20
☎02303 / 12436

Computer-Technik P. Schillhofer
4905 Spenge, Niedernstraße 12
☎ 05225 / 3184

Sebus-Elektronik
5100 Aachen, Pontstraße 119
☎ 02 41 / 36898

Räbiger Computersysteme
5160 Düren, Veldener Straße 65
☎02421 / 43877

VID DATA SYSTEME
WEISING KG
5483 Bad Neuenahr · Postf. 933
☎ 02641 / 1478

MiCom-Computer Olaf Mertens
5630 Remscheid 11,
Grünenplatzstraße 16–18
☎ 02191 / 590313

MicroComp GmbH
6200 Wiesbaden, Klarentaler Str. 6
☎ 06121 / 45377

Dontenwill GmbH
7000 Stuttgart 1, Kronprinzenstraße 6
☎ 0711 / 294665

electronic + computer GRIGENTIN + FALK
7580 Bühl, Hauptstraße 17,
☎ 07223 / 21170

Dontenwill GmbH
8000 München 2, Landwehrstraße 40
☎ 089 / 597993

Thönnes Elektronik
8000 München 70
Fürstenrieder Straße 206
☎ 089 / 7148285

Computerstudio Landshut
8300 Landshut, Bismarckplatz 18
☎ 0871 / 28275

ALPHATRON
marco hildebrandt
8520 Erlangen · Luitpoldstraße 22
☎ 09131 / 22600

Tabelle der Steuerbefehle

↑X: Control-X
* : Normaleinstellung nach dem Einschalten (Spalte N)
x : dezimale Ziffer (z.B. xxx = 132)
n : dez. ASCII von x (z.B. -> nnn = 49,51,50)
h : hex. ASCII von x (z.B. -> hhh = 31,33,32)
c : Zeichen und ASCII (z.B. A -> dez. 65 bzw. $41)

Beschreibung	Tasten	Dezimal (Hexadezimal)	N
Zeichenabstand (Zeichen pro Inch) SW 1-6,1-7			
Weit (9)	ESC n	27,110 (1B,6E)	
Pica (10)	ESC N	27, 78 (1B,4E)	
Elite (12)	ESC E	27, 69 (1B,45)	
Halbeng (13.4)	ESC e	27,101 (1B,65)	
Eng (15)	ESC q	27,113 (1B,71)	
Sehr eng (17)	ESC Q	27, 81 (1B,51)	
Proportionalschrift			
Pica prop. (144 Pkt.p.Inch)	ESC p	27,112 (1B,70)	
Elite prop. (160 Pkt.p.Inch)	ESC P	27, 80 (1B,50)	
Abstand d. Zeichen (x = 0 - 9)	ESC s x	27,115,n (1B,73,h)	1
zusätzl. Abstand d.Zeich.in Elite (x = 1 - 6)	ESC x	27,n (1B,h)	-
Fremdsprachliche Zeichensätze SW 1-1,1-2,1-3			
Amerikanisch	ESC Z ↑G ↑§	27,90,7,0 (1B,5A,07,00)	
	oder		
	ESC Z ↑E ↑§	27,90,5,0,27,68,2,0	
	ESC D ↑B ↑§	(1B,5A,05,00,1B,44,02,00)	
Britisch	ESC Z ↑D ↑§	27,90,4,0,27,68,3,0	
	ESC D ↑C ↑§	(1B,5A,04,00,1B,44,03,00)	
Deutsch	ESC Z ↑C ↑§	27,90,3,0,27,68,4,0	
	ESC D ↑D ↑§	(1B,5A,03,00,1B,44,04,00)	
Französisch	ESC Z ↑A ↑§	27,90,1,0,27,68,6,0	
	ESC D ↑F ↑§	(1B,5A,01,00,1B,44,06,00)	
Schwedisch	ESC Z ↑B ↑§	27,90,2,0,27,68,5,0	
	ESC D ↑E ↑§	(1B,5A,02,00,1B,44,05,00)	
Italienisch	ESC Z ↑F ↑§	27,90,6,0,27,68,1,0	
	ESC D ↑A ↑§	(1B,5A,06,00,1B,44,01,00)	
Spanisch	ESC D ↑G ↑§	27,68,7,0 (1B,44,07,00)	
Nullen mit Schrägstrich			
mit Schrägstr.(0)	ESC D ↑§ ↑A	27,68,0,1 (1B,44,00,01)	
ohne Schrägstr.(O)	ESC Z ↑§ ↑A	27,90,0,1 (1B,5A,00,01)	*
Text hervorheben			
Unterstreichen ein	ESC X	27,88 (1B,58)	
aus	ESC Y	27,89 (1B,59)	*
Fettdruck ein	ESC !	27,33 (1B,21)	
aus	ESC "	27,34 (1B,22)	*
Überschrift ein	↑N	14 (0E)	
aus	↑O	15 (0F)	*
Zeichen Wiederhol. (xxx mal c)	ESC R xxx c	27,82,nnn,c (1B,52,hhh,c)	
Rückschritt (nur einfach)	↑H	8 (08)	
Schreibrichtung			
Links nach rechts	ESC >	27,62 (1B,3E)	
Beide Richtungen	ESC <	27,60 (1B,3C)	*
Zeilenabstand			
6 Zeilen pro Inch	ESC A	27,65 (1B,41)	*
8 Zeilen pro Inch	ESC B	27,66 (1B,42)	
Abstand (xx/144)" (xx = 1 - 99)	ESC T xx	27,84,nn (1B,54,hh)	24
Zeilenvorschub			
LF (Line Feed)	↑J	10 (0A)	
VT (Vertical Tab)	↑K	11 (0B)	6
FF (Form Feed)	↑L	12 (0C)	SW 1-4
Mehrfach-LF (x = 1 - (15) s.Handbuch)	↑_ x	31,n (1F,h)	
LF-Richtung vorwärts	ESC f	27,102 (1B,66)	*
rückwärts	ESC r	27,114 (1B,72)	
Auto.LF nach CR			SW 1-8
ein	ESC D (§) ↑§	27,68,128,0 (1B,44,80,00)	
aus	ESC Z (§) ↑§	27,90,128,0 (1B,5A,80,00)	
Auto.CR nach LF ein	ESC 1 0	27,108,48 (1B,6C,30)	*
aus	ESC 1 1	27,108,49 (1B,6C,31)	
Randeinstellung			
Momentane Position als oberen Rand	ESC v	27,118 (1B,76)	
Linken Rand auf Zeichenposition xxx setzen	ESC L xxx	27,76,nnn (1B,4C,hhh)	0
Zeilenabschluß			
Zeilenende nur bei CR	ESC Z § ↑§	27,90,64,0 (1B,5A,40,00)	
bei CR,LF,VT,FF	ESC D § ↑§	27,68,64,0 (1B,44,40,00)	*
Zeilenpuffer- LF überlauf --	ESC D ↑§	27,68,32,0 (1B,44,20,00)	
	ESC Z ↑§	27,90,32,0 (1B,5A,20,00)	*
Datenbytelänge SW 1-5			
8 Bit	ESC Z ↑§ 2	27,90,0,50 (1B,5A,00,32)	
7 Bit	ESC D ↑§ 2	27,68,0,50 (1B,44,00,32)	*

IW.DEMO

```
11000 REM IW.DEMO VON THORSTEN SCHUNK, 1985
11010 TEXT : PRINT CHR$ (21): HOME
11020 I = 0:L = 0:PCMD$ = "": REM MEISTBENUTZTE VAR.
      ZUERST DEF. (GESCHWINDIGKEIT)
11030 GOSUB 63000: REM RECHNER-CONFIGURATION ERMITTELN
11040 D$ = CHR$ (4): IF NOT PRODOS THEN D$ = CHR$ (13) +
      D$
11050 GOTO 14000
12000 REM *** PRINTER INPUT ROUTINE ***
12010 REM IN:(AIIE);OUT:PCMD$
12020 REM USES: I,L,P,PCMD$
12030 PRINT : PRINT "DRUCKER: ";: CALL - 868
12040 PCMD$ = "":L = 0
12050 POKE 512,44: POKE 513,0: POKE 514,192: POKE 515,16:
      POKE 516,251: POKE 517,173: POKE 518,0: POKE
      519,192: POKE 520,141: POKE 521,16: POKE 522,2: POKE
      523,44: POKE 524,16: POKE 525,192: POKE 526,96: REM
      TASTATUR-ABFRAGE
12060 CALL 512:P = PEEK (528): REM ASCII-CODE HOLEN
12070 IF P = 255 THEN GOTO 12140: REM DEL
12080 IF P = 141 AND NOT AIIE THEN 12210: REM EINGABE-ENDE
      FUER II,II+
12090 IF PEEK ( - 16287) > 127 THEN 12210: REM EINGABE
      BEENDEN IIE,IIC
12100 IF PEEK ( - 16286) < 128 THEN P = P - 128: REM LOW
      ASCII
12110 IF L = > 170 THEN 12060
12120 PCMD$ = PCMD$ + CHR$ (P):L = L + 1
12130 PRINT P", ";: GOTO 12060
12140 REM DELETE
12150 IF NOT L THEN 12180
12160 P = 0: IF RIGHT$ (PCMD$,1) < > "" THEN P = ASC (
      RIGHT$ (PCMD$,1))
12170 FOR I = 0 TO 2 + (P = > 10) + (P = > 100): PRINT
      CHR$ (8);: CALL - 868: NEXT
12180 L = L - 1: IF L = 0 THEN PCMD$ = "": GOTO 12060
12190 IF L < 0 THEN PCMD$ = "": PRINT "ABBRUCH": GOTO
      12210
12200 PCMD$ = LEFT$ (PCMD$,L): GOTO 12060
12210 PRINT : RETURN
13000 REM *** PRINTER OUTPUT ROUTINE ***
13010 REM ZU DRUCKENDER STRING IN PCMD$ (MAX. 170 ZEICHEN)
13020 REM IN: PCMD$, (S. LINE 13140) ;OUT: -
13030 REM USES:I,PSET,PCMD
13040 REM ES IST RATSAM, DEN DRUCKER AUF 'LF AM ANSCHLAG'
      (27,68,32,0) EINZUSTELLEN, DA DIES NORMALERWEISE DIE
      S(SC) TUT
13050 PCMD = 1: REM +CR (WIE PRINT) +LF (WIE (S)SC)
13060 REM ZUS. EINSPRUNGPKT.(WIE '?;')
13070 IF NOT PSET THEN GOSUB 13130
13080 IF LEN (PCMD$) > 170 THEN PCMD$ = PCMD$ + PCMD$: REM
      STRING TOO LONG ERROR
13090 FOR I = 1 TO LEN (PCMD$): IF MID$ (PCMD$,I,1) = ""
      THEN POKE 800 + I,0: NEXT : GOTO 13110
13100 POKE 800 + I, ASC ( MID$ (PCMD$,I,1)): NEXT : REM
      STRING IN DRUCKERPUFFER POKEN
13110 IF PCMD THEN POKE 800 + I,13: POKE 801 + I,10:I = I
      + 2:PCMD = 0: REM LINE 908
13120 POKE 800 + I,255: CALL 768: RETURN : REM ENDMARKE
      SETZEN UND SENDEN
13130 REM MASCHINENSPR.-ROUTINE EINRICHTEN
13140 REM A2C,SSC,SC(U.U.STREG,TDREG,RDY)
13150 REM AUF RICHTIGE WERTE SETZEN
13160 REM PSET AUF 0 ZURUECKSETZEN, FALLS PAGE 3 BENUTZT
      WIRD
13170 IF NOT SSC AND NOT SC THEN PRINT : PRINT
      "UNTERPROGRAMM NICHT BENUTZEN": PRINT "OHNE VORHER
      VARIABLEN ZU BESTIMMEN!": POP : POP : STOP
13180 I = 768: POKE I,216: POKE I + 8,0: POKE I + 10,160:
      POKE I + 12,185: POKE I + 14,192: POKE I + 15,73:
      POKE I + 17,208: POKE I + 18,249: POKE I + 19,189:
      POKE I + 20,33: POKE I + 21,3: POKE I + 22,201: POKE
      I + 23,255
13190 POKE I + 24,240: POKE I + 25,6: POKE I + 26,153:
      POKE I + 28,192: POKE I + 29,232: POKE I + 30,208:
      POKE I + 31,236: POKE I + 32,96: POKE I + 33,255
13200 IF A2C THEN POKE I + 1,234: POKE I + 2,162: POKE I +
      3,192 + SSC: POKE I + 4,32: POKE I + 5,200: POKE I +
      6,194: POKE I + 7,162: POKE I + 9,234: POKE I +
      11,SSC * 16: POKE I + 13,137: POKE I + 16,80: POKE I
      + 27,136: GOTO 13240
13210 POKE I + 1,141: POKE I + 2,6: POKE I + 3,192: POKE I
      + 4,169: POKE I + 5,0: POKE I + 6,170: POKE I + 7,32
13220 IF SSC THEN POKE I + 9,192 + SSC: POKE I + 11,SSC *
      16: POKE I + 13,137: POKE I + 16,16: POKE I +
      27,136: GOTO 13240
13230 IF SC THEN POKE I + 9,192 + SC: POKE I + 11,SC * 16:
      POKE I + 13,132: POKE I + 16,128: POKE I + 27,128:
      IF TDREG AND STREG THEN POKE I + 13,STREG - 49152:
      POKE I + 27,TDREG - 49152: POKE I + 16,RDY
13240 PSET = 1: RETURN
14000 REM *** HAUPTPROGRAMM ***
14010 REM STELLT 3 UNTERPROGRAMME VOR
14020 IF COL80 THEN C8 = 1: COL80 = 0 :REM 80Z/Z
      hier desaktiviert
14030 IF C8 THEN COL80 = 1: PRINT
14040 HOME : PRINT
14050 PRINT "DIESE IMAGEWRITER-UTILITY STELLT": PRINT
      "DREI UNTERROUTINEN VOR:"
14060 PRINT : PRINT "- DIE PRINTER-INPUT-ROUTINE"
14070 PRINT "ES WERDEN ALLE LOW-ASCII-";: IF AIIE THEN
      PRINT : PRINT "UND MIT DER SOLID-APPLE-TASTE": PRINT
      "AUCH ALLE HIGH-ASCII-"
14080 PRINT "ZEICHEN ALS EINGABE AKZEPTIERT.": IF AIIE
      THEN PRINT "DIE DEL-TASTE LOESCHT DIE LETZTE
      TASTE.": PRINT "OPEN-APPLE-";
14090 PRINT "RETURN BEENDET DIE EINGABE."
14100 PRINT : PRINT "- DIE PRINTER-OUTPUT-ROUTINE"
14110 PRINT "SIE SENDET JEDEN BELIEBIGEN STRING": PRINT
      "DIREKT AN DEN DRUCKER."
14120 PRINT : PRINT "- DIE COMPUTER-ID-ROUTINE"
14130 PRINT "MIT IHR KANN DIE RECHNERCONFIGURATION": PRINT
      "ERMITTELT WERDEN, DAMIT": PRINT "ALLE PROGRAMMTEILE
      KORREKT ARBEITEN"
14140 PRINT : GET A$: HOME : PRINT "SIE KOENNEN JETZT
      STEUERZEICHEN": PRINT "EINGEBEN, DIE JEWEILS
      LETZTE": PRINT "STEUERZEICHENSEQUENZ LAESST SICH":
      PRINT "UNTER EINEM DATEINAMEN IHRER WAHL": PRINT
      "ABSPEICHERN": PRINT : GET A$
14150 HOME : PRINT : PRINT "1 STEUER-SEQUENZ EINGEBEN":
      PRINT : PRINT "2 STEUERSEQUENZ SPEICHERN": PRINT :
      PRINT "3 ENDE": PRINT : GET A$: ON A$ < > "1" AND A$
      < > "2" AND A$ < > "3" GOTO 14150: ON A$ = "3" GOTO
      14180: ON A$ = "2" GOTO 14170
14160 PRINT : PRINT "EINGABE MIT (OFFENER-APFEL-)RETURN":
      PRINT "BEENDEN": GOTO 14190
14170 IF PSET THEN PRINT "DIE DATEI KANN SPAETER MIT BRUN
      NAME": PRINT "GESTARTET WERDEN": PRINT : PRINT
      D$;"CATALOG": INPUT "NAME:";FILE$: PRINT
      D$;"BSAVE"FILE$",A$300,L"I + 33: GOTO 14150
14180 TEXT : PRINT CHR$ (21): PRINT D$"PR#0": HOME : END
14190 GOSUB 12000: REM GET INPUT
14200 IF PCMD$ < > "" THEN GOSUB 13000: PRINT "O.K.": REM
      STRING AN DRUCKER SENDEN
14210 PRINT : GOTO 14150
63000 REM *** COMPUTER ID ***
63010 REM SETZT AIIE, A2C, COL80, SSC, SC ABHAENGIG VON
      HARDWARE
63020 REM UND PRODOS JE NACH DOS-VERSION
63030 AIIE = 0:A2C = 0:COL80 = 0:SSC = 0:SC = 0
63040 IF PEEK (64435) = 6 THEN AIIE = 1: REM
      IDENTIFICATION BYTE FUER IIE
63050 IF AIIE AND PEEK (64448) = 0 THEN A2C = 1:COL80 =
      1:SSC = 1: GOTO 63110: REM IDENTIFICATION BYTE FUER
      IIC
63060 IF INT ( PEEK (49932) / 16) = 8 THEN COL80 = 1: REM
      80COL-CARD IN SLOT 3
63070 FOR SLOT = 1 TO 7
63080 IF INT ( PEEK (49164 + SLOT * 256) / 16) = 3 THEN
      GOSUB 63130
63090 NEXT
63100 IF NOT SSC AND NOT SC THEN PRINT "DRUCKER-KARTE
      NICHT ERKANNT!": PRINT : PRINT "BITTE REMS NACH DEM
      'COMPUTER ID' LESEN": POP : STOP
63110 IF PEEK (978) = 190 THEN PRODOS = 1: REM REENTER DOS
      VECTOR
63120 RETURN
63130 IF PEEK (49163 + SLOT * 256) = 01 THEN SSC =
      SLOT:SLOT = 7: RETURN : REM SUPER SERIAL CARD
63140 IF PEEK (49163 + SLOT * 256) = 88 THEN SC =
      SLOT:SLOT = 7: RETURN : REM ALTE SERIAL CARD
63150 RETURN
63160 REM ANMERKUNGEN, FALLS DRUCKER-KARTE NICHT ERKANNT
      WURDE
63170 REM ZEILEN 63070 BIS 63140 LOESCHEN UND DURCH EIGENE
      ROUTINE NACH FOLGENDEN ANWEISUNGEN ERSETZEN
63180 REM SSC UND SC AUFGRUND DER STATUS- UND
      TRANSFER-REGISTER DER DRUCKER-KARTE BESTIMMEN
63190 REM SSC = N, FALLS STREG = $CN89 UND TDREG = $CN88
63200 REM SC = N, FALLS STREG = $CN84 UND TDREG = $CN80
63210 REM WOBEI N DER DRUCKER-SLOT IST
63220 REM FALLS DIE KARTE ANDERE AUSGABESTELLEN BENUTZT,
      SO IST SC=N , STREG UND TDREG SIND AUF DIE
      ENTSPRECHENDEN WERTE UND RDY AUF DEN
      'DRUCKER-BEREIT'-WERT DES STREG ZU SETZEN
```

IW.IN

Bereits in IW.DEMO als Data-Statements enthalten

```
                    1             ORG  $0200
                    2   *
                    3   * Tastatur direkt abfragen
                    4   *
                    5   KBD       EQU  $C000
                    6   STROBE    EQU  $C010
                    7   PUFFER    EQU  $0210
0200: 2C 00 C0      8   ENTRY     BIT  KBD
0203: 10 FB         9             BPL  ENTRY
0205: AD 00 C0      10            LDA  KBD
0208: 8D 10 02      11            STA  PUFFER
020B: 2C 10 C0      12            BIT  STROBE
020E: 60            13            RTS
```

IW.OUT

Bereits in IW.DEMO als DATA-Statements enthalten

```
                    1             ORG  $0300
                    2   *
                    3   * IW-Output-Routine
                    4   *
                    5   A2C       EQU  0          ;IIc: 1
                    6   SC        EQU  0          ;alte SC:1
                    7   SLOT      EQU  1          ;Drucker
                    8   *
                    9   SLOTCX    EQU  $C006      ;SLOT-ROM
                    10  INITCARD  EQU  $C0+SLOT*$100 ;Einsprung
                    11  INIT2C    EQU  $C2C8      ;IIc-Init
                    12  STREG     EQU  $C089      ;Status-Reg.
                    13  TDREG     EQU  $C088      ;Transfer-Reg.
                    14  STREGSC   EQU  $C084      ;Status-Reg.
                    15  TDREGSC   EQU  $C080      ;Transfer-Reg.
                    17  *
0300: D8            18  ENTRY     CLD
                    19            DO   A2C        ;IIc
                    20            NOP
                    21            LDX  #$C0+SLOT  ;Index
                    22            JSR  INIT2C     ;Init Port
                    23            LDX  #$00       ;Index=0
                    24            NOP
                    25            ELSE            ;II/IIe/II+
0301: 8D 06 C0      26            STA  SLOTCX     ;Enable
0304: A9 00         27            LDA  #$00
0306: AA            28            TAX
0307: 20 00 C1      29            JSR  INITCARD   ;Drucker
                    30            FIN
030A: A0 10         31            LDY  #$10*SLOT  ;Offset
                    32  TRANS     DO   SC         ;fertig?
                    33            LDA  STREGSC,Y
                    34            EOR  #$80
                    35            ELSE
030C: B9 89 C0      36            LDA  STREG,Y
                    37            DO   A2C
                    38            EOR  #$50
                    39            ELSE
030F: 49 10         40            EOR  #$10
                    41            FIN
                    42            FIN
0311: D0 F9         43            BNE  TRANS      ;nein
0313: BD 21 03      44            LDA  BUFF,X     ;Zeichen
0316: C9 FF         45            CMP  #$FF       ;Ende?
0318: F0 06         46            BEQ  RETURN     ;ja
                    47            DO   SC         ;senden
                    48            STA  TDREGSC,Y
                    49            ELSE
031A: 99 88 C0      50            STA  TDREG,Y
                    51            FIN
031D: E8            52            INX             ;nächstes
031E: D0 EC         53            BNE  TRANS      ;stets
0320: 60            54  RETURN    RTS
0321: FF            55  BUFF      DFB  $FF        ;Endmarke
```

Ab $0321 befinden sich später die Ctrl-Zeichen, die über IW.DEMO eingegeben worden sind.

BILDSCHIRM-ECHO

Mehrere Peeker-Leser haben bereits angefragt, wie man beim Imagewriter das Bildschirmecho während des Druckens einstellt. Dies ist nicht ganz einfach. Falls ein Apple IIe mit Super-Serial-Card in Slot 1 benutzt wird, kann man ein 40-Z/Z-Echo mit folgendem Miniprogramm erzeugen. (Es setzt Bit 7 von Speicherstelle $07F9. Siehe "Super Serial Card Manual", S. 53.)
Beim Apple IIe mit den neuen ROMs funktioniert auch ein 80-Z/Z-Echo. us

```
10 DATA 173,249,7,9,128,141,249,7,96
20 FOR X = 700 TO 700 + 8: READ Y: POKE X,Y:
   NEXT : CALL 700
```

ASCII-Texte im HGR-Bildschirm

Mit einem Zeichengenerator-Programm

von Markus Geltenpoth

Das nachstehend beschriebene Programm, das aus zwei Modulen besteht, dient
1. zum Generieren eigener Zeichensätze für die HGR-Grafik, die hinsichtlich des Bit-Musters den Zeichen des 40-Z/Z-Zeichensatzes entsprechen, sowie
2. zur simultanen Wiedergabe von Texten des 40-Z/Z-Textbildschirm-Speichers auf der HGR-Grafik-Seite 1.

Viele Apple-II-Besitzer haben sich sicher schon oft darüber geärgert, daß man beim Apple II keine eigenen Zeichen definieren kann. Deshalb habe ich ein Programm namens ASCII.EDITOR geschrieben, das es erlaubt, bis zu 224 Zeichen selbst zu definieren.
Ein weiteres Modul namens COPY.TEXT tastet später den Text-Bildschirm ab und überprüft bei jedem Zeichen, ob es einen Wert größer 32 hat. Wenn dies zutrifft, sucht sich das Programm den Speicherplatz des dazugehörigen, selbstdefinierten Zeichens und speichert es im Hires-Bildschirm ab. Wenn der Wert des Zeichens im Textmodus kleiner als 32 ist, wird der Hires-Bildschirm an der dazugehörigen Stelle nicht verändert. *Alles, was Sie also normalerweise auf dem 40-Z/Z-Textbildschirm sehen würden, wird statt dessen automatisch auf der Hires-Seite angezeigt.*

1. Das Applesoft-Programm

Das Applesoft-Programm **ASCII.EDITOR** wird benötigt, um einen neuen Zeichensatz zu definieren oder einen alten umzudefinieren und dann auf Diskette zu speichern. Man hat die Möglichkeit, den Speicherbereich der zu erstellenden Zeichen zu löschen oder ihn so zu lassen, wie er gerade ist. Letzteres gestattet, alte Zeichensätze vor dem Start des Programms einzuladen und diese vom Programm aus zu verbessern. Es wird dann jeweils vor dem Definieren das schon erstellte Zeichen am Bildschirm gezeigt. Zu diesem Zweck ist auf der Peeker-Sammeldiskette der Zeichensatz-File **ASCII.CODES** aufgenommen.
Das Programm ist menügesteuert und erklärt sich von selbst.

2. Das Maschinenprogramm

Wie schon erwähnt, kopiert die Maschinenroutine **COPY.TEXT** die Zeichen des Textbildschirms auf den Hires-Bildschirm. Wenn man also seine selbstdefinierten Zeichen sehen will, muß man 1. das Maschinenprogramm und 2. den File mit den definierten Zeichen eingeladen haben. Dann kann man mit „CALL 768" parallel zu den Zeichen im Textmodus die selbstdefinierten Zeichen in die Grafik-Seite 1 übertragen. Nach dem Einschalten der Grafik mit HGR kann man mit „POKE -16302,0" auf Hires über den gesamten Bildschirm umschalten.
Sicher werden Sie sich fragen, warum ich nicht mit indirekt indizierten Befehlen arbeite, sondern den „unschöneren" Weg der erheblich längeren Routine gewählt habe. Das hat seinen guten Grund: Da das Programm immerhin 8K kopiert, muß man schon auf die Geschwindigkeit der Op-Codes achten. Durch die direkt-indizierte Adressierung gewinnt das Programm wertvolle Millisekunden. Das macht sich dann bemerkbar, wenn man den Input-Vektor auf das Programm richtet.

3. Zusätzliche Hilfen

Um nicht immer „CALL 768" eintippen zu müssen, kann auch der Zeiger des &-Kommandos mit folgenden Pokes auf den Programmanfang des Maschinenprogramms gesetzt werden:
POKE 1014,0: POKE 1015,3
Nach diesen Pokes wird immer nach Eingabe des &-Befehls der HGR-Bildschirm aktualisiert.
Noch eleganter ist es, den Keyboard-Vektor KSWL (Adresse $0038, $0039) auf den Programmanfang zu richten und am Ende des Maschinenprogramms aus dem „RTS" ein „JMP $FD1B" (Input-ROM-Routine KEYIN) zu machen. Von Applesoft aus kann das folgendermaßen geschehen:
POKE 917,76: POKE 918,27: POKE 919,253
(Änderung des RTS in JMP $FD1B)
POKE 56,0: POKE 57,3: CALL 1002
(Einbinden des neuen Vektors in DOS 3.3)
Die letzten 2 Pokes und der CALL-Aufruf müssen unbedingt in *einer* Zeile hintereinander eingegeben werden. Danach wird Ihnen zunächst noch nichts Besonderes auffallen, aber wenn Sie auf HGR umschalten, werden Sie sehen, daß jetzt der Text-Bildschirm immer automatisch kopiert wird (siehe hierzu **TEXT.COPY-.DEMO**).

Kurzhinweise

1. Zweck:
Erstellung eines eigenen Zeichensatzes; Kopieren des 40-Z/Z-Text-Bildschirms in den HGR-Bildschirm.
2. Konfiguration:
speziell für Apple II+ (ggf. mit Kleinschreibumrüstsatz) gedacht, aber auch mit Einschränkungen auf IIe und IIc einsetzbar; hier werden jedoch die Kleinbuchstaben ($40 ff.) nicht korrekt konvertiert. Notbehelf: Nach FLASH werden Großbuchstaben als Kleinbuchstaben auf dem HGR-Bildschirm ausgegeben.
Nur DOS 3.3
3. Test:
RUN COPY.TEXT.DEMO
4. Sammeldisk:
ASCII.EDITOR
(Applesoft-Zeicheneditor)
COPY.TEXT.DEMO
(Applesoft-Demoprogramm)
T.COPY.TEXT
(Big-Mac-Quelltext)
COPY.TEXT
(Maschinenprogramm zur Übertragung der Zeichen)
ASCII.CODES
(Zeichensatz-File)

Speicheraufteilung

```
$Ø3ØØ–$Ø397  Maschinenroutine 1
$Ø399–$Ø3B7  Maschinenroutine 2 (für ASCII.EDITOR)
$6ØØØ–$67ØØ  Selbstdefinierte Zeichen
```

COPY.TEXT

```
                    1             ORG  $3ØØ
                    2   *
                    3   * Markus Geltenpoth
                    4   *
                    5   * Dieses Programm kopiert Zeichen
                    6   * vom Text- in den Hires-Bildschirm
                    7   * und lädt dabei für jedes Zeichen
                    8   * aus dem Text-Bildschirm ein Zeichen
                    9   * aus der Tabelle ab $6ØØØ.
                    1Ø  * Ab dieser Adresse werden von dem
                    11  * Zeicheneditor selbstdefinierte
                    12  * Zeichen gespeichert.
                    13  *
                    14  Z1        EQU  $6ØØØ
                    15  Z2        EQU  $6ØE8
                    16  Z3        EQU  $61C8
                    17  Z4        EQU  $62A8
                    18  Z5        EQU  $6388
                    19  Z6        EQU  $6468
                    2Ø  Z7        EQU  $6548
                    21  Z8        EQU  $6628
                    22  *
                    23  * Z1–Z8 sind die Anfangsadressen
                    24  * von 224 Bytes. Sie stellen die
                    25  * verschiedenen Zeilen eines
                    26  * Zeichens dar.
                    27  *
                    28  IOSAVE    EQU  $FF4A
                    29  IOREST    EQU  $FF3F
                    3Ø  KEYIN     EQU  $FD1B
                    31  *
Ø3ØØ: 2Ø 4A FF      32            JSR  IOSAVE
Ø3Ø3: A2 ØØ         33            LDX  #$ØØ        ;Zähler
Ø3Ø5: BD ØØ Ø4      34  LOOP      LDA  $4ØØ,X      ;Zeichen
Ø3Ø8: 38            35            SEC
Ø3Ø9: E9 2Ø         36            SBC  #$2Ø
Ø3ØB: 9Ø 31         37            BCC  LASSEN
Ø3ØD: A8            38            TAY              ;retten
Ø3ØE: B9 ØØ 6Ø      39            LDA  Z1,Y
Ø311: 9D ØØ 2Ø      4Ø  P1        STA  $2ØØØ,X
Ø314: B9 E8 6Ø      41            LDA  Z2,Y
Ø317: 9D ØØ 24      42  P2        STA  $24ØØ,X
Ø31A: B9 C8 61      43            LDA  Z3,Y
Ø31D: 9D ØØ 28      44  P3        STA  $28ØØ,X
Ø32Ø: B9 A8 62      45            LDA  Z4,Y
Ø323: 9D ØØ 2C      46  P4        STA  $2CØØ,X
Ø326: B9 88 63      47            LDA  Z5,Y
Ø329: 9D ØØ 3Ø      48  P5        STA  $3ØØØ,X
Ø32C: B9 68 64      49            LDA  Z6,Y
Ø32F: 9D ØØ 34      5Ø  P6        STA  $34ØØ,X
Ø332: B9 48 65      51            LDA  Z7,Y
Ø335: 9D ØØ 38      52  P7        STA  $38ØØ,X
Ø338: B9 28 66      53            LDA  Z8,Y
Ø33B: 9D ØØ 3C      54  P8        STA  $3CØØ,X
Ø33E: E8            55  LASSEN    INX              ;nächstes
Ø33F: DØ C4         56            BNE  LOOP
Ø341: AD Ø7 Ø3      57            LDA  LOOP+2
Ø344: C9 Ø7         58            CMP  #$Ø7        ;Schirmende?
Ø346: FØ 1D         59            BEQ  ENDE        ;ja, Endroutine
                    6Ø  *
                    61  * Diese Routine erhöht alle Pointer
                    62  *
Ø348: EE Ø7 Ø3      63            INC  LOOP+2
Ø34B: EE 13 Ø3      64            INC  P1+2
Ø34E: EE 19 Ø3      65            INC  P2+2
Ø351: EE 1F Ø3      66            INC  P3+2
Ø354: EE 25 Ø3      67            INC  P4+2
Ø357: EE 2B Ø3      68            INC  P5+2
Ø35A: EE 31 Ø3      69            INC  P6+2
Ø35D: EE 37 Ø3      7Ø            INC  P7+2
Ø36Ø: EE 3D Ø3      71            INC  P8+2
Ø363: DØ AØ         72            BNE  LOOP
Ø365: A9 Ø4         73  ENDE      LDA  #$Ø4
Ø367: 8D Ø7 Ø3      74            STA  LOOP+2
Ø36A: A9 2Ø         75            LDA  #$2Ø
Ø36C: 8D 13 Ø3      76            STA  P1+2
Ø36F: A9 24         77            LDA  #$24
Ø371: 8D 19 Ø3      78            STA  P2+2
Ø374: A9 28         79            LDA  #$28
Ø376: 8D 1F Ø3      8Ø            STA  P3+2
Ø379: A9 2C         81            LDA  #$2C
Ø37B: 8D 25 Ø3      82            STA  P4+2
Ø37E: A9 3Ø         83            LDA  #$3Ø
Ø38Ø: 8D 2B Ø3      84            STA  P5+2
Ø383: A9 34         85            LDA  #$34
Ø385: 8D 31 Ø3      86            STA  P6+2
Ø388: A9 38         87            LDA  #$38
Ø38A: 8D 37 Ø3      88            STA  P7+2
Ø38D: A9 3C         89            LDA  #$3C
Ø38F: 8D 3D Ø3      9Ø            STA  P8+2
Ø392: 2Ø 3F FF      91            JSR  IOREST
Ø395: 6Ø            92            RTS
Ø396: EA            93            NOP
Ø397: EA            94            NOP
                    95  *
                    96  * Statt des RTS und der 2 NOPs
                    97  * kann bei Benutzung des Input-
                    98  * Vektors der Befehl JMP KEYIN
                    99  * benutzt werden.
                    1ØØ *
                    1Ø1 * Neue Assemblerroutine: CALL 921
                    1Ø2 * Diese Routine zerlegt ein Byte
                    1Ø3 * in seine Bits.
                    1Ø4 *
Ø398: ØØ            1Ø5 BYTE      HEX  ØØ
Ø399: AC 98 Ø3      1Ø6           LDY  BYTE
Ø39C: A9 4Ø         1Ø7           LDA  #%Ø1ØØØØØØ
Ø39E: 8D 98 Ø3      1Ø8           STA  BYTE
Ø3A1: A2 Ø6         1Ø9           LDX  #6
Ø3A3: 98            11Ø LOOP1     TYA
Ø3A4: 2D 98 Ø3      111           AND  BYTE
Ø3A7: 9D B1 Ø3      112           STA  BITS,X
Ø3AA: 4E 98 Ø3      113           LSR  BYTE
Ø3AD: CA            114           DEX
Ø3AE: 1Ø F3         115           BPL  LOOP1
Ø3BØ: 6Ø            116           RTS
                    117 BITS      DS   7
                    118 *
                    119 * Zusätzlich kann auch der &-Vektor
                    12Ø * (Adresse $3F5–$3F7) auf das Programm
                    121 * gesetzt werden:
                    122 * $3F5 = $4C, $3F6 = $ØØ, $3F7 = $Ø3)
```

184 Bytes

ASCII.EDITOR

```
1ØØØ REM ** ASCII.EDITOR VON MARKUS GELTENPOTH **
1Ø1Ø PRINT CHR$ (4);"BLOAD COPY.TEXT"
1Ø2Ø LOMEM: 26624: PRINT CHR$ (21)
1Ø3Ø DIM A(8)
1Ø4Ø HOME : PRINT "    ZEICHEN-DEFINITIONSPROGRAMM"
1Ø5Ø VTAB 3: PRINT "MIT DIESEM PROGRAMM KOENNEN EIGENE
     ZEICHEN DEFFINIERT WERDEN. DIESE ZEI-   CHEN WERDEN
     DANN GEGEN DIE ANGEGEBENEN  ZEICHEN AUSGETAUSCHT UND
     KOENNEN DURCH  AUFRUF VON 'CALL 768' AUF DEN HIRES-"
1Ø6Ø PRINT "BILDSCHIRM UEBERTRAGEN WERDEN.          ES
     KOENNEN INSGESAMT 224 ZEICHEN DE-    FINIERT WERDEN,
     DIE DURCH EIN MASCHINEN-PROGRAMM NACH HIRES KOPIERT
     WERDEN. ALLECONTROL-ZEICHEN IM PRINT-MODUS WERDEN
     IGNORIERT."
1Ø7Ø PRINT "DURCH DIESES PROGRAMM HAT MAN DIE MOEG-
     LICHKEIT, BEWEGTE BILDER IM TEXT-MODUS  DARZUSTELLEN,
     UM SIE DANN IN BESSERER   GRAFIK IN DIE HIRES ZU
     UEBERTRAGEN"
1Ø8Ø VTAB 22: PRINT " PROGRAMM VON MARKUS GELTENPOTH, 1985"
1Ø9Ø PRINT " ZEICHENSPEICHERBEREICH LOESCHEN (J/N)";: GET
     AZ$: IF AZ$ < > "J" THEN 111Ø
11ØØ FOR I = 24576 TO 26623: POKE I,Ø: NEXT : REM **
     LOESCHEN DES SPEICHERBEREICHS DER DEFINIERTEN ZEICHEN
     **
1111Ø VTAB 23: PRINT "                  TASTE": VTAB 23:
     HTAB 23: GET A$
112Ø Z = Ø: REM ** ZEICHEN-NUMMER **
113Ø HOME : PRINT "            ZEICHEN-NR.: ";Z + 1
114Ø Z1 = Ø
115Ø PRINT " DEFINITION FUER ZEICHEN: "; CHR$ (32 + Z);"
     (CHR$(";Z + 32;"))"
116Ø INVERSE : VTAB 5: HTAB 15: PRINT ".Ø123456."
117Ø IF AZ$ < > "J" THEN 136Ø
118Ø FOR I = Ø TO 7: VTAB 6 + I: HTAB 15: PRINT I: VTAB 6
     + I: HTAB 23: PRINT ".": NEXT
119Ø VTAB 14: HTAB 15: PRINT ".......": NORMAL
12ØØ VTAB 16: PRINT " STEUERUNG:"
1211Ø PRINT "     I       SP = PUNKT SETZEN/LOESCHEN"
1222Ø PRINT "   J-+-K     N  = NAECHSTES ZEICHEN"
1233Ø PRINT "     M       E  = ENDE"
1244Ø X = 16:Y = 6
1255Ø P1 = SCRN( X - 11,Y * 2 - 2):P2 = SCRN( X - 1,Y * 2 -
     1)
1266Ø VTAB Y: HTAB X: GET A$: COLOR= P1: PLOT X - 1,Y * 2 -
     2: COLOR= P2: PLOT X - 1,Y * 2 - 1
1277Ø X = X + (A$ = "K") * (X < 22) - (A$ = "J") * (X > 16)
1288Ø Y = Y + (A$ = "M") * (Y < 13) - (A$ = "I") * (Y > 6)
1299Ø P = SCRN( X - 1,Y * 2 - 1)
```

```
1300 IF A$ = " " THEN IF P = 2 THEN VTAB Y: HTAB X: PRINT
     " "
1310 IF A$ = " " THEN IF P = 10 THEN INVERSE : VTAB Y:
     HTAB X: PRINT " ": NORMAL
1320 IF A$ = "N" THEN 1430
1330 IF A$ = "E" THEN 1520
1340 Z1 = Z1 + 1
1350 GOTO 1250
1360 REM ** GESPEICHERTES ZEICHEN AUF DEN BILDSCHIRM
     BRINGEN **
1370 FOR I = 0 TO 7:A = PEEK (24576 + I * 224 + Z): POKE
     920,A: CALL 921: FOR J = 0 TO 6:A(J) = PEEK (945 +
     J): NEXT
1380 VTAB 6 + I: INVERSE : HTAB 15: PRINT I;: NORMAL : FOR
     J = 0 TO 6: IF A(J) = 0 THEN PRINT " ";: NEXT : GOTO
     1400
1390 INVERSE : PRINT " ";: NORMAL : NEXT
1400 INVERSE : PRINT ".": NORMAL
1410 NEXT I: INVERSE
1420 GOTO 1190
1430 REM ** ALTES ZEICHEN BERECHNEN UND IN DEN SPEICHER
     'POKEN' **
1440 IF Z1 = 0 THEN 1480
1450 FOR I = 1 TO 8:A = 0: FOR J = 1 TO 7:A1 = 0: IF SCRN(
     J + 14,(I + 5) * 2 - 1) = 2 THEN A1 = 1
1460 A = A + A1 * 2 ↑ (J - 1): NEXT
1470 POKE 24576 + (I - 1) * 224 + Z,A: NEXT
1480 Z = Z + 1: REM ** NAECHSTES ZEICHEN DEFINIEREN **
1490 IF A$ = "E" THEN RETURN
1500 IF Z + 32 = 256 THEN 1540
1510 GOTO 1130
1520 REM ** ENDE DER DEFINITION, SPEICHERBEREICH MIT
     DEFINIERTEN ZEICHEN AUF DISKETTE SPEICHERN **
```

```
1530 GOSUB 1430: REM ** LETZTES ZEICHEN SPEICHERN **
1540 HOME : VTAB 3: PRINT "DIESEN ZEICHEN-FILE SICHERN?
     (J/N) ";: GET A$: IF A$ = "N" THEN 1590
1550 PRINT : INPUT "NAME DES FILES: ";N$
1560 PRINT : PRINT "DISKETTE EINLEGEN UND TASTE
     DRUECKEN";: GET A$
1570 PRINT ""
1580 PRINT CHR$ (4);"BSAVE ";N$;",A$6000,L$700"
1590 HOME : VTAB 3: PRINT "UM DIESE ZEICHEN AUF DEM
     BILDSCHIRM ZU  SEHEN, MUESSEN SIE ERST DAS PROGRAMM:
     'COPY.TEXT' (A$0300) UND DIESEN ZEICHEN-FILE (A$6000)
     EINLADEN."
1600 PRINT "IN HGR KANN MAN DANN NACH 'CALL 768'    DIE
     DEFINIERTEN ZEICHEN, ANSTELLE DER"
1610 PRINT "IM TEXT-MODUS GEPRINTETEN ZEICHEN AUF   DEM
     BILDSCHIRM SEHEN."
1620 PRINT : PRINT : END
```

COPY.TEXT.DEMO

```
10 PRINT CHR$ (4)"BLOAD ASCII.CODES"
20 PRINT CHR$ (4)"BLOAD COPY.TEXT"
30 PRINT CHR$ (21): REM 80-ZEICHENKARTE ABSTELLEN FUER IIE,
   IIC
40 HGR : POKE - 16302,0: REM NUR GRAFIK
50 HOME : PRINT "AUSGABE UEBER COPY.TEXT? (J/N)";: CALL 768
60 GET A$: PRINT A$: CALL 768: IF A$ = "J" THEN 80
70 POKE 1014,0: POKE 1015,3: END : REM &-VEKTOR SETZEN
80 POKE 917,76: POKE 918,27: POKE 919,253: REM JMP KEYIN
   STATT RTS
90 POKE 56,0: POKE 57,3: CALL 1002: REM EINGABE-VEKTOR
   SETZTEN UND DOS ANHAENGEN
```

Komfortabler INPUT von Strings

von Ulrich Kußmann

Der im Applesoft-ROM implementierte INPUT-Befehl ist nicht gerade luxuriös ausgestattet. Man darf nicht alle Zeichen eingeben und kann die Länge des Eingabe-Strings nicht begrenzen. Für solche Probleme gibt es Bildschirm-Maskengeneratoren, die dieses Problem elegant lösen. Sie haben nur einen Nachteil: Sie kosten Geld! Man braucht nicht immer aufwendige Programme, um seine kleinen Probleme zu lösen. Es gibt auch Kompromisse: **GETTEXT** ist eine komfortable Texteingabe für Applesoft, die auch Textvorgaben aus einer Stringvariablen übernimmt. Es setzt sich zusammen aus dem modifizierten Programm INALL von H. Grumser (Peeker, Heft 5/85) und der eigentlichen INPUT-Routine **GETX**, die übrigens – nach Löschen einiger Zeilen – auch allein in Assemblerprogrammen verwendet werden kann. Diese bietet ein definiertes Eingabefeld zwischen 1 und 255 Zeichen, Delete, Insert und die Escape-Funktion. GETTEXT kann an jede freie Stelle des 48K-Speichers geladen werden und benutzt trotzdem den Ampersand-Vektor. Es ist deshalb sehr einfach in ein Applesoft-Programm einzubinden. Die beiden Versionen für DOS 3.3 und ProDOS unterscheiden sich nur in einer Adreßdefinition. Das Programm läuft auf Apple IIe, IIc und II Plus und unterstützt auch die entsprechenden 80-Zeichenkarten.

INALL wurde an zwei Stellen verändert: Zusätzlich wird die Zahl der einzugebenden Zeichen abgefragt (GETBYT), und statt des Applesoft-INPUT wird GETX und der Teil der INLIN-Routine verwendet, der Bit 7 der Zeichen auf Null setzt und ein Null-Byte an den eingegebenen String hängt (GDBUFS1).

Programmablauf

GETX besitzt drei wichtige Zeiger. Der erste, **ANZAHL**, enthält die zugelassene Zahl an Zeichen. **PTEND** gibt die Zahl der eingegebenen Zeichen an. Wenn PTEND und ANZAHL gleich sind, dann ist das Zeilenende erreicht. Der dritte Zeiger, **PTAKT**, zeigt auf die gerade aktuelle Cursor-Position. PTAKT und PTEND dürfen nicht kleiner als Null und nicht größer als ANZAHL werden (linker und rechter Rand des Eingabefeldes). Zusätzlich wird mit ihrer Hilfe entschieden, ob das Programm die neuen Buchstaben an den Text anhängen oder löschen kann, oder ob es mitten im Text Buchstaben einfügen oder löschen muß (Insert-Modus). Am Beginn von GETX werden PTAKT und PTEND auf Null gesetzt. Ein vierter Pointer (ZEIGER) wird nur verwendet, um einen String aus dem Speicher in den Tastatur-Eingabepuffer zu bewegen.

Nach dem Rückstellen der Zeiger wird das Eingabefeld angelegt. Wenn die beim Aufruf genannte Applesoft-Variable einen String enthält, dann wird ihr Inhalt in das Eingabefeld übernommen. Die Länge des Variableninhaltes darf dabei die vorher definierte Länge des Eingabefeldes nicht überschreiten, sonst erfolgt eine Fehlermeldung. **SIGN** ist das Zeichen für einen Punkt und kann durch andere Zeichen (z.B. $A0; Leertaste) ersetzt werden, wenn das Eingabefeld anders gestaltet werden soll. Nach dem Ausdruck der Punkte wird die Zeichenausgabe wieder auf den Anfang der Eingabezeile bzw. auf PTEND zurückgesetzt. Hierauf folgt die Cursor-Routine: Es wird kein Cursor angezeigt, wenn das Eingabefeld voll (CUR) und PTAKT gleich PTEND ist. Wenn dann ein Eingabeversuch erfolgt, sendet das Programm zur Warnung einen Piepston (EOL).
Um den Strich-Cursor darzustellen, wird zuerst ein Strich ausgegeben (CURSOR) und darauf das Zeichen, auf das PTAKT zeigt. Zwischendurch wird in der Warteschleife KEYWAIT laufend die Speicherstelle $C000 auf eine Tastatureingabe abgefragt. Das Setzen und Löschen des Carry-Flags dient dazu, zwischen Strich- und Zeichenausgabe abzuwechseln.
Wurde eine Taste gedrückt, dann erfolgt eine Verzweigung nach TASTE. Hier wird die eingegebene Taste im X-Register zwischengespeichert und das aktuelle Zeichen noch einmal ausgegeben, denn es könnte ja auch noch der Strich des Cursors auf dem Bildschirm stehen. Es folgt die Rückstellung des Tastatur-Eingaberegisters und eine Abfrage, ob das Zeichen ein Befehl für das Programm oder ein eingegebenes Zeichen ist.
Nach der Entscheidung, ob das eingegebene Zeichen ein Befehl (BACK = Linkspfeil; RETURN = Eingabeannahme; VOR = Rechtspfeil; ESC = Löschen + Return; DEL = Delete) oder eine Eingabe ist, wird die entsprechende Unterroutine abgearbeitet. Bei einer Eingabe muß entschieden werden, ob der Text in den String eingefügt oder angehängt wird. Wichtig ist, daß die beiden Pointer PTEND und PTAKT bei jeder Veränderung entsprechend gestellt werden. Wie man schon bemerkt haben wird, unterhält das Programm eine „doppelte Buchführung“: Die Zeichen werden sowohl in den Bildschirmspeicher als auch in den Tastatur-Eingabepuffer geschrieben.

Die Veränderungen beim Löschen oder Einfügen werden in den Teilen MOVEUP und DEL1 in den Puffer eingetragen; beide

Teile benutzen das Unterprogramm UPDATE, um den Bildschirminhalt zu aktualisieren. An dem gelöschten bzw. gesetzten Carry-Flag kann UPDATE unterscheiden, ob vorher ein Insert- oder ein Lösch-Befehl abgearbeitet wurde.
Da beim II Plus keine DEL-Taste existiert, sollte die DELETE-Zuweisung in Zeile 69 durch „DELETE EQU $84 ;Ctrl-D" ersetzt werden. Dies wirkt sich im Code nur in der Adresse $80EB aus, die von $FF zu $84 geändert werden sollte.

Verschiebbarkeit im Speicher

Wenn ein Programm im Speicher an allen Stellen lauffähig sein soll, dürfen Sprungbefehle (JMP, JSR) nur für im Speicher feststehende Unterprogramme (z.B. ROM-Routinen) verwendet werden. Innerhalb von GETTEXT erfolgen Verzweigungen nur relativ. Weil aber solche Verzweigungen maximal 126 bzw. 129 Bytes lang sein dürfen, „hangelt" es sich an einigen Stellen über mehrere Branch-Befehle zur gewünschten Stelle.
Zusätzlich darf das Programm keine internen Datenregister haben, die mit LDA, STA oder ähnlichen Befehlen abgefragt werden. Die internen Register von GETX liegen in der Zero-Page und sparen so, weil sie oft abgefragt werden, einige Bytes, da die meisten Zero-Page-Befehle nur zwei statt drei Bytes lang sind. Das Programm ist nur bedingt relokativ, weil gleich am Anfang der Ampersand-Vektor gesetzt wird. GETTEXT erfährt beim Starten durch „BRUN GETTEXT" seine jeweilige Position im Speicher vom Betriebssystem (siehe Hinweis zur Anpassung an ProDOS. Auf der Peeker-Sammeldiskette ist die ProDOS-Version unter dem Namen GETTEXT.PRODOS enthalten). Verschöbe man es zu einem späteren Zeitpunkt, dann würde der Applesoft-Interpreter ins Leere springen. Hier liegt auch der einzige Unterschied des Programms zwischen DOS 3.3 und ProDOS. (Das BASIC.SYSTEM 1.0 der ProDOS-Versionen 1.0.1 und 1.0.2 ist gleich).

Ein Beispiel

Wie man das Maschinenprogramm in Applesoft einbinden kann, zeigt das Beispiel **GETTEXT.DEMO**: GETTEXT wird direkt hinter das BASIC-Programm geladen, und der Anfang des freien Speichers (Variablenspeicher) LOMEM wird um die Länge von GETTEXT heraufgesetzt. Vorsicht: Wenn ein solches Programm verlängert wird, muß beim Austesten GETTEXT jedesmal neu geladen werden. Umgeht man dieses durch einen GOTO-Start, dann stürzt das Programm bei Aufruf der Assemblerroutine ab, weil diese von Applesoft überschrieben wurde. (Dieses Programm läuft auf dem Apple II Plus mit Videx-Karte erst nach Anpassung der Tabulierungen. Der HOME-Befehl ist durch „PRINT CHR$ (12)" zu ersetzen.)

Literatur:
U. Stiehl, Apple Assembler, Hüthig Verlag 1984.
U. Stiehl, ProDOS für Aufsteiger, Hüthig Verlag 1984.
D. Worth, P. Lechner, Beneath Apple DOS, Quality Software 1981.
Das Apple-Sonderheft, mc (Nr.89), Franzis Verlag 1984.

Kurzhinweise

1. Zweck:
Hilfsprogramm zur Erweiterung des Tastatur-INPUT-Befehls durch Editierfunktionen; nicht für Disketten-Textdatei-INPUT gedacht!
2. Konfiguration:
Apple II+, IIe, IIc mit 40 Z/Z oder 80 Z/Z; bei 80 Z/Z mit Videx-Karte Programmanpassung von GETTEXT.DEMO erforderlich. Zwei getrennte Versionen des Maschinenprogramms für DOS 3.3 und ProDOS (BASIC.SYSTEM 1.0). Für ProDOS in Zeile 140 von GETTEXT.DEMO den Namen „GETTEXT.PRODOS" einsetzen.
3. Test:
DOS 3.3 booten
RUN GETTEXT.DEMO
4. Sammeldisk:
GETTEXT.DEMO
(Applesoft-Programm)
GETTEXT + T.GETTEXT
(Version für DOS 3.3)
GETTEXTG.PRODOS
(Version für ProDOS)

GETTEXT.DEMO

```
100 REM <<< GETTEXT DEMO >>>
110 HOME
120 REM Startadresse = LOMEM
130 START = PEEK (175) + PEEK (176) * 256
140 PRINT CHR$ (4);"BRUN GETTEXT,A";START
150 REM LOMEM = LOMEM + 402 (Länge von GETTEXT)
160 CLEAR : LOMEM: PEEK (175) + PEEK (176) * 256 + 402
170 PRINT "GETTEXT: Eine komfortable Texteingabe"
180 REM Es folgt der Aufruf von GETTEXT: & Stringvariable,
    Zeichenzahl
190 VTAB 10: HTAB 8: PRINT "Name: ";: & X$,20
200 VTAB 14: HTAB 14: PRINT X$
210 VTAB 1: HTAB 9: INVERSE : PRINT "KORREKTUR?";: NORMAL
220 PRINT SPC( 120)
230 VTAB 10: HTAB 8: PRINT "Name: ";: & X$,20
240 VTAB 16: HTAB 14: PRINT X$
```

GETTEXT

```
BSAVE GETTEXT, A$8000, L$0192
```

Für ProDOS ist Zeile 51 des Assemblerlistings zu ändern in:

```
                51   ADRESS   EQU  $BEB9
```

Es ergeben sich beim Assemblieren dann folgende Änderungen:

```
8020: AD B9 BE  56            LDA  ADRESS
8023: 69 03     57            ADC  #$03
8025: 90 03     58            BCC  AMPER1
8027: EE BA BE  59            INC  ADRESS+1
802A: 8D F6 03  60   AMPER1   STA  $3F6
802D: AD BA BE  61            LDA  ADRESS+1
```

```
                1   ********************************
                2   *                              *
                3   *    <<< GETTEXT >>>           *
                4   *                              *
                5   * Ein komfortabler BASIC-INPUT  *
                6   * für Applesoft-BASIC          *
                7   * (benutzt die modifizierte    *
                8   * INALL-Routine von H. Grumser) *
                9   * von Ulrich Kußmann, Bonn 1985 *
                10  * Aufruf über Ampersand-Vektor: *
                11  *                              *
                12  * z.B.:    & X$, 20            *
                13  *                              *
                14  ********************************
                15  *
                16  FORPNT   EQU  $85
                17  GDBUFS1  EQU  $D539
                18  STRCPY   EQU  $DA9A
                19  CHKSTR   EQU  $DD6C
                20  CHKCOM   EQU  $DEBE
                21  PTRGET   EQU  $DFE3
                22  ERRDIR   EQU  $E306
                23  STRLIT1  EQU  $E3E9
                24  GETBYT   EQU  $E6F8
                25  NUMBER   EQU  $EB
                26  *
8000: 18        27            CLC
8001: 90 17     28            BCC  AMPER
                29  *
                30  * => modifizierte INALL-Routine
                31  *
8003: 20 06 E3  32            JSR  ERRDIR
8006: 20 E3 DF  33            JSR  PTRGET
8009: 20 6C DD  34            JSR  CHKSTR
800C: 85 85     35            STA  FORPNT
800E: 84 86     36            STY  FORPNT+1
8010: 20 BE DE  37            JSR  CHKCOM
8013: 20 F8 E6  38            JSR  GETBYT
```

```
8016: 86 EB     39          STX  NUMBER
8018: F0 1D     40          BEQ  GETX       ;immer
                41  *
                42  * => Ampersand-Vektor setzen!
                43  *
                44  * ADRESS ist der Zeiger im
                45  * Betriebssystem auf die Start-
                46  * adresse des Programms:
                47  *
                48  *     DOS 3.3   =   $AA72
                49  *      ProDOS   =   $BEB9
                50  *    (BASIC.SYSTEM 1.0)
                51  ADRESS   EQU  $AA72
                52  *
801A: A9 4C     53  AMPER    LDA  #$4C       ;JMP
801C: 8D F5 03  54          STA  $3F5
801F: 18        55          CLC
8020: AD 72 AA  56          LDA  ADRESS
8023: 69 03     57          ADC  #$03
8025: 90 03     58          BCC  AMPER1
8027: EE 73 AA  59          INC  ADRESS+1
802A: 8D F6 03  60  AMPER1   STA  $3F6
802D: AD 73 AA  61          LDA  ADRESS+1
8030: 8D F7 03  62          STA  $3F7
8033: 60        63          RTS
                64  ********************************
                65  *        Modul: GETX           *
                66  ********************************
                67  *
                68  SIGN      EQU  $AE        ;","
                69  DELETE    EQU  $FF
                70  BUFFER    EQU  $0200
                71  IQERR     EQU  $E199
                72  COUT      EQU  $FDED
                73  BELL      EQU  $FF3A
                74  *
                75  * -> Interne Register:
                76  *
                77  ANZAHL    EQU  NUMBER     ;($EB)
                78  PTEND     EQU  $EC
                79  PTAKT     EQU  $ED
                80  ZEIGER    EQU  $71
                81  *
                82  * -> Pointer rücksetzen und Text
                83  *    aus der Variablen übernehmen,
                84  *    wenn vorhanden.
                85  *
8034: 4C 99 E1  86  IQERR1   JMP  IQERR
                87  *
8037: A0 00     88  GETX     LDY  #$00
8039: 84 ED     89          STY  PTAKT
803B: 84 EC     90          STY  PTEND
                91  *
803D: B1 85     92          LDA  (FORPNT),Y
803F: F0 30     93          BEQ  FELD
8041: C5 EB     94          CMP  ANZAHL
8043: F0 02     95          BEQ  OKAY
8045: B0 ED     96          BCS  IQERR1
8047: 85 EC     97  OKAY     STA  PTEND
8049: 85 ED     98          STA  PTAKT
                99  *
                100 * -> Variablenpointer auf Text im
                101 *    Speicher in Zero-Page poken
                102 *
804B: C8        103         INY
804C: B1 85     104         LDA  (FORPNT),Y
804E: 85 71     105         STA  ZEIGER
8050: C8        106         INY
8051: B1 85     107         LDA  (FORPNT),Y
8053: 85 72     108         STA  ZEIGER+1
                109 *
                110 * -> Text aus der Variablen in
                111 *    Puffer und auf Bildschirm
                112 *    bringen; dabei Bit 7 auf
                113 *    1 setzen
                114 *
8055: A0 00     115         LDY  #$00
8057: B1 71     116 MOVE    LDA  (ZEIGER),Y
8059: 09 80     117         ORA  #%10000000 ;Bit 7 = 1
805B: 99 00 02  118         STA  BUFFER,Y
805E: 20 ED FD  119         JSR  COUT
8061: C8        120         INY
8062: C4 EC     121         CPY  PTEND
8064: D0 F1     122         BNE  MOVE
                123 *
8066: C4 EB     124         CPY  ANZAHL     ;Feld
8068: D0 07     125         BNE  FELD       ;voll?
806A: A9 AE     126         LDA  #SIGN
806C: 99 00 02  127         STA  BUFFER,Y
806F: D0 1A     128         BNE  CUR
                129 *
```

```
                130 * -> Eingabefeld (Punkte) anlegen
                131 *
8071: A9 AE     132 FELD     LDA  #SIGN
8073: 99 00 02  133          STA  BUFFER,Y
8076: 20 ED FD  134          JSR  COUT
8079: C8        135          INY
807A: C4 EB     136          CPY  ANZAHL
807C: D0 F3     137          BNE  FELD
                138 *
                139 * -> Ausgabepunkt zurückstellen
                140 *
807E: 99 00 02  141          STA  BUFFER,Y
8081: A9 88     142          LDA  #$88       ;BS
8083: 20 ED FD  143 FELD1    JSR  COUT
8086: 88        144          DEY
8087: C4 EC     145          CPY  PTEND
8089: D0 F8     146          BNE  FELD1
                147 *
                148 * -> kein Cursor, wenn Zeilenende
                149 *    und Zeile voll
                150 *
808B: A4 ED     151 CUR      LDY  PTAKT
808D: C4 EB     152          CPY  ANZAHL
808F: D0 0B     153          BNE  CURSOR
                154 *
8091: AD 00 C0  155 CUR1     LDA  $C000
8094: 10 FB     156          BPL  CUR1
8096: AE 10 C0  157          LDX  $C010
8099: 18        158          CLC
809A: 90 3B     159          BCC  WAS        ;immer
                160 *
                161 * -> Cursor-Routine
                162 *
809C: A9 DF     163 CURSOR   LDA  #$DF       ;"_"
809E: 20 ED FD  164          JSR  COUT
80A1: A9 88     165          LDA  #$88       ;BS
80A3: 20 ED FD  166          JSR  COUT
80A6: 38        167          SEC
                168 *
                169 * Tastenabfrage und Warteschleife
                170 *
80A7: A2 00     171 KEYWAIT  LDX  #$00
80A9: A0 A0     172          LDY  #160       ;Frequenz
80AB: AD 00 C0  173 WARTEN   LDA  $C000
80AE: 30 18     174          BMI  TASTE
80B0: E8        175          INX
80B1: D0 F8     176          BNE  WARTEN
80B3: C8        177          INY
80B4: D0 F5     178          BNE  WARTEN
                179 *
80B6: 90 E4     180          BCC  CURSOR
80B8: A4 ED     181          LDY  PTAKT
80BA: B9 00 02  182          LDA  BUFFER,Y
80BD: 20 ED FD  183          JSR  COUT
80C0: A9 88     184          LDA  #$88       ;BS
80C2: 20 ED FD  185          JSR  COUT
80C5: 18        186          CLC
80C6: 90 DF     187          BCC  KEYWAIT    ;immer
                188 *
                189 * -> Taste gedrückt!
                190 *    Letzten Buchstaben drucken
                191 *
80C8: AA        192 TASTE    TAX             ;A retten
80C9: A4 ED     193          LDY  PTAKT
80CB: B9 00 02  194          LDA  BUFFER,Y
80CE: 20 ED FD  195          JSR  COUT
80D1: A9 88     196          LDA  #$88       ;BS
80D3: 20 ED FD  197          JSR  COUT
                198 *
                199 * -> Befehl oder Zeichen?
                200 *
80D6: 8A        201          TXA             ;A holen
80D7: AE 10 C0  202 WAS      LDX  $C010
80DA: C9 88     203          CMP  #136
80DC: F0 4B     204          BEQ  BACK
80DE: C9 8D     205          CMP  #141
80E0: F0 29     206          BEQ  RETURN
80E2: C9 95     207          CMP  #149
80E4: F0 35     208          BEQ  VOR
80E6: C9 9B     209          CMP  #155
80E8: F0 2D     210          BEQ  ESC
80EA: C9 FF     211          CMP  #DELETE
80EC: F0 1B     212          BEQ  DEL0
80EE: C9 A0     213          CMP  #160       ;Ctrl?
80F0: 90 99     214 CUR2     BCC  CUR
                215 *
                216 * -> Zeichen in Puffer speichern
                217 *    und auf Bildschirm ausgeben
                218 *
80F2: A6 EC     219          LDX  PTEND      ;Zeile
80F4: E4 EB     220          CPX  ANZAHL     ;voll?
```

```
80F6: F0 3F     221           BEQ  EOL
80F8: C4 EC     222           CPY  PTEND       ;Insert?
80FA: D0 40     223           BNE  INSERT
                224  *
80FC: 99 00 02  225           STA  BUFFER,Y    ;Append!
80FF: 20 ED FD  226           JSR  COUT
8102: E6 ED     227           INC  PTAKT
8104: E6 EC     228  INCR      INC  PTEND
8106: 18        229  CUR3      CLC
8107: 90 E7     230           BCC  CUR2        ;immer
8109: F0 68     231  DEL0      BEQ  DEL         ;Sprung
                232  *
                233  * -> Return
                234  *
810B: A6 EC     235  RETURN    LDX  PTEND
810D: 20 39 D5  236  RETURN1   JSR  GDBUFS1    ;INALL
8110: C8        237           INY              ;2. Teil
8111: 20 E9 E3  238           JSR  STRLIT1
8114: 4C 9A DA  239           JMP  STRCPY      ;Exit
                240  *
                241  * -> Escape
                242  *
8117: A2 00     243  ESC       LDX  #0
8119: F0 F2     244           BEQ  RETURN1
                245  *
                246  * -> Cursor vorrücken
                247  *
811B: C4 EC     248  VOR       CPY  PTEND       ;Zeilen-
811D: F0 E7     249           BEQ  CUR3        ;ende?
                250  *
811F: B9 00 02  251           LDA  BUFFER,Y
8122: 20 ED FD  252           JSR  COUT
8125: E6 ED     253           INC  PTAKT
8127: D0 DD     254           BNE  CUR3        ;immer
                255  *
                256  * -> Backspace
                257  *
8129: C0 00     258  BACK      CPY  #$00        ;Zeilen-
812B: F0 0A     259           BEQ  EOL         ;anfang?
                260  *
812D: A9 88     261           LDA  #$88        ;BS
812F: 20 ED FD  262           JSR  COUT
8132: C6 ED     263           DEC  PTAKT
8134: 18        264           CLC
8135: 90 CF     265           BCC  CUR3        ;immer
                266  *
                267  * -> Anzahl voll (Warnung)!
                268  *
8137: 20 3A FF  269  EOL       JSR  BELL
813A: D0 CA     270           BNE  CUR3
                271  *
                272  * -> Text einfügen wenn:
                273  *    PTAKT <> PTEND
                274  *
813C: AA        275  INSERT    TAX              ;A retten
813D: A4 EC     276           LDY  PTEND
813F: 88        277  MOVEUP    DEY
8140: B9 00 02  278           LDA  BUFFER,Y
8143: 99 01 02  279           STA  BUFFER+1,Y
8146: C4 ED     280           CPY  PTAKT
8148: D0 F5     281           BNE  MOVEUP
                282  *
814A: 8A        283           TXA              ;A holen
814B: 99 00 02  284           STA  BUFFER,Y
814E: 20 ED FD  285           JSR  COUT
8151: E6 ED     286           INC  PTAKT
8153: 18        287           CLC
                288  *
                289  * -> veränderten Pufferinhalt
                290  *    ausgeben
                291  *
8154: A4 ED     292  UPDATE    LDY  PTAKT
8156: 08        293           PHP
8157: B9 00 02  294  UPDAT1    LDA  BUFFER,Y    ;Puffer-
815A: 20 ED FD  295           JSR  COUT        ;inhalt
815D: C8        296           INY              ;ab PTAKT
815E: C4 EB     297           CPY  ANZAHL      ;aus-
8160: D0 F5     298           BNE  UPDAT1      ;geben
                299  *
8162: A9 88     300           LDA  #$88        ;BS
8164: 20 ED FD  301  PRINTB    JSR  COUT        ;Ausgabe
8167: 88        302           DEY              ;zurück-
8168: C4 ED     303           CPY  PTAKT       ;stellen
816A: D0 F8     304           BNE  PRINTB
                305  *
816C: 28        306           PLP
816D: 90 95     307           BCC  INCR
816F: C6 EC     308           DEC  PTEND
8171: B0 93     309           BCS  CUR3        ;immer
                310  *
                311  * -> Delete
                312  *
8173: C0 00     313  DEL       CPY  #$00        ;Zeilen-
8175: F0 8F     314           BEQ  CUR3        ;anfang?
                315  *
8177: C6 ED     316           DEC  PTAKT
8179: 88        317           DEY
817A: B9 01 02  318  DEL1      LDA  BUFFER+1,Y  ;Puffer-
817D: 99 00 02  319           STA  BUFFER,Y    ;inhalt
8180: C8        320           INY              ;ver-
8181: C4 EC     321           CPY  PTEND       ;schieben
8183: D0 F5     322           BNE  DEL1
                323  *
8185: A9 AE     324           LDA  #SIGN
8187: 99 00 02  325           STA  BUFFER,Y
818A: A9 88     326           LDA  #$88        ;BS
818C: 20 ED FD  327           JSR  COUT
818F: 38        328           SEC
8190: B0 C2     329           BCS  UPDATE      ;immer

402 Bytes
```

Computerbücher
die gehen, für Computer die kommen.

Arne Schäpers
ProDOS-Analyse
Versionen 1.0.1, 1.0.2, 1.1.1
1985, 320 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

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
2., durchges. Auflage 1984,
216 S., kart., DM 28,—
ISBN 3-7785-1049-5

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

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

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
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

PEEKER Börse

Verkauf Software

Astrologie: Berechn. u. Graphik. Info n. voreins. 1 DM in Briefm., C. Landscheidt, Im Dorfe 14, D-2804 Lilienthal.

Computer-Literatur:
Liste bei: **Lindemanns Buchhandl.** kostenlos.
Nadlerstr. 10, 7000 Stuttgart-1

********* **WARGAMES** *********

Die Besten von SSI für APPLE C 64 ATARI.
Info gegen 80 Pfg. RP

Computer-Service Th. Müller
Postfach 2526
7600 Offenburg

– – STOCKMASTER II – – –
Das Apple-Programm für echte Börsengewinne. Diskette 485,– DM. Beschreibung P12 anfordern bei Töngi, COMPUTER-PRAXIS, Aspeltstr. 4, D-6500 Mainz 1

PIRATE DEFENCE 2.0 Kopierschutz gehört zu den sichersten Schutzsystemen Deutschlands. Für DOS, ProDOS, DIVERSI u.a. Info (50 Pf), Chr. Bregler, Tulpenstr. 2, 7519 Eppingen. Händleranfragen erw.!

Software-Uhr f. Apple II+ e, c ausbaufähig zum Zeitschalter Diskette + Anleitung DM 25,– Dortmund, T. 39 19 20

Diverse Software für Apple IIe zu verkaufen. Nur Originale. Liste gegen freien Rückumschlag.
Michael Sprengel
Amselweg 4
7903 Laichingen 3

Apple: Public Domain: Volume DM 15, Games, Schach, Graphic etc. Derw. Lehrerpro. Gratisinfo: Fa. Waltrau. Muhle Waldwinkel 3, 2105 Seevetal 3

Apple Pascal 1-2 u. Apple Works! Original! Hälfte NP! C. v. Fournier, Fasanenstr. 4, 8550 Forchheim (2259)

Apple II emuliert Sharp-Pockets! Prgrübertrg. v. Gamep. z. 11-Pincon. Info geg. 2×80 Pfg-Brfmk., K. Schmidt, Hasencleverstraße 25, 2000 Hamburg 74

Verkauf Hardware

Apple-SUPERANGEBOT

Apple II

PAL-Farbinterface	85,–
A2 Language Karte 16KB	120,–
Apple Pilot, Sprache	97,–
Quickfile IIe (deutsch)	59,–
6502 Hardware Manual	14,–
6502 Programmier Manual	14,–
Stand f. Monitor III	19,–

Apple III

Apple III 256 KB	2330,–
Profile 5 MB Laufw.	1420,–
Silentype Drucker	235,–
Silentype Papier (15)	47,–
Parallel Interface	150,–
Visicalc III	160,–
Mail List Manager	99,–
Hand Holding Basic	60,–
Apple Writer III	150,–
Business Graphics A III	105,–
Access III	90,–
Pascal III Utility Lab.	45,–

Weiterhin bieten wir an:
INTRO 8-Zoll-Diskettenlaufwerk für Apple III, kompl. 1256,–
Täglich von 9.00 bis 16.00 Uhr
Tel.: 02 28-268 42 96 u. 268 42 13
Bonndata GMBH, Poppelsdorfer Allee 27-33, 5300 Bonn 1

Verkaufe Apple Zusatzkarten:

16K, Z80 Karten	75,– DM
80 Zeichen Karten	180,– DM
128KB Karten	205,– DM
ERPHI CONTROLLER nur	278,– DM
TEAC FD-55-FV nur	398,– DM
IIe Kompatibler nur	1710,– DM

auch IBM-Karten / Infos kostenlos
Axel Braukmann, Am Sportplatz 51
4005 Meerbusch-2,
Tel. 0 21 59 / 36 26

Verk. w. Syst. w. neuw. Applekomp. IBS-Space-84, 192 kB RAM, Knürrgeh., Lüft., sep. Tast., Taxan Mon. gn., Z-80 Card, Z-80B Card (IBS AP-22, 64 kB, 6 MHz), Floppy-Contr., Ehring-Contr., Epson Drukkerinterf., 80-Z. Videx Ultraterm, orig. The Clock, Supertalker (dig. Sprachein- u. ausg.), 1 Floppy 135 kB, 2 Teac FD 55 A (140 kB), 2 Teac FD 55 F (640 kB), orig. Apple Graphik-Tabl. Tel. 0 80 84 / 36 27 nach 18 Uhr.

Apple II + 64K Disk Minotor VIDEX Z80 PAL Div. Softw. CP/M Finanz Wirtsch, Spiele Literatur Handbü. DM 3000,–
Tel.: 030/4045247

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

Verk. Apple IIe, 64K RAM, mit Bildschirm, Disk. Laufwerk, Matrixdrucker, Z80-Karte, CP/M-Karte, Grafikf. mit umfangreicher Software (DB2, Wordstar uva.). VP DM 5.500, Tel. 0 62 03 / 6 57 30.

3 Drucker an einen Computer? 3 Computer an einen Drukker? Info: Fricke Computertechnik, Wattstraße 30, 2400 Lübeck, Tel. 04 51 / 60 47 49

Epson Interfaceumbau f. AWorks DM 20. Mahr, Waldakker 71, 73 Esslingen

Apple IIe, 2 Disk II mit Contr., 128 KB. 80-Zeichen Card, Super Serie II Card, Printer Card, Z80 Card (CPM), Joystick, DM 3600. 2 × 640 KB TEAC FD55 Drives incl. Contr. im Gehäuse anschlußfertig DM 1300. Alles zus. nur DM 4600. Tel. 0 71 81 / 6 15 40 ab 19 Uhr.

Apple II+ 64K, Z80, 80Z, 2 Floppy, Monitor, Drucker, viel Software, Lit., DM 2600,–
Tel. 0 71 51 / 6 15 84

Apple IIe, 1 Disk+Contr., Monitor, Prem. Softc. (Z80B, CP/M, 80ZZ, + 64K) für DM 3000,–
T. 0 82 31 / 8 56 24

Ramworks-Karte 512KRAM DM 1500,– / sFr 1250,–. Mit AW-Patch für Apple IIe. Z-RAM 512 KRAM+CP/M DM 2300,– –/sFr 1950,– Für Apple IIc. Weiteres div. Zubehör Liste Verl.
Tel. Schweiz 061/47 89 34.

Verk. Apple IIe Comp. 128KB Ram Z80-Prozessor, 80 Zeichen, Joyst. 2 Laufwerke 35/40, Monitor, sep. Tastatur *** VHB 3300,– ***
Michael Knöpel, Tel. 0 62 41/61 22

ZUFALLGENERATORkarte für Apple II. Echte Zufallzahlen durch Halbleiterrauschen. DM 98,– incl. Versk. per NN. Ralf Pagel, Schönberger Str. 14, 2300 Kiel, T. 04 31 / 72 96 31

MAC 128 auf 512K DM 798–Bausatz kompl. DM 298,–, Tel. 18.00 h, T. 089 / 98 58 98

Apple IIe mit Bildschirm + Lüfter 128K Z80 1 Teak Doppellaufwerk a. 360 K umschaltbar 40/80 Tr. VB DM 3500,–, Tel. 040 / 601 00 51

Apple II+, 64K, 20rig. Disks, Monitor, 80Z-K., Z80-K., komplett oder einzeln, Preis VHS,
T. 0 62 21 / 86 16 33

Erphi-Contr. 198,–,
T. 07 11 / 42 43 25

ERPHI-SYSTEM Erphi-Controller 275,–, 2 × TEAC FD 55 F (2 × 80 Sp) à 460,– oder komplett 1150,– (alles neuw.) Schoeben Udo, Tel. 08 21 / 15 23 77

Akustik-Koppler, Dataphon s21d Anschlußkabel, Terminalprogramm neu, DM 360,–,
Tel. 0 26 66 / 704

Original Apple 80Z/Z-Karte f. IIe m. Buch DM 120,–
Tel. 0 71 59 / 36 02

Ankauf Software

Finanzbuchhaltungs-, Auftragsabwicklungs- und Lohnbuchhaltungs-Programme für den Apple II+ gesucht. T. 05164/8566

Ankauf Hardware

Suche Apple-Interface für Schreibmaschine TA 8008 Gabriele. Wer kann helfen?
Tel. 04 61 / 2 59 99

Suche Interface (RGB II B-80) für TAXAN-Monitor. M. Rosbund, Eierwiesenstr. 62, 7024 Filderstadt 1

Kontakte

Wer kann mir bei Problemen mit CP/M helfen? Gernot Widmann, T. 0 71 93 / 62 86 ab 18 Uhr.

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

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 ..... (1985 erscheinen 11 Ausgaben – 1 Doppelnummer) zum Jahresbezugspreis von DM 72,– (Inland) incl. MwSt. Die Lieferung erfolgt frei Haus. Porto, Verpackung und Zustellgebühren übernimmt der Verlag. Der Jahresbezugspreis für das Ausland beträgt DM 72,– incl. MwSt., zzgl. DM 16,80 Versandspesen.

Ich wünsche jährliche Berechnung durch:

☐ Verlagsrechnung ☐ Abbuchung von meinem Bank- bzw. Postscheckkonto

_____
Bank / PschA

_____
Bankleitzahl Kto.-Nr.

_____
Datum Unterschrift

PEEKER

Buch-Shop

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 Apple-Programme:

☐ 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,–

☐ CP/M ja ☐ CP/M nein
☐ Pascal ja ☐ Pascal nein

☐ Apple DOS 3.3, Begleitdiskette, DM 28,–
☐ Apple ProDOS, Band 1, Begleitdiskette, DM 28,–
☐ Apple 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,–

_____
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 7 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-Shop

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

Postfach 10 28 69

6900 Heidelberg 1

POSTKARTE

peeker
Versandbuchhandlung

Postfach 10 28 69

6900 Heidelberg 1

POSTKARTE

peeker
Softwareabteilung

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

PEEKER Börse

Peeker-Termine

Verschiedenes

Es geht doch: APPLEWORKS auf II+ u. Kompatiblen. Info: Fa. Paul Arping, T. 04 31 / 73 71 35

Börsen-Programme für Apple II gesucht. Angebote an Nortmann, Schaperstr. 24, 6200 Wiesbaden, T. 0 61 21 - 52 78 20

Horoskop-Programme für Apple II gesucht. Angebote an Nortmann, Schaperstr. 24, D-6200 Wiesbaden, T. 0 61 21 - 52 78 20

Apple 256-KB-RAM-Floppy 349,– DM, 128-KB-Karte 249,– DM mit Softw. Accelerator-Karte 599,– DM, TFM II Supermodem 300, 1200, 1200/75 Bd. nur 478,– DM – FUCOM GmbH, Postf. 483, 4600 Dortmund 1

Hard- und Software für Apple-Computer gesucht. Wer schreibt gute Software? Auch Gebrauchtes angenehm. Zahle Höchstpreise! Suche noch Leute, die nebenbei verdienen wollen. Zuschriften Chiffre P1005

** Neue Bücher für den Apple II **

Elektronik- und Graphik-Programme
Die Programm-Fundgrube zu folgenden Themen: Statistik, Filter, Netzwerke, Laplace- und Fourier-Transf., Komplexe Rechnung, Diagramme usw., zahlr. Abb., 184 S., 16,5 × 23,5 cm, DM 39,80

Apple II – leicht programmiert
Der Wegweiser zum eigenen Programm: Zählen, Schleifen, Strings, Eingabe, Druckformatierung, Adressverw. usw., 96 S., 16,5 × 23,5 cm, DM 22,80

BASIC-Tricks für den Apple II
Die Trickkiste für die professionelle Programmgestaltung: Eingabe, Menüs, Sortieren, Listengestaltung, Fehlererkennung/Korrektur, Dateien usw., 144 S., 16,5 × 23,5 cm, DM 29,80

Bei Bestellung bitte DM 2,50 für Porto und Verpackung addieren.

beam-Verlag,
Pf. 1148P, 355 Marburg

****

Ausgabe	Anzeigenschluß	Erstverkaufstag
1	22. 11. 85	23. 12. 85
2	20. 12. 85	20. 01. 86
3	17. 1. 86	17. 2. 86
4	21. 2. 86	24. 3. 86
5	21. 3. 86	21. 4. 86
6	18. 4. 86	20. 5. 86
7	23. 5. 86	23. 6. 86
8	20. 6. 86	21. 7. 86
9	18. 7. 86	18. 8. 86
10	22. 8. 86	22. 9. 86
11	19. 9. 86	20. 10. 86
12	17. 10. 86	17. 11. 86

Tabelle der wichtigen Applesoft-Interpreter-Routinen

von Harald Grumser

Der Applesoft-Interpreter bietet eine Fülle von Routinen, die z.T. über recht unkomplizierte Schnittstellen verfügen oder auch nur durch „JSR" oder „CALL" aufgerufen werden können. Die im folgenden zusammengestellte Tabelle umfaßt solche Unterprogramme und Routinen, die mit sinnvollem und vertretbarem Aufwand in eigene Programme aufgenommen werden können und erhebt daher auch nicht den Anspruch auf Vollständigkeit, da selbstverständlich eine subjektive Auswahl getroffen wurde.
Voraussetzung für das genaue Verständnis *aller* hier aufgeführten Routinen ist die Kenntnis der Arbeitsweise des Interpreters, die hier nicht vermittelt werden kann. Diese Aufstellung soll aber auch helfen, tiefere Einblicke zu gewinnen.

Einige der erwähnten Routinen scheinen sehr beschränkte Anwendungsmöglichkeiten zu bieten, doch würde eine engere Vorauswahl den Programmierern nicht gerecht werden, die in der Lage sind, durch geeignete Hilfsmittel (Intercept-Programmierung, CHRGET-Manipulation) auch umfangreichere Applikationen zu erstellen.
Obwohl die Ausführungen der einzelnen Applesoft-Befehle (im folgenden wird Applesoft durch A/S abgekürzt) nicht immer für eigene Aufrufe geeignet sind, wurden einige in die Tabelle aufgenommen, um auf etwaige Besonderheiten aufmerksam zu machen.
Eine detaillierte Schilderung aller Routinen würde ein Buch füllen. Daher wurden im wesentlichen nur die Parameter aufgeführt, die im Zusammenhang mit dem entsprechenden Aufruf von Bedeutung sind. Es sollte sich von selbst verstehen, daß z.B. zur Durchsuchung eines Programms nach einer bestimmten Zeilennummer die entsprechenden Programmzeiger gesetzt sein müssen.
Die Tabelle ist nach Sachgebieten gegliedert, um ein einfacheres Auffinden der benötigten Routinen zu erleichtern. Diese Zuordnung kann jedoch nicht immer eindeutig festgelegt werden und sollte nur als Anhaltspunkt verstanden werden.

Die Label-Namen entstammen den Bezeichnungen, die schon W.F. Luebbert in seinem Buch „What's Where in the Apple" benutzte und stützen sich darüber hinaus auf das kommentierte Assemblerlisting von G. Bredon. Einige wenige Labels sind anderen Quellen entnommen.

Alle Angaben beziehen sich auf den alten A/S-Interpreter, wie er im II+ und IIe implementiert wurde. Abweichungen des Interpreters des IIc und des neuen IIe (Enhanced IIe) werden gesondert angegeben. (Siehe auch „Die neuen ROMs für den Apple IIe", Teil 3, Peeker 10/85, S. 34.)

Bei der Erklärung der einzelnen Routinen sind im Text z.T. die symbolischen Labels aufgeführt, deren Adresse unter den Rubriken stets aufgeführt sind (mit Ausnahme von TXTPTR = $00B8, $00B9), um langes Suchen zu ersparen. Wegen der sehr unterschiedlichen Nutzung des Hauptfließkomma-Akkus (FAC) wurde dieser stets aufgeführt. Die Adressenangaben beziehen sich dabei auf die in dieser Routine relevanten Abschnitte von FAC. Da der Textpointer bei der Auswertung eines Befehls immer weitergerückt wird, ist dies nicht besonders vermerkt. Wenn nicht anders angegeben, enden die Routinen mit RTS.
In der **Tabelle der Zero-Page-Variablen** sind alle Adressen aufgeführt, die in dieser Aufstellung benutzt werden. Diese Tabelle bietet weitergehende Informationen über die Bedeutung der einzelnen Speicherstellen.

– Unter der Rubrik *Vorher* sind alle Eingabeparameter zusammengestellt, die vor Aufruf entsprechend gesetzt werden müssen. Dabei bedeutet
A = Akkumulator,
X = X-Register,
Y = Y-Register,
S = Stackpointer,
Z = Zero-Flag,
C = Carry-Flag.

Ist diese Rubrik nicht angegeben, so kann der Aufruf durch einfaches „JSR" erfolgen. Muß beim Aufruf *kein* Prozessorregister gesetzt werden, so eignet sich diese Routine auch für einen „CALL"-Befehl aus A/S. Die dezimale Adresse entnehme man der **Tabelle der aufgeführten Routinen**.
– Die Rubrik *Aufruf* deutet auf eine Besonderheit bei der Benutzung hin, wie z.B. ein erforderlicher vorangehender „JSR CHRGET"-Befehl.
– Unter *Nachher* werden die Ausgabeparameter aufgeführt, die als Resultat erwünscht sind. Änderungen anderer Variablen werden hier nicht berücksichtigt. Bei einigen Routinen ist wichtig zu wissen, welche Register erhalten bleiben. In diesem Fall wird z.B. der unveränderte Inhalt des Akkus durch „A =" gekennzeichnet.

Wird im Text auf eine andere aufgeführte Routine verwiesen, so ist dieser ein „↑" vorangestellt. Referenzen auf A/S-Befehle sind durch Anführungszeichen gekennzeichnet.

Gliederungsübersicht

1. Programmsteuerung

1.1. Applesoft-Befehle

1.1.1. Einfache Befehle

RUN ($D566) – führt das A/S-Programm im Speicher aus (entspricht nicht dem „RUN"-Befehl, da kein konditioniertes RUN mit Zeilenangabe möglich ist).

STOP ($D86E) – führt „STOP" aus. Dabei werden TXTPTR und CURLIN für „CONT" gerettet. Danach erfolgt unabhängig von „ONERR" eine „BREAK"-Meldung und ein Warmstart (↑RESTART).
Vorher: C = 1; Z = 1.
Nachher: OLDTXT ($0079, $007A) = TXTPTR; OLDLIN ($0077, $0078) = CURLIN ($0075, $0076).

END ($D870) – beendet das Programm und springt nach ↑RESTART. Dies entspricht ↑STOP, es wird jedoch keine Meldung ausgegeben.
Vorher: Z = 1.
Nachher: siehe ↑STOP.

CONT ($D896) – versucht das unterbrochene Programm fortzuführen. Ist dies nicht möglich (z.B. nach Änderung des Programms oder „CLEAR"), erfolgt ein Sprung nach ↑ERROR.
Vorher: OLDTXT ($0079, $007A) = Zeiger auf nächsten A/S-Befehl - 1; OLDLIN ($0077, $0078) = Zeilennummer des nächsten Befehls.
Aufruf: JSR ↑CONT, JMP ↑NEWSTT.
Nachher: ERRNUM ($00DE) = $D2, falls kein „CONT" möglich.

TRACE ($F26D) – setzt das Trace-Flag.
Nachher: TRCFLG ($00F2) >= $80.

NOTRACE ($F26F) – löscht das Trace-Flag.
Nachher: TRCFLG ($00F2) < $80.

NORMAL ($F273) – setzt den Normal-Ausgabemodus.
Nachher: INVFLG ($0032) = $FF; ORMASK ($00F3) = $00.

INVERSE ($F277) – setzt den Invers-Ausgabemodus.
Nachher: INVFLG ($0032) = $3F; ORMASK ($00F3) = $00.

FLASH ($F280) – setzt den Flash-Ausgabemodus.
Nachher: INVFLG ($0032) = $7F; ORMASK ($00F3) = $40.

RESTORE ($D849) – setzt den DATA-Zeiger auf den Programmanfang - 1.
Nachher: DATPTR ($007D, $007E) = TXTTAB ($0067, $0068) - 1.

LOAD ($D8C9) – lädt ein A/S-Programm via READ ($FEFD) von Kassette. Dabei wird auch das LOCK-Byte eingelesen. Diese Routine entfällt beim IIc.

SAVE ($D8B0) – sichert das A/S-Programm via WRITE ($FECD) auf Kassette. Dabei wird zunächst die Programmlänge, dann das LOCK-Byte und zuletzt das Programm gespeichert. Diese Routine entfällt beim IIc.
Vorher: LOCK ($00D6) = $80, um Programm bei jeder Eingabe zu starten.

STORE ($F39F) – führt „STORE" aus mittels Monitor-Routine WRITE ($FECD). Diese Routine entfällt beim IIc.

RECALL ($F3BC) – führt „RECALL" aus mittels Monitor-Routine READ ($FEFD). Diese Routine entfällt beim IIc.

SHLOAD ($F775) – liest eine Shape-Tabelle mittels der Monitor-Routine READ ($FEFD) von Kasette unterhalb von HIMEM ein und setzt die entsprechenden Zeiger. Diese Routine entfällt beim IIc.
Nachher: MEMSIZE ($0073, $0074) = FRETOP ($006F, $0070) = SHPTAB ($00E8, $00E9) = HIMEM - Tabellenlänge.

1.1.2. Befehle mit Parametern

RUNLINE ($D91B) – führt ein konditioniertes „RUN" aus (entspricht „CLEAR : GOTO n"). Nach dem Aufruf von ↑CLEARC erfolgt ein Sprung zu ↑GOLINE.
Vorher: siehe ↑GOLINE.

LET ($DA46) – weist der folgenden Variablen den nach dem „="-Zeichen angegebenen Ausdruck zu. Stimmen die beiden Typen nicht überein, erfolgt ↑TMMERR. Diese Routine ruft ↑PTRGET und ↑FRMEVL auf und benutzt bei String-Variablen einige String-Routinen.
Vorher: TXTPTR = Zeiger auf ersten Buchstaben des Variablennamens.
Nachher: FORPNT ($0085, $0086) = Zeiger auf vorgefundene oder neu angelegte Variable; ansonsten siehe ↑FRMEVL.

LIST ($D6A5) – führt „LIST" aus und springt nach ↑NEWSTT (siehe ↑MAINLIST).
Vorher: TXTPTR = Zeiger auf erstes Zeichen der Bereichsangabe - 1 (z.B. „100, 200") mit Bit 7 = 0 und $00 oder „:" als Endmarker.
Aufruf: Register durch JSR ↑CHRGET setzen.

DEL ($F331) – löscht den angegebenen Zeilenbereich und alle Variablen (entspricht „DEL").
Vorher: TXTPTR = Zeiger auf 1. Zeichen vor Zeilenbereichsangabe.
Aufruf: Register durch JSR ↑CHRGET setzen.

GOTO ($D93E) – setzt die Programmzeiger auf die angegebene Zeile, indem das Programm nach dieser durchsucht wird. Die Suche erfolgt abhängig von der aktuellen Zeile nach oben bis zum Programmende oder vom Programmanfang aus. Wird die Zeile nicht gefunden, so verzweigt die Routine zu ↑UNDERR.
Vorher: TXTPTR = Zeiger auf Zeilenangabe - 1.
Aufruf: Register durch JSR ↑CHRGET setzen. Die Ausführung der neuen Zeile erfolgt erst nach Aufruf von ↑NEWSTT.
Nachher: TXTPTR = Zeiger auf neue Zeile; LINNUM ($0050, $0051) = neue Zeilennummer.

GOSUB ($D921) – führt „GOSUB" aus. Diese Routine überprüft zunächst mittels ↑CHKSTK,

ob der Stack noch 6 Bytes aufnehmen kann. Nach dem Ablegen des TXTPTR, der aktuellen Zeilennummer und dem „GOSUB"-Token ($B0) geht sie dann in ↑ GOLINE über.
Vorher: siehe ↑ GOLINE.

ONGOTO ($D9EC) – führt „ON I GOTO (GOSUB)" aus. Dabei wird zunächst der Index I mittels ↑ GETBYT eingelesen und der TXTPTR auf die I. Zeilenzahl gesetzt, um dann mit dem „GOTO"- oder „GOSUB"-Token im Akku nach ↑ GOCMD zu springen.
Vorher: TXTPTR = Zeiger auf Index.

POP ($D96B) – holt einen „GOSUB"-Eintrag vom Stack. Dabei werden alle in diesem Unterprogramm geöffneten „FOR-NEXT"-Schleifen abgeschlossen. Wurde kein Unterprogramm aufgerufen (kein „GOSUB"-Token auf dem Stack), verzweigt das Programm zu ↑ RWGERR. Enthält das Y-Register den Wert $42, wird „POP" durchgeführt und die Programmzeiger belassen.
Im anderen Fall werden die Programmzeiger vom Stack geholt und eingesetzt. Die Routine geht dann in ↑ DATA über, um die Zeiger auf das Ende des (ursprünglichen) Befehls zu setzen.
Vorher: Y = $42 führt „POP" aus, ansonsten „RETURN".
Aufruf: Die Ausführung des nächsten A/S-Befehls fährt erst nach Aufruf von ↑ NEWSTT fort.
Nachher: TXTPTR = Zeiger auf neuen Befehl (nur bei „RETURN"); CURLIN ($0075, $0076) = neue Zeilenummer (nur bei „RETURN").

IF ($D9C9) – wertet den nachfolgenden Ausdruck aus und prüft, ob „THEN" oder „GOTO" folgt, um ggf. nach ↑ SYNERR zu springen. Ist der Ausdruck falsch, wird ↑ REM ausgeführt, um den Rest der Zeile zu überspringen. Ist er richtig, wird bei nachfolgender Zahl zu ↑ GOTO gesprungen, ansonsten zu ↑ GOCMD.
Vorher: TXTPTR = Zeiger auf logischen Ausdruck.

HIMEM1 ($F28C) – setzt den übergebenen Wert als neuen HIMEM-Wert ein. Falls dieser kleiner als das Ende der Variablen ist, erfolgt ↑ MEMERR. Eine Überprüfung für die Obergrenze findet nicht statt.
Vorher: LINNUM ($0050, $0051) = einzusetzender HIMEM-Wert.
Nachher: MEMSIZE ($0073, $0074) = FRETOP ($006F, $0070) = neuer HIMEM-Wert.

LOMEM1 ($F2AC) – setzt den übergebenen Wert als neuen LOMEM-Wert ein. Falls dieser größer als „HIMEM" und kleiner als der aktuelle LOMEM-Wert ist, erfolgt ↑ MEMERR. Durch einen Sprung nach ↑ CLEARC werden alle Variablen gelöscht.
Vorher: LINNUM ($0050, $0051) = einzusetzender LOMEM-Wert.
Nachher: siehe ↑ CLEARC.

1.2. Manipulation des Textpointers

DATA ($D995) – setzt TXTPTR auf das Ende des aktuellen Befehls, d. h. auf „:" oder $00.
Nachher: TXTPTR = Zeiger auf Befehlsende.

DATAN ($D9A3) – zählt die Zeichen ab TXTPTR bis zum nächsten „:" (nicht in Hochkomma) oder $00.
Vorher: TXTPTR = Zeiger auf Programmtext.
Nachher: Y = Anzahl der Zeichen.

REM ($D9DC) – setzt TXTPTR auf das Ende der aktuellen Zeile, d. h. auf $00.
Nachher: TXTPTR = Zeiger auf Zeilenende.

REMN ($D9A6) – zählt die Zeichen ab TXTPTR bis zum nächsten $00.
Vorher: siehe ↑ DATAN.
Nachher: Y = Anzahl der Zeichen.

ADDON ($D998) – setzt TXTPTR um die angegebene Anzahl von Zeichen weiter.
Vorher: Y = Anzahl der Zeichen.
Nachher: TXTPTR = TXTPTR + Y.

STRTXT ($DE81) – übergeht ein Zeichen (i.allg. Hochkomma), führt ↑ STRLIT aus und setzt den Textpointer hinter den String.
Vorher: C = 1; TXTPTR = Zeiger auf letztes Zeichen vor eigentlichem String.
Nachher: TXTPTR = Zeiger auf erstes Zeichen nach String; ansonsten siehe ↑ STRLIT.

1.3. Auswertung

CHRGET ($00B1) – liest das nächste Zeichen des Programmtextes in den Akku und überspringt Leerzeichen. Diese Routine wird beim Kaltstart in die Zero-Page kopiert.
Vorher: TXTPTR = Zeiger auf letztes eingelesenes Zeichen.
Nachher: TXTPTR = TXTPTR + 1 (falls kein Leerzeichen); Z = 1, wenn End-of-Statement-Zeichen ($00, „:"); C = 1, wenn Ziffer ($30-$39); A = eingelesenes Zeichen.

CHRGOT ($00B7) – liest das Zeichen, auf das TXTPTR verweist, in den Akku und überspringt Leerzeichen.
Vorher: TXTPTR = Zeiger auf einzulesendes Zeichen.
Nachher: siehe ↑ CHRGET.

FRMEVL ($DD7B) – wertet den durch den Textpointer bezeichneten Ausdruck (auch Vergleiche) aus, bis ein EOS („:" oder $00) folgt. Diese Routine stellt ein Kernstück des Interpreters dar und benutzt eine große Anzahl von Unterprogrammen. Der Ausdruck wird im Hauptfließkomma-Akku abgelegt, entweder als Deskriptor (Länge und Adresse des Strings) oder als Fließkommazahl (1 Byte Exponent, 4 Bytes Mantisse, ungepackte Form).
Vorher: TXTPTR = Zeiger auf 1. Zeichen des Ausdrucks.
Nachher: FAC ($009D-$00A1) = Fließkommazahl, bei num. Ausdruck; FACSIGN ($00A2) = Vorzeichen, bei num. Ausdruck; FAC ($009D-$009F) = Deskriptor, bei String; VALTYP ($0011) = Flag für Typ ($00 = Zahl, $FF = String).

FRMNUM ($DD67) – wertet den durch den Textpointer bezeichneten Ausdruck via ↑ FRMEVL aus. Liegt kein numerischer Ausdruck vor, erfolgt die Meldung „TYP MISMATCH".
Vorher: siehe ↑ FRMEVL.
Nachher: siehe ↑ FRMEVL.

GETVAL ($DE60) – wertet einen durch den Textpointer bezeichneten Ausdruck aus. Diese Routine ist das wichtigste Werkzeug von ↑ FRMEVL. Zu Beginn der Routine wird festgelegt, ob es sich um einen der folgenden Typen handelt: Variablenname, Zahl als ASCII-Folge (Bit 7 = 0), String in Hochkomma, Minuszeichen, NOT-Befehl, FN-Befehl, Funktionsausdruck. Ansonsten wird eine geöffnete Klammer erwartet und ↑ FRMEVL rekursiv aufgerufen. Hier ist das Zeichen zur Erkennung des Ausdrucksendes vom Typ abhängig.
Vorher: TXTPTR = Zeiger auf letztes Zeichen vor Ausdruck.
Nachher: siehe ↑ FRMEVL.

PARCHK ($DEB2) – ruft ↑ CHKOPN auf und springt nach ↑ FRMEVL.
Vorher: siehe ↑ SYNCHR.
Nachher: siehe ↑ FRMEVL.

GETBYT ($E6F8) – wertet den folgenden Ausdruck als Integer zwischen 0 und 255 aus. Stimmen Typ oder Wertebereich nicht überein, erfolgt eine Fehlermeldung.
Vorher: TXTPTR = Zeiger auf erstes Zeichen des Ausdrucks.
Nachher: X = FAC+4 ($00A1) = ausgewerteter Ausdruck; A = erstes Zeichen nach Ausdruck.

GTBYTC ($E6F5) – ruft ↑ CHRGET auf und geht in ↑ GETBYT über.
Vorher: TXTPTR = Zeiger auf auszuwertenden Ausdruck - 1.
Nachher: siehe ↑ GETBYT.

GETNUM ($E746) – wertet die zwei folgenden Ausdrücke als Zahlen aus, die durchein Komma getrennt sein müssen. Die erste Zahl darf im Bereich 0-65535 liegen, die zweite zwischen 0 und 255.
Vorher: TXTPTR = Zeiger auf erstes Zeichen des ersten Ausdrucks.
Nachher: LINNUM ($0050, $0051) = erste Zahl; X = FAC+4 ($00A1) = zweite Zahl.

LINGET ($DA0C) – wertet eine Zahl (0-64000) als ASCII-Folge (Bit 7 = 0) aus.
Vorher: TXTPTR = Zeiger auf erstes Zeichen - 1.
Aufruf: Register durch JSR ↑ CHRGET setzen.
Nachher: LINNUM ($0050, $0051) = ausgewertete Zahl.

PTRGET ($DFE3) – sucht die angegebene Variable in der Variablenliste. Durch SUBFLAG kann bestimmt werden, ob auch Integer-Variablen gesucht werden sollen. Falls nicht, erfolgt bei Integer-Variablen die Meldung „SYNTAX". Existiert die Variable oder das Array-Element nicht, wird neu angelegt (bei Arrays nur bei kleinerem Index als 11).

Vorher: TXTPTR = Zeiger auf 1. Buchstaben des Namens; SUBFLAG ($0014) = Flag für Integer (Bit 7 = 1: keine Integers erlaubt).
Nachher: VARPNT ($0083, $0084) = Zeiger auf Variableninhalt.

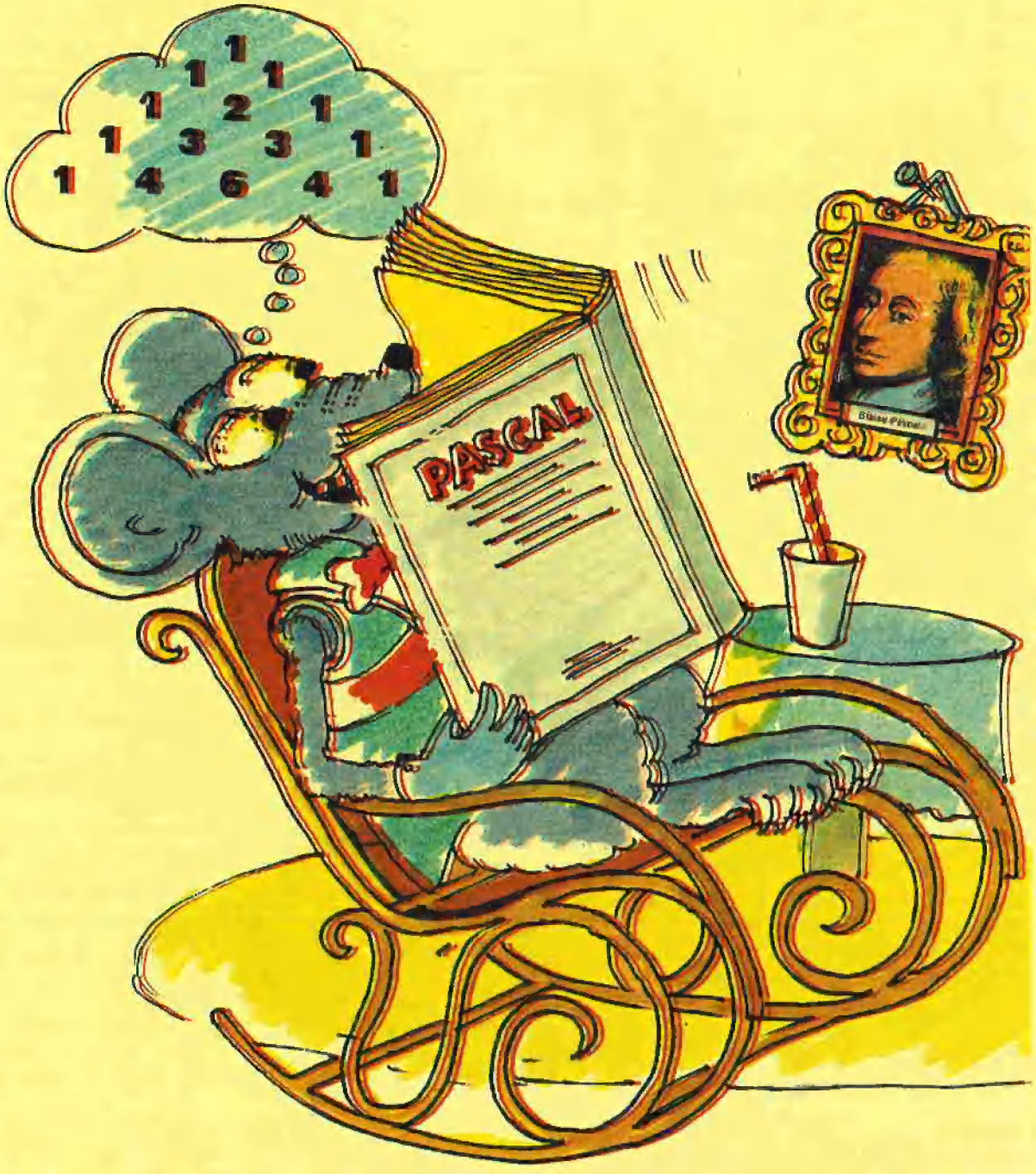
Fortsetzung Seite 50

Pascal-Kompaktkurs

UCSD- und Turbo-Pascal

Teil 1: Grundlagen für Applesoft-Programmierer

von Ulrich Stiehl



I. Allgemeines

1. Pascal und BASIC

Kennen Sie einerseits Applesoft-BASIC und haben Sie andererseits von Pascal keinen „Schimmer einer Ahnung"? Dann ist der erste Teil unseres Kompaktkurses für Sie bestens geeignet. Als BASIC-Programmierer können Sie hier überraschend schnell in Pascal einsteigen, denn wir haben uns im ersten Teil auf diejenigen Befehle beschränkt, die Ihnen in ähnlicher Form von Applesoft her bereits bekannt sind. Dies sind immerhin weit mehr als die Hälfte aller Pascal-Befehle. Jedem Pascal-Befehlsbeispiel wird das entsprechende Applesoft-Pendant vorangestellt, so daß Sie Zeile für Zeile Applesoft mit Pascal vergleichen und auf diesem Wege die Besonderheiten von Pascal erlernen können.
Es dürfte nicht wenige Pascal-Puristen geben, die eine Kontrastierung von Pascal mit BASIC für einen didaktisch falschen Ansatz halten, wie überhaupt Pascal-Anhänger gelegentlich etwas doktrinäre Ansichten vertreten. Wir sind hier ganz anderer Meinung. *Wenn* jemand bereits BASIC und womöglich auch Assembler gelernt hat, *dann* wäre es völlig abwegig, nicht auf diesem Wissen aufbauen zu wollen. Außerdem ist Pascal keine „heilige" Sprache. Sie hat zweifelsohne ihre Vorzüge, aber auch ebenso ihre Mängel.

Listing
Das abgedruckte Listing **KURS1** ist die UCSD-Version (lauffähig unter Apple-Pascal 1.1 und 1.2), die auf der Peeker-Sammeldisk #10 unter dem Dateinamen **UCSD.TEXT** abgelegt ist. Außerdem befindet sich auf derselben Sammeldisk die Turbo-Version namens **TURB.PAS** (lauffähig unter CP/M-Turbo-Pascal 3.0). Bei der Turbo-Version wurden aus Platzgründen die Applesoft-Kommentare gestrichen, damit direkt im Arbeitsspeicher compiliert werden kann. Außerdem waren einige Anpassungen erforderlich.

Definitionen
Standard-Pascal = von Niklaus Wirth, Schweiz, in den Jahren 1968-1973 entwickelte und nach Blaise Pascal benannte Programmiersprache, die im zweiten Teil vom „User Manual and Report" (Springer-Verlag) definiert wird.
UCSD-Pascal = an den Apple II(+/e/c) unter 6502 stark angepaßtes, spezielles UCSD-Pascal (UCSD = University of California in San Diego) mit eigenem Diskettenbetriebssystem; neueste Version 1.2; Hersteller: Fa. Apple, USA.
Turbo-Pascal = an den Apple II(+/e/c) unter Z80 schwach angepaßtes, spezielles Turbo-Pascal mit normalem Apple-CP/M-Betriebssystem (z.B. CP/M 2.26); neueste Version 3.0; Hersteller: Fa. Borland, USA.

2. Editor und Compiler

BASIC ist normalerweise als Interpretersprache implementiert, wenngleich beispielsweise zu Applesoft verschiedene Compiler erhältlich sind. Pascal ist demgegenüber normalerweise als Compilersprache implementiert; dies gilt namentlich für UCSD- und Turbo-Pascal. Bei einer Compilersprache muß zunächst der **Quelltext** (Source-Text) mit einem **Editor** erstellt und dann auf Diskette gespeichert werden (in UCSD-Pascal mit dem Suffix „.TEXT", in Turbo-Pascal mit dem Suffix „.PAS", also z.B. DEMO.TEXT oder DEMO.PAS). Der Editor ist im Prinzip wie ein gewöhnliches Textverarbeitungsprogramm aufgebaut, doch darf nicht völlig „endlos", d.h. ohne jegliche Returns, geschrieben werden. Praktisch heißt dies, daß eine Quelltextzeile normalerweise nicht mehr als 78 Zeichen umfassen sollte. Ein Programm, d.h. die Summe der Quelltextzeilen, könnte folgendermaßen aussehen:

```
PROGRAM DEMO;
BEGIN
WRITELN ('Demo')
END.
```

Der fertige Quelltext dieses Mini-Programms wird auf der Diskette gespeichert und läßt sich dann mit Hilfe eines **Compilers** zu einem lauffähigen **Objektcode** compilieren, der auf der Diskette in UCSD-Pascal mit dem Suffix „.CODE" und in Turbo-Pascal mit dem Suffix „.COM" abgelegt wird (also z.B. DEMO.CODE oder DEMO.COM). Der UCSD-Compiler erzeugt jedoch nur einen pseudo-compilierten Objektcode (= P-Code), der insofern unselbständig ist, als er von dem sog. P-Code-Interpreter abgearbeitet werden muß, während der Turbo-Compiler einen reinen Z80-Objektcode erzeugt, der unabhängig von Turbo-Pascal wie eine normale CP/M-COM-Datei gestartet werden kann. Damit kann man auch Turbo-Pascal-Programme an Personen weitergeben, die über Turbo-Pascal gar nicht verfügen.
(Editor und Compiler sowie das UCSD- bzw. CP/M-Betriebssystem können aus Platzgründen hier nicht behandelt werden.)

3. UCSD- und Turbo-Pascal

Für die Programmentwicklung unter UCSD-Pascal ergibt sich folgender Zyklus:
1. Quelltext im Editor schreiben oder ändern.
2. Quelltext auf Diskette speichern.
3. P-Code mit Compiler auf Diskette erzeugen. Nach Fehlermeldungen zurück zu 1.
4. P-Code aus dem UCSD-Betriebssystem heraus über den Run-Befehl von der Diskette starten.

Unter Turbo-Pascal, das im wesentlichen aus der sehr kompakten Datei TURBO.COM besteht, die Editor und Compiler vereint, ergeben sich demgegenüber folgende Varianten:
1. Quelltext im Editor schreiben oder ändern.
2. Quelltext auf Diskette speichern.
3. Objektcode wahlweise auf Diskette oder direkt im Speicher erzeugen (letzteres nur bei kleineren Quelltexten). Nach Fehlermeldungen zurück zu 1.
4. Objektcode wahlweise aus dem CP/M-Betriebssystem heraus von der Diskette oder im Speicher aus dem Turbo-Menü heraus über den Run-Befehl starten (letzteres nur bei kleineren Objektcodes).

„Kleinerer" Quelltext und Objektcode besagt, daß man unter Turbo-Pascal je nach CP/M-Version bis zu ca. 30K für Text *und* Code zur Verfügung hat, was für Anfängerprogramme immer ausreicht. Da man somit unter Turbo-Pascal wie ein „Wirbelwind" (lat. „turbo") zwischen Edit, Compile und Run ohne jeglichen Diskettenzugriff wechseln kann, eignet sich Turbo-Pascal besser für Anfänger, die nach der „Versuch-und-Irrtum"-Methode lernen, während UCSD-Pascal mehr für systematische „Papier-und-Bleistift"-Programmierer gedacht ist, denen selten ein Tipp- oder Denkfehler unterläuft. Nehmen wir als Beispiel unseren KURS1: Die reine Compilierungszeit beträgt hier unter UCSD-Pascal 130s; hinzu kommen bis zu 10s für jeden Editor-, Compiler- oder Run-Modus-Wechsel, so daß ein Edit-Compile-Run-Zyklus mehr als 150s einnimmt. Demgegenüber dauert unter Turbo-Pascal die reine Compilierungszeit ca. 10s, und der Edit-Compile-Run-Wechsel ist praktisch „blitzartig". Hinzu kommt ferner, daß erstens der Turbo-Objektcode schneller als der UCSD-P-Code abgearbeitet wird und daß zweitens Turbo-Pascal problemlos mit 1 Diskettenlaufwerk betrieben werden kann, während man bei UCSD-Pascal wegen des riesigen Betriebssystems als 1-Drive-Besitzer am besten gleich gar nicht anfängt. Aus alledem könnte man schließen, daß Turbo-Pascal erheblich besser als UCSD-Pascal sei. Dem ist jedoch nicht so. Beide Pascal-Versionen haben ihre Vor- und Nachteile. UCSD-Pascal ist besser an den Apple angepaßt (mit implementierter Turtle-Grafik, mit einem gesonderten 6502-Assembler usw.), während zusätzliche Turbo-Pascal-Eigenschaften nur bei der IBM-PC-Version vorhanden sind.

Apple-UCSD-Pascal 1.2 erhält man über Apple-Händler mit 4 Disketten, 2 Handbüchern und 2 Ergänzungsbroschüren zum Preis von ca. DM 900,-. Zum Lieferumfang gehören ein komplettes Diskettenbetriebssystem mit Filer, Formatter usw. sowie zahlreiche Hilfsprogramme wie Assembler, Linker, Library-Verwaltungsprogramme usw. Apple-UCSD-Pascal ist auf jedem Apple lauffähig (II+/e/c), wobei jedoch eine 80-Zeichenkarte zweckmäßig und zwei Laufwerke praktisch unumgänglich sind.

Turbo-Pascal 3.0 erhält man über die Firma Heimsoeth mit 2 Disketten und 1 Handbuch zum Preis von ca. DM 230,-. Eine Z80-Karte und das CP/M-Betriebssystem müssen gesondert für zusammen ca. DM 1000,- erworben werden (Die Turbo-Systemdiskette bootet nicht). Eine 80-Zeichenkarte ist erforderlich, ein zweites Laufwerk hingegen entbehrlich. Für die optimale Anpassung an den Apple muß man selbst sorgen. Dies beginnt bereits bei der Konfigurierung der 80-Z/Z-Bildschirmsteuerung und endet bei der Hires-Grafik.

– Turbo-Pascal eignet sich besser für *Anfänger* und solche Programmierer, die kleine Programme ohne „Libraries" erstellen wollen. UCSD-Pascal eignet sich besser für *Fortgeschrittene* und solche Programmierer, die von dem nur in UCSD-Pascal hochentwickelten „Bibliothekskonzept" Gebrauch machen wollen.

– Die *Verarbeitungsgeschwindigkeit* (mit Ausnahme des Diskettenzugriffs) ist bei Turbo-Pascal größer als bei UCSD-Pascal.
– Turbo-Pascal eignet sich besser für anspruchsvolle *Fließkomma-Mathematik*, während UCSD-Pascal bei *Grafik* und sonstigen apple-spezifischen Anwendungen vorzuziehen ist.

Dies sind einige der Vor- und Nachteile, die Sie bei Ihrer Kaufentscheidung berücksichtigen können.

4. Ist Pascal standardisiert?

Würde man den Schwierigkeitsgrad einer Programmiersprache von der Anzahl der Befehle abhängig machen, so wäre (6502-)Assembler sehr leicht und Pascal sehr schwer. Die Anzahl der Befehle ist indes nur ein quantitatives Problem; wichtiger ist die Systematik und Einheitlichkeit der Befehle. Obwohl Pascal als eine standardisierte Sprache gerühmt wird, haben sich leider bei den einzelnen Implementierungen zahlreiche „Wunderlichkeiten" eingeschlichen, die es erforderlich machen, daß man sich bei zu vielen Befehlen zu viele Spezialfälle merken muß. Greifen wir zur Verdeutlichung die Befehle WRITE/WRITELN und READ/READLN heraus, die in etwa den Applesoft-Befehlen PRINT (ohne und mit folgendem Semikolon) und GET/INPUT entsprechen:

WRITE (A) etwa PRINT A$;
WRITELN (A) etwa PRINT A$
READ (A) etwa GET A$
READLN (A) etwa INPUT A$

Man beachte, daß im Gegensatz zu Applesoft in Pascal die aus- oder einzugebende Variable A eingeklammert werden muß und daß der „$"-Zusatz entfällt (A sei hier eine sog. Char-Variable, d.h. eine Einzelzeichen-Variable).
Was passiert nun, wenn man die Klammer leer läßt, also
WRITE () bzw.
WRITELN ()
eingibt. In UCSD-Pascal ist beides zulässig, in Turbo-Pascal hingegen beides verboten.
Wie steht es nun mit einem nackten
WRITE bzw.
WRITELN
ohne Klammern (ähnlich wie ein nacktes PRINT in Applesoft)? Hier stellen wir fest, daß in Turbo-Pascal WRITE und WRITELN erlaubt sind, in UCSD-Pascal dagegen nur WRITELN. Es gilt also im einzelnen:
WRITE (): UCSD ja, Turbo nein
WRITELN (): UCSD ja, Turbo nein
WRITE: UCSD nein, Turbo ja
WRITELN: UCSD ja, Turbo ja
Setzen wir unsere Experimente mit READ/READLN fort. In Applesoft wären GET bzw INPUT ohne nachfolgende Variable nicht denkbar. Schlagen wir hingegen irgendein Pascal-Buch auf, z.B. „Rollke; Grundkurs, Band 2, S. 25", so finden wir plötzlich ohne nähere Erläuterung ein nacktes READLN. Sie werden sich mit Recht fragen, was eigentlich mit dem über die Tastatur eingegebenen Wert passiert, wenn keine Wertzuweisung erfolgt. Außerdem werden Sie sich überlegen, ob neben READLN auch READ funktioniert. Durch Experimente werden Sie ermitteln:
READ: UCSD nein, Turbo ja
READLN: UCSD ja, Turbo ja
Ferner gibt es noch die Möglichkeit, bei READ/READLN die Tastatur direkt ohne Bildschirmecho abzufragen, und zwar mit dem Zusatz KEYBOARD in UCSD-Pascal bzw. KBD in Turbo-Pascal. Diese Befehle entsprechend etwa dem Applesoft-Peek der Speicherstelle -16384 = $C000. Hier werden Sie feststellen, daß in UCSD- bzw. Turbo-Pascal
READ (KEYBOARD) bzw.
READ (KBD)
beide keinerlei Wirkung zeigen, weil die entsprechende Stelle im Objektcode quasi übersprungen wird, während zumindest
READ
allein ohne KBD in Turbo-Pascal möglich ist. Umgekehrt zeigen in UCSD- bzw. Turbo-Pascal
READLN (KEYBOARD) bzw.
READLN (KBD)
beide eine Wirkung.
Gehen wir nun noch einen Schritt weiter und versuchen wir, über die Tastatur der Char-Variablen A durch Tippen der Return-Taste jeweils den entsprechenden ASCII-Code 13 zuzuweisen. Dann stellen wir fest, daß in UCSD-Pascal
READ (A) und
READ (KEYBOARD, A)
beide eine Leertaste (ASCII-Code 32) statt Return zuweisen.
Anders bei Turbo-Pascal. Hier ergibt
READ (A)
Ctrl-Z (ASCII-Code 26), während
READ (KBD, A)
endlich das ersehnte Return zuweist.
Um die Verwirrung komplett zu machen, sei an dieser Stelle bereits erwähnt, daß nur wenige der zahlreichen Pascal-Datentypen über READLN/WRITELN ein- bzw. ausgegeben werden können. Nehmen wir als Beispiel die Wahrheitswerte. In Applesoft ist der Befehl
PRINT 1=1
zulässig und ergibt den Wert 1 (= wahr), während
PRINT 1=2
den Wert 0 (= falsch) ausgibt. In UCSD-Pascal können Sie im Gegensatz zu Turbo-Pascal nicht einfach
WRITELN (1=1)
verwenden. Vielmehr müssen Sie zu umständlichen Formulierungen greifen, z.B.
IF 1=1 THEN WRITELN (1)
oder
IF 1=1 THEN WRITELN ('wahr')
Wenn Sie sich jedoch einige Zeit mit UCSD-Pascal befaßt haben, werden Sie den ORD-„Trick" kennenlernen.
WRITELN (ORD (1=1))
gibt den Wert 1 aus, während
WRITELN (ORD (1=2))
den Wert 0 anzeigt. Was passiert jedoch, wenn Sie beispielsweise den Befehl
WRITELN (ORD (NOT (1=1)))
verwenden. Während Turbo-Pascal hier wieder 0 ausgibt, erhalten Sie in UCSD-Pascal plötzlich -2. Folglich müssen Sie in UCSD-Pascal lernen, unter welchen Umständen Sie einen vernünftigen Wert erwarten dürfen.
Diese Beispiele zeigen auf das deutlichste, daß man sich in Pascal mit vielen „wundersamen Dingen" vertraut machen muß und daß es mit der vielgerühmten Standardisierung nicht zum besten steht.

II. Das definitorische Grundgerüst

1. Namen

Namen sind (1) reservierte Wörter sowie (2a) vordefinierte und (2b) benutzerdefinierte Bezeichner. Wie die meisten Programmiersprachen kennt auch Pascal Befehlswörter, z.B. FOR, AND, OR usw., die jedoch nur aus Buchstaben (wahlweise Groß- oder Kleinbuchstaben) bestehen. Im einzelnen unterscheidet man die **reservierten Wörter** (z.B. AND, OR, END usw.) von den **vordefinierten Bezeichnern** = Standardbezeichnern (z.B. ABS, CHR, GOTOXY usw.), da letztere von der jeweiligen Pascal-Implementierung abhängen. Wie in Applesoft kann man in Pascal den Konstanten und Variablen eigene Namen geben, die **benutzerdefinierte Bezeichner** heißen, z.B. im Falle von konstanten Zahlen:
A = 10
A1 = 10
ZAHL = 10 usw.
Benutzerdefinierte Bezeichner müssen mit einem Groß- oder Kleinbuchstaben anfangen, dürfen danach jedoch auch Ziffern aufweisen. In UCSD-Pascal sind die ersten 8 Zeichen eines Bezeichners signifikant, in Turbo-Pascal theoretisch sogar bis zu 127 Zeichen. Sonderzeichen ($, %, !, # usw.) sind nicht zulässig (Ausnahme: _ = Unterstreichungsstrich). Benutzerdefinierte Bezeichner können reservierten Wörtern und Standardbezeichnern *ähneln* (z.B. FOR_EVER, BEGIN_NER usw.), dürfen ihnen jedoch (normalerweise) nicht gleichen. Der Unterstreichungsstrich hat übrigens in UCSD-Pascal einen „Nullwert", d.h. FOR_EVER und FOREVER sind gleich. Schließlich beachte man, daß Groß- und Kleinbuchstaben in demselben Programm beliebig gemischt werden könnten, d.h. man könnte z.B. ZAHL als Konstante definieren und später im Programm als zahl, zAhL usw. aufrufen.
Während es in Applesoft im Prinzip nur Konstanten- und Variablen-Bezeichner gibt, kennt man in Pascal Konstanten-, Variablen-, Prozedur-, Funktions-, Programm- (und in Turbo-Pascal zusätzlich) Label-Bezeichner. (In UCSD-Pascal sind als Labels nur Ziffern zulässig, in Turbo-Pascal wahlweise Ziffern oder Namen.)

2. Begrenzer

In Pascal gibt es Sonderzeichen, die entweder bestimmte Funktionen erfüllen (z.B. die mathematischen Operatoren +, -, *, /) oder ausschließlich der Begrenzung von Namen dienen (z.B. Leertaste und Return). Operatoren sind automatisch gleichzeitig Begrenzer. Einerseits läßt der Editor eine frei formatierte Eingabe des Quelltextes zu. Andererseits reagiert der Compiler etwas „unwirsch" auf fehlende Begrenzer. Betrachten wir hierzu folgende Beispiele (die eingeklammerten Nummern gehören nicht zum Programm):
{PRINT: PRINT}
(1) WRITELN; WRITELN
(2) WRITELN;
WRITELN
(3) WRITELN;WRITELN
(4) WRI
TELN; WRI TELN

Befehle werden in Pascal durch „;“ und in Applesoft durch „:“ getrennt. Nach dem „;“ kann man zusätzlich eine Leertaste (1) oder ein Return (2) einfügen oder auf beides verzichten (3). Man darf jedoch nicht *mitten in* ein Befehlswort ein Return oder eine Leertaste einfügen (4).

```
{PRINT A * A}
(1) WRITELN ( A * A )
(2) WRITELN (A*A)
(3) WRITELN(A*A)
```

Wie ersichtlich, werden Ausdrücke, die der WRITELN-Befehl ausgeben soll, im Gegensatz zu Applesoft in runde Klammern eingeschlossen. „*“, „(“ und „)“ dienen gleichzeitig als Begrenzer, so daß Leertasten entbehrlich sind (2, 3), jedoch zulässig wären (1).

```
{PRINT 10 DIV 2}
(1) WRITELN (10 DIV 2)
(2) WRITELN (10
DIV
2
)
(3) WRITELN (10DIV2)
(4) WRITELN ((10)DIV(2))
```

Das dritte Beispiel zeigt das Ergebnis einer Integer-Division an (analog zu manchen BASIC-Versionen). Die Version (3) ist unzulässig, weil „DIV“ als Befehlswort entweder links und rechts durch eine Leertaste (1) bzw. ein Return (2) oder durch einen sonstigen Begrenzer abgesetzt werden muß, z.B. durch eine überflüssige Klammer (4).

Man kann bei Begrenzern im einzelnen unterscheiden:
Einzelchen-Begrenzer: + - * / = < > : . , ; ↑
Doppelzeichen-Begrenzer: := .. <= >= <>
Paarzeichen-Begrenzer: {-} (-) [-] '-'
Standard-Begrenzer: (Leertaste, Return)
(In Turbo-Pascal gibt es außerdem die Begrenzer # und $ bei hexadezimalen Zahlen.)

Man merke sich:
a) Ein Name (Befehlswort, Bezeichner) sowie ein Doppelzeichen-Begrenzer darf niemals *in sich* durch einen Standard-Begrenzer unterbrochen werden, z.B.:
A := 1 ist richtig
A : = 1 ist falsch
b) Ein Name (s.o. DIV usw.) muß stets links und rechts begrenzt werden.
Einige Begrenzer wollen wir bereits an dieser Stelle andeutungsweise erklären:

(-)
steht nach WRITELN und diversen Funktionen, z.B. SIN (X).

{-}
wird bei Kommentaren und Compiler-Optionen verwendet; ersatzweise „(*-*)“ als Doppelzeichen-Begrenzer tippen.

[-]
kommt u.a. bei der Längendefinition von Strings vor; ersatzweise in Turbo-Pascal „(.-.)“ benutzen.

+, -, *, /
sind mathematische Operatoren wie in Applesoft.

=, <>, <=, >=
sind Vergleichsoperatoren wie in Applesoft, doch beachte man, daß ><, =< und => illegal sind.

'-'
wird bei String-Werten, z.B. 'Demo', verwendet; in Applesoft Gänsefüßchen: "Demo";

3. Programmaufbau

(Zu dem nachfolgenden Kapitel vergleiche man im Listing KURS1 die Abschnitte 1.0 und 2.0.)

Ein Pascal-Programm besteht aus folgenden Teilen:
1. Kopf (PROGRAM)
2. Labels (LABEL)
3. Konstanten (CONST)
4. Typen (TYPE)
5. Variablen (VAR)
6. Prozeduren (PROCEDURE) (mit 7 mischbar)
7. Funktionen (FUNCTION) (mit 6 mischbar)
8. Programm (Hauptprogramm)

Die Komponenten 1 und 8 sind obligatorisch, alle anderen fakultativ. Jede Komponente außer dem Hauptprogramm selbst beginnt mit einem reservierten Wort. In UCSD-Pascal muß die Reihenfolge strikt eingehalten werden; außerdem darf beispielsweise das reservierte Wort „VAR“ nur ein einziges Mal vorkommen. All diese Einschränkungen gelten nicht für Turbo-Pascal.

3.1. Kopf und Programm

```
PROGRAM DEMO1; {1}

BEGIN {8}
WRITELN ('Anfang');
WRITELN ('Ende')
END.
```

Der Kopf (1) besteht aus dem reservierten Wort „PROGRAM“, gefolgt von einem benutzerdefinierten Programmnamen, gefolgt von einem „;“, hier also
PROGRAM DEMO1;
Vor dem Kopf stehen oft Compileroptionen in geschweiften Klammern. Beispielsweise bedeutet die Option
{$LCONSOLE:}
in UCSD-Pascal, daß während der Compilierung ein ausführliches Listing ($L) auf den Bildschirm (= Console) ausgegeben wird.
Nach dem Kopf steht in unserem KURS1-Listing
USES TRANSCEND, APPLESTUFF;
Damit ist gemeint, daß das UCSD-Programm aus der Programmbibliothek „SYSTEM.LIBRARY“ die (Maschinen)routinen oder Units „TRANSCEND“ und „APPLESTUFF“ benutzt. Dies gilt nicht für die Turbo-Pascal-Version von KURS1; dort sind die transzendenten Funktionen bereits standardmäßig implementiert.

Das eigentliche Programm oder genauer Hauptprogramm (8) beginnt mit dem Wort „BEGIN“ und endet mit dem Wort „END“, gefolgt von einem „.“ Wenn der Compiler auf „END.“ mit Punkt stößt, hört er mit der Compilierung auf. Diesen Umstand kann man ausnutzen, um zusätzliche, noch nicht fertig entwickelte Passagen des Quelltextes von der Compilierung auszuschließen.
Zwischen „BEGIN“ und „END.“ stehen die einzelnen Befehle, die *voneinander* durch „;“ getrennt werden. *Vor* einem „END.“ (oder bei Prozeduren oder allgemein bei Verbundanweisungen vor einem „END;“) muß deshalb niemals ein Semikolon stehen. Man darf es jedoch grundsätzlich setzen (sog. Leeranweisung).

Die Befehle eines Pascal-Programms werden im einfachsten Fall sequentiell abgearbeitet, also hier erst
WRITELN ('Anfang');
und dann
WRITELN ('Ende')
„BEGIN“ und „END“ haben eine Art Einklammerungsfunktion („Verbundanweisung“), worauf weiter unten eingegangen wird.
Zeilennummern fehlen in Pascal, wenn man von dem selten benutzten GOTO-LABEL-Befehl absieht, der in Form eines Beispiels ohne weitere Erläuterungen demonstriert werden soll:

```
PROGRAM GEHEN;
{$G+}
LABEL 1,2,3,4;
BEGIN GOTO 3;
1: WRITELN (1);
GOTO 4;
2: WRITELN (2);
GOTO 1;
3: WRITELN (3);
GOTO 2;
4: WRITELN (4) END.
```

Dieses für Puristen und Nicht-Puristen gleichermaßen abschreckende Beispiel gibt die Zahlen 3, 2, 1, 4 (in dieser Reihenfolge) aus.

3.2. Konstanten und Variablen

```
PROGRAM DEMO2; {1}

CONST {3}
IK = 10;
RK = 1.2345;
CK = 'x';
SK = 'xyz';

VAR {5}
IV: INTEGER;
RV: REAL;
CV: CHAR;
SV: STRING;

BEGIN {8}
WRITELN (IK);
END.
```

Bei diesem erweiterten Demo werden zusätzlich Konstanten (3) und Variablen (5) definiert.

Die **Konstantendefinition** beginnt mit dem Wort „CONST“, gefolgt von einer oder mehreren Gleichsetzungen, die jeweils durch „;“ abgeschlossen werden (*auch* die letzte Gleichsetzung). Gleichsetzungen haben links vom „=“ einen benutzterdefinierten Bezeichner und rechts vom „=“ einen Wert. Im einzelnen gilt (Beispiele):

I = 10;
wird als Integer-Zahl erkannt.
R = 10.0;
wird als Real-Zahl erkannt.
C = 'A';
wird als Char = „Character" = Einzelzeichen erkannt (oder als 1-Zeichen-String).
S = 'ABC';
wird als Zeichenkette = Zeichenfolge = String erkannt.
Bei ganzen Zahlen, die Real-Zahlen sein sollen, muß man also einen Dezimalpunkt einfügen, damit der Compiler weiß, ob eine Integer- oder eine Real-Zahl gemeint ist. Man beachte nämlich, daß es im Gegensatz zu Applesoft in Pascal keine Typzusätze wie „%" für Integer-Zahlen oder „$" für Strings gibt. Außerdem beachte man, daß im Unterschied zu Applesoft in Pascal eine Konstante wirklich eine Konstante ist, also nachträglich im Programm nicht mehr umdefiniert werden kann.

Die **Variablendefinition** beginnt mit dem Wort „VAR", gefolgt von einer oder mehreren Typbestimmungen, die jeweils durch „;" abgeschlossen werden (*auch* die letzte Typbestimmung). Typbestimmungen haben links vom „:" einen benutzterdefinierten Bezeichner und rechts vom „:" eine Datentypbezeichnung. Es gibt u.a. folgende elementare Datentypen:

R: REAL;
Fließkommazahl: in Applesoft 9stellige Mantisse = intern 5 Bytes, in UCSD-Pascal 6stellige Mantisse = intern 4 Bytes, in Turbo-Pascal 11stellige Mantisse = intern 6 Bytes.

I: INTEGER;
Ganzzahl mit Vorzeichen im Bereich -32768 bis +32767: in Applesoft sowie UCSD- und Turbo-Pascal intern 2 Bytes.

C: CHAR;
Einzelnes Zeichen: in UCSD-Pascal intern 2 (!) Bytes, in Turbo-Pascal intern 1 Byte, in Applesoft „String mit 1 Zeichen". Die 2 Bytes rühren bei UCSD-Pascal daher, daß intern grundsätzlich nur mit „Words" = Wörtern = Doppelbytes gearbeitet wird. Dies gilt nicht für Turbo-Pascal.

S: STRING; {nur UCSD!}
S1: STRING[10]
S2: STRING[255]
Zeichenkette oder String. Die Länge, die wie in Applesoft bis zu 255 Zeichen betragen kann, muß in eckigen Klammern angegeben werden. Im Gegensatz zu Applesoft darf die Länge einer String-Variablen in Pascal zwar während des Programms schrumpfen, aber niemals über die vordefinierte Länge hinauswachsen. Es gibt also keinen sich dynamisch verändernden String-Pool mit Garbage-Collection.
Wenn die Längenangabe in UCSD-Pascal fehlt, nimmt der Compiler eine Länge von 80 Zeichen an. Strings werden intern „gepackt" gespeichert, d.h. 1 Byte pro Zeichen, denn Strings sind eigentlich sog. „PACKED ARRAYs OF CHAR". (In Turbo-Pascal entfällt „PACKED" durch automatisches Packen.)
Auf die Darstellung weiterer elementarer Datentypen, z.B. BOOLEAN (UCSD/Turbo) oder BYTE (nur Turbo), sowie „höherer" Datentypen, die mit „TYPE" definiert werden, wird in diesem ersten Teil verzichtet.

In Pascal müssen grundsätzlich alle Konstanten und Variablen *vor* deren Benutzung „deklariert" werden, und zwar bereits vor dem ersten „BEGIN". Dies gilt beispielsweise auch für Laufvariablen von FOR-Schleifen. Ein Applesoft-Programmierer könnte nun annehmen, daß man bei einem großen Programm zunächst Hunderte von Variablen festlegen müßte, bevor man mit der eigentlichen Programm anfängt. Dem ist jedoch nicht so. Infolge der Möglichkeit lokaler Variablen bei Prozeduren (in unserem KURS1 nicht benutzt) reduziert sich die Anzahl der globalen Definitionen auf ein Minimum. Hinzu kommt, daß lokale Namen mit globalen Namen formal identisch sein können, so daß man beispielsweise die FOR-Laufvariable I ungerührt innerhalb von verschiedenen Prozeduren jeweils neu definieren kann.

Abschließend sei auf die Unterschiede zwischen „=", „:" und „:=" eingegangen.

„=" wird u.a. erstens bei Konstantendeklarationen und zweitens bei Vergleichen benutzt, z.B.:
CONST K = 10;
IF K = 10 THEN ...
Dies entspricht der Applesoft-Syntax.

„:" wird u.a. bei Variablendeklarationen benutzt, z.B.
VAR R: REAL;
Variablendeklarationen entfallen in Applesoft, denn dort werden sie „beim Gebrauch erklärt".

„:=" wird bei Wertzuweisungen benutzt, z.B.
R := 123.456;
Hier wird in Applesoft entweder die Form
LET R = 123.456
oder üblicherweise die Form
R = 123.456 gewählt.

3.3. Prozeduren

```
PROGRAM DEMO3; {1}

PROCEDURE A1; {6}
BEGIN
WRITELN ('Prozedur')
END;

BEGIN {8}
A1
END.
```

In Applesoft beginnen Unterprogramme, die mit GOSUB n aufgerufen werden, mit einer bestimmten Zeilennummer n und enden mit RETURN. In Pascal beginnen Unterprogramme mit dem reservierten Wort „PROCEDURE", gefolgt von einem benutzterdefinierten Namen, gefolgt von „;", also z.B.
PROCEDURE A1;
Das eigentliche Unterprogramm beginnt dann mit „BEGIN" und endet mit „END;" mit Semikolon (im Gegensatz zum „END." mit Punkt beim Hauptprogramm). Der Aufruf des Unterprogramms bzw. der Prozedur erfolgt im Hauptprogramm durch einfache Angabe des Prozedurnamens, also z.B.
A1;
Mit dieser vereinfachten Erklärung sind die beachtlichen Leistungen einer Pascal-Prozedur (sowie der analogen Pascal-Funktion) im Vergleich zu den relativ eingeschränkten Applesoft-GOSUBs jedoch bei weitem nicht gewürdigt, denn neben der Möglichkeit lokaler Variablen können auch Variablenwerte zwischen Unterprogramm und Hauptprogramm ausgetauscht werden. Eigentlich sollte im Listing KURS1 das Wort „Prozedur" überhaupt nicht vorkommen, doch kann man in UCSD-Pascal im Gegensatz zu Turbo-Pascal größere Programme, wozu der KURS1 bereits gehört, gar nicht ohne Prozeduren schreiben, weil das Hauptprogramm in UCSD-Pascal nicht mehr als gut 1000 Bytes umfassen darf. Deshalb muß man auch dann Unterprogramme bilden, wenn diese *nur ein einziges Mal* aufgerufen werden. Da Prozeduren außerdem *vor* dem Hauptprogramm aufgeführt werden müssen, führt dies zu einer starken „Kopflastigkeit" des Pascal-Programms: Auf eine große Anzahl von Prozeduren folgt ein verkümmertes Hauptprogramm, das sich im Aufruf der einzelnen Prozeduren erschöpft. Deshalb kann man große UCSD-Pascal-Programme nie von vorne nach hinten kontinuierlich schreiben bzw. lesen.

III. Kommentar zum KURS1

Der KURS1 gliedert sich in die Abschnitte 2.1. bis 2.8. Wenn Sie die Peeker-Sammeldisk #10 besitzen, so konvertieren Sie sich die gewünschte Version (UCSD.TEXT bzw. TURB-.PAS) auf ihre Pascal-Systemdiskette und compilieren Sie den Quelltext, wie es im Peeker 10/1985, S. 46 beschrieben wurde. Dann starten Sie den KURS1 mit den Run-Befehl. Nun wählen Sie aus dem Bildschirm-Menü der Reihe nach die 8 Demos aus und vergleichen Sie das Bildschirm-Ergebnis mit dem entsprechenden Abschnitt des Listings sowie den nachfolgenden Kommentar-Abschnitten 1 bis 8. Wenn Sie den jeweiligen Abschnitt verstanden haben, so nehmen Sie einige Änderungen in den Befehlsbeispielen vor, denn nur durch eigene Anwendung der Befehle werden Sie mit ihnen richtig vertraut werden. In den Kommentaren werden auch einige zusätzliche Befehle genannt, die entweder nur in UCSD- oder nur in Turbo-Pascal vorkommen.

1. Bildschirm-Ausgabe

WRITE: Der WRITE-Befehl schickt die Werte einfacher Datentypen (Integer, Real, Char und String) an das momentane Ausgabegerät, das in unserem Beispiel stets der Bildschirm ist; andere Ausgabegeräte sind Drucker, Diskettendatei usw. Auf das Befehlswort WRITE folgt in runden Klammern entweder *ein* Wert oder mehrere durch Kommas getrennte Werte. Diese Trennung von Aufzählungen durch Kommas gilt auch für viele andere Pascal-Konstrukte, z.B.
VAR
A,B,C: INTEGER;
statt
VAR
A: INTEGER;
B: INTEGER;
C: INTEGER;
Man beachte, daß in Pascal auf jeden Befehl ein „;" folgt, sofern danach kein END steht. Das

Pascal-„;“ entspricht dem Applesoft-„:“.
Beispiele für die Typen Integer, Real, Char und String:

```
{Je 1 Wert}
WRITE (1);
WRITE (123.456);
WRITE ('P');
WRITE ('Peeker');
{Je 2 Werte}
WRITE (1,2);
WRITE (1.1, 2.2);
WRITE ('A', 'B');
WRITE ('AA', 'BB');
{Je 1 Konstante}
WRITE (IK);
WRITE (RK);
WRITE (CK);
WRITE (SK);
{Konstante + Variable}
WRITE (IK, IV);
WRITE (RK, RV);
WRITE (CK, CV);
WRITE (SK, SV);
```

Man beachte die einfachen Anführungszeichen bei Stringwerten. Ferner beachte man, daß Kommentare in geschweifte Klammern = Akkoladen gesetzt werden. Und schließlich denke man immer an das „;“. Dies kann nicht oft genug eingehämmert werden, denn ein einziges fehlendes „;“ bedeutet vollständige Neucompilierung!
Bei Stringwerten ist zu bemerken, daß man über ca. 78 Zeichen (in Applesoft ca. 230 Zeichen) normalerweise nicht hinausgehen kann. Bei längeren Texten muß man entsprechend stückeln. Während in Applesoft das Gänsefüßchen ein Problem darstellt, stellt in Pascal der Apostroph ein Problem dar. Mit z.B.

```
WRITELN ('Pascal''s triangle');
```

kann man den Apostroph ausgeben.

WRITELN: Im Unterschied zu WRITE schickt WRITELN in UCSD-Pascal ein Return und in Turbo-Pascal ein Return-Linefeed nach dem *letzten* Wert, also quasi nach dem „)“, hinterher:

```
WRITELN (1);
```

ergibt 1 + Return.

```
WRITELN (1, 2);
```

ergibt 12 + Return.
Steuerzeichen können mit

```
S := CHR (X); {zu „:=“ und CHR s.u.}
```

definiert werden (in Applesoft mit CHR$ (X)), wobei S eine (String- oder) Char-Variable ist und X die Nummer des Ctrl-Zeichens darstellt.
In UCSD-Pascal wird R = Return als

```
R := CHR (13); {Char}
```

und in Turbo-Pascal als

```
R := CHR (13) + CHR (10); {String}
```

definiert.
Für ein nacktes Return bzw. Return-Linefeed genügt bei beiden Versionen jedoch auch das parameterlose

```
WRITELN;
```

Invers/Normal: Für inverse Bildschirmdarstellung gilt in Apple-Pascal 1.2 (nicht Altversion 1.1!)

```
WRITE (CHR (15)); {invers}
WRITE (CHR (14)); {normal}
```

und in Turbo-Pascal – ähnlich wie INVERSE/NORMAL in Applesoft:

```
LOWVIDEO; {invers}
NORMVIDEO; {normal}
```

Home: Den Bildschirm löscht man in UCSD-*und* Turbo-Pascal mit

```
WRITE (CHR (12));
```

oder *nur* in UCSD-Pascal mit

```
PAGE (OUTPUT);
```

oder *nur* in Turbo-Pascal mit

```
CLRSCR;
```

GOTOXY: Der Cursor läßt sich mit GOTOXY (X,Y) positionieren, wobei X für die Spalte (UCSD: 0-79, Turbo: 1-80) und Y für die Zeile steht (UCSD: 0-23, Turbo: 1-24). Das von Applesoft her bekannte HTAB 1, VTAB 5 wäre also in Turbo-Pascal mit

```
GOTOXY (1,5);
```

und in UCSD-Pascal mit

```
GOTOXY (0,4);
```

zu programmieren.

2. Formatierung

Der WRITE/WRITELN-Befehl läßt eine rechtsbündig-formatierte Ausgabe (Print-Using) von Zahlen und Strings zu. Die allgemeine Form lautet

WRITELN (Ausdruck:G:N);

wobei G für die *Gesamtanzahl* der Stellen des Feldes und N für die *Nachkommastellen* (*nur* bei Fließkomma-Zahlen) steht. Beispiele (jedes „□“ steht für 1 Leertaste):

```
WRITELN (1:5);
```

gibt die Integer-Zahl 1 aus als
□□□□1

```
WRITELN ('A':3);
```

ergibt
□□A

```
WRITELN ('Peeker':7);
```

ergibt
□Peeker
Bei formatierten Fließkomma-Zahlen wird die letzte Nachkommastelle gerundet, so daß man die unterschiedlich langen Mantissen in UCSD- und Turbo-Pascal beachten muß. In UCSD-Pascal ist wegen der nur 6stelligen Mantisse oft keine sinnvolle Rundung möglich. Beispiele:

```
WRITELN (-123.477:7:2);
```

ergibt
-123.48

```
WRITELN (123.477:7:2);
```

ergibt
□123.48
Wenn eine Rundung in dem angegebenen G-N-Format nicht mehr möglich ist, wird *automatisch* auf das für unformatierte Real-Zahlen sonst übliche Exponentialformat (oder bei Turbo-Pascal auf Zusatznullen ...000...) umgeschaltet, wodurch das Feld überschritten werden kann. Wenn die WRITELN-Anweisung Formelausdrücke enthält, so steht die G-N-Formatierung ganz am Schluß, z.B.

```
WRITELN ((5.3 - 4.2) * 3.1:8:2);
```

Bei *nicht*-formatierter Ausgabe von Real-Zahlen steht in der *ersten* Stelle im Falle einer negativen Zahl ein „-“ und im Falle einer positiven Zahl eine *Leertaste* und kein „+“. In Turbo-Pascal wird noch eine *weitere* Leertaste vorangestellt. Beispiel:

```
WRITELN (123.456);
```

ergibt in UCSD-Pascal
□1.2345E2
und in Turbo-Pascal
□□1.2345600000E+02

3. Wertzuweisungen

Die Wertzuweisung zu einer Variablen heißt in Applesoft
V = Ausdruck
und in Pascal

V := Ausdruck;

Links vom „:=“ steht eine Integer-, Real-, Char- oder String-Variable, und rechts vom „:=“ steht ein Literal („wörtlicher“ Ausdruck), eine Konstante, eine Variable oder ein Formelausdruck. Beispiele:

```
IV := 10;
RV := 10.5;
CV := 'A';
SV := 'Turbo';
IV := IK;
RV := RK;
CV := CK;
SV := SK;
RV := RV + RV;
RV := 10.5 + 20.5;
RV := RK + 10.5;
```

Zu den Formelausdrücken s. nächste Abschnitte.

In UCSD- und Turbo-Pascal gibt es die Systemkonstante

MAXINT

(= maximale Integer-Zahl 32767),
die u.a. bei der Wertzuweisung verwendet wird, z.B.

```
IV := MAXINT;
```

Hierfür könnte man auch schreiben:

```
IV := 32767;
```

Andere Systemkonstanten sind TRUE, FALSE NIL und – in Turbo-Pascal – PI.
Man beachte „=“ und „:=“ bei z.B.

```
(a) CONST K = MAXINT;
(b) V := MAXINT;
```

Bei (a) handelt es sich um eine Konstantendeklaration, bei (b) um eine Wertzuweisung.

4. Tastatur-Eingabe

READ: Dieser Befehl, der in Klammern eine Variable erwartet, entspricht etwa dem GET-Befehl in Applesoft. Man verwendet READ, um ein einzelnes Zeichen über das momentane Eingabegerät einzulesen. Dies ist hier die Tastatur (sowie sonst z.B. ein Disketten-Textfile). Beispiel:

```
READ (CV);
```

Bei diesem Befehl wartet UCSD-Pascal auf einen Tastendruck. Tippt man „A“, so wird erstens das Zeichen „A“ angezeigt und zweitens der Char-Variablen CV zugewiesen. Nach „A“ braucht man in UCSD-Pascal kein Return zu drücken. In Turbo-Pascal ist hingegen ein Return erforderlich.

READ (KEYBOARD, CV); {UCSD}
unterscheidet sich von READ (CV) dadurch, daß das getippte Zeichen *nicht* am Bildschirm angezeigt wird. „KEYBOARD“ gehört zu den vordefinierten Bezeichnern. In Turbo-Pascal heißt der entsprechende Befehl
READ (KBD, CV); {Turbo}
Hier ist wie bei den UCSD-Befehlen READ (CV)

und READ (KEYBOARD, CV) *kein* Return erforderlich. Um das UCSD-READ-(CV) in Turbo-Pascal zu simulieren, verwende man die Befehlsfolge
READ (KBD, CV); WRITE (CV);

READLN: Dieser Befehl liest eine String-Variable (= eines oder mehrere Zeichen) ein, zeigt den String am Bildschirm an und erwartet Return als Ende der Eingabe, z.B.
READ (SV);
Man kann mit READLN auch Integer- und Real-Zahlen eingeben, doch „verabschiedet" sich Pascal mit einem Systemfehler, wenn man bei Zahlen nicht die zulässigen Zahlzeichen verwendet. Bei
READ (IV);
wäre bereits eine Leertaste zwischen 2 Ziffern „tödlich". Deshalb gehört ein narrensicheres Zahlen-READLN zu den Standard-Prozedur-Übungen eines jeden UCSD-Pascal-Novizen. (In Turbo-Pascal kann man wie in Applesoft über die VAL-Funktion einen eingegebenen String in eine Zahl konvertieren.)

5. Integer-Mathematik

Die Ganzzahl-Mathematik ist für Kenner des alten Apple-Integer-BASIC nichts Neues. In Applesoft selbst gibt es keine echte Integer-Mathematik, wohl aber bei bestimmten Applesoft-Compilern (z.B. TASC- oder Hayden-Compiler).
Ganzzahlen liegen im Bereich -32768 bis +32767 (intern in UCSD- und Turbo-Pascal 2 Bytes).

+, - und ∗: Dies sind die einfachen Operatoren; hinzu kommt noch +/- als Vorzeichen, das ohne Leertaste direkt vor die Zahl gesetzt werden muß. Beispiele:
IV := 10 + 20;
IV := 20 - 10;
IV := 10 ∗ 10;
WRITELN (-10 + -20); {nur Turbo}
Die Zuweisung (Fließkomma-Variable := Integer-Ausdruck) ist zulässig, z.B.
RV := 3 + 5;
Die umgekehrte Anweisung (Integervariable := Fließkomma-Ausdruck), z.B.
IV := 3.1 - 3.2;
ist im Gegensatz zu Applesoft verboten.
Man merke sich bereits jetzt zwei Besonderheiten von Pascal:
1. Auf das Mischen von Datentypen reagiert der Pascal-Compiler oft „unwirsch". Ein Mischung in der Art
RV := 10 + 10.5;
ist gerade noch zulässig.
2. Pascal-Variablen werden beim Programmstart im Gegensatz zu Applesoft nicht automatisch auf 0 gesetzt. Das Mini-Programm
PROGRAM TEST;
VAR R: REAL;
BEGIN
WRITELN (R)
END.
hätte zur Folge, daß für R derjenige „Wert" angezeigt wird, der sich zufällig im Speicher befindet.
DIV und MOD: Wenn gilt
Dividend : Divisor = Quotient + Rest
dann gilt
DIV = Quotient-Operator,
MOD = Rest-Operator.
Beispiele:
WRITELN (7 DIV 3);
ergibt 2.
WRITELN (7 MOD 3);
ergibt 1.
ABS: Die Absolutfunktion eliminiert ein mögliches Vorzeichen, z.B.
WRITELN (ABS (-10));
ergibt 10.
SQR: Im Gegensatz zum Applesoft-SQR ist hier das Quadrat und nicht die Quadratwurzel gemeint:
WRITELN (SQR (10));
ergibt 100.

PRED und SUCC: PRED = Predecessor = Vorgänger = Integer-Zahl - 1; SUCC = Successor = Nachfolger = Integer-Zahl + 1. Beispiele:
WRITELN (PRED (10));
ergibt 9.
WRITELN (SUCC (10));
ergibt 11.
Vgl. auch Abschnitt 8: PRED und SUCC bei Char-Variablen.

ROUND und TRUNC: Hierbei handelt es sich um das Runden (ROUND) oder Abhacken (TRUNC von to truncate) einer Fließkomma-Zahl zu einer Integer-Zahl, z.B.
WRITELN (ROUND (10.8));
ergibt 11 und
WRITELN (TRUNC (10.8));
ergibt 10.
Wenn sich beim Runden/Abhacken eine zu große Integer-Zahl ergeben würde, erfolgt eine Fehlermeldung. Nur in Turbo-Pascal gibt es wie in Applesoft die INT-Funktion für Real-Zahlen, die größer als MAXINT sind, z.B.
WRITELN (INT (1234567.89));

RANDOM liefert in UCSD-Pascal eine Integer-Zufallszahl, z.B.
WRITELN (RANDOM);
IV := RANDOM;
In Turbo-Pascal ist
RV := RANDOM;
ohne Parameter eine Fließkomma-Funktion und
IV := RANDOM (OBERGRENZE);
mit Parameter eine Integer-Funktion.
PWROFTEN (Power of Ten = Zehnerpotenz; fehlt in Turbo-Pascal) hat in UCSD-Pascal als Argument einen Integer-Exponenten zur im Ausdruck nicht erwähnten Basis 10, z.B.
WRITELN (PWROFTEN (2));
ergibt 100 (10 ↑ 2).
In Turbo-Pascal gibt es noch weitere, insbesondere für Assemblerprogrammierer interessante Integer-Operationen, z.B. SHL = Shift Left, SHR = Shift Right, HI = High Byte, LO = Low Byte, SWAP = Low/High-Vertauschung, sowie im übrigen den zusätzlichen Datentyp BYTE.

6. Fließkomma-Mathematik

Die Fließkomma-Operatoren und -Funktionen sind in Pascal und Applesoft gleichartig, so daß wir nur auf die Besonderheiten aufmerksam zu machen brauchen.
+, -, ∗, / sind die üblichen Operatoren für plus/minus/mal/geteilt. Beispiele:
RV := (1.1 + 2.2) ∗ (3.3 - 2.2);
WRITELN (1.1 ∗ 2.2 + 3.3);
WRITELN (1.1 ∗ (2.2 + 3.3));
Man beachte die von Applesoft her bekannten Prioritätsregeln: Punkt geht vor Strich, Klammer vor alles. Ferner sei darauf hingewiesen, daß die Ausgabe von Real-Werten stets im Exponentialformat erfolgt, sofern keine G-N-Formatierung (s. Abschnitt 2) spezifiziert wird.

Potenz: Der Applesoft-„hoch"-Operator „X ↑ Y" fehlt in Pascal und muß durch
EXP (Y ∗ LN (X));
simuliert werden. Beispiel:
WRITELN (EXP (3.0 ∗ LN (2.0)));
entspricht 2 ↑ 3.

SQR und SQRT: SQR steht für „Square" = Quadrat = X ∗ X, und SQRT steht für „Square Root" = Quadratwurzel von X. In Applesoft steht SQR für SQRT. Beispiele:
WRITELN (SQR (3.0));
ergibt 9.0.
WRITELN (SQRT (9.0));
ergibt 3.0.

SIN, COS, ATAN: Dies sind die üblichen trigonometrischen Funktionen Sinus, Cosinus und Arcustangens. Die Berechnung erfolgt wie in Applesoft im Bogenmaß (= Pi/180; Gegensatz: Gradmaß = 360 Grad). Als Umrechnungsfaktor verwende man in Turbo-Pascal
F := PI / 180;
oder in UCSD-Pascal
F := 0.017453;
Es gilt dann (Beispiele):
WRITELN (SIN (30 ∗ F));
ergibt theoretisch 0.5, und
WRITELN (COS (60 ∗ F));
ergibt ebenfalls theoretisch 0.5. In Turbo-Pascal ist die Konstante **PI** (analog zur Konstanten MAXINT) zusätzlich implementiert.
Die **TAN**-Funktion fehlt in Pascal und muß durch
RV := SIN (X) / COS (X);
simuliert werden.
Statt **ATAN** in UCSD-Pascal heißt es ARCTAN in Turbo-Pascal.

LOG, LN, EXP: LOG steht nur in UCSD-Pascal für die dekadische Logarithmus-Funktion (fehlt in Turbo-Pascal). LN steht in UCSD/Turbo-Pascal für die natürliche Logarithmus-Funktion (in Applesoft LOG genannt). Und EXP steht in UCSD/Turbo-Pascal für die Exponentialfunktion von e. Beispiele:
WRITELN (LOG (100));
ergibt 2, weil 10 ↑ 2 = 100.
WRITELN (LN (2.7182818285));
ergibt 1, weil e ↑ 1 = 2.71... .
WRITELN (EXP (1));
ergibt 2.7182818285.
In Turbo-Pascal begegnet man noch den zusätzlichen Funktionen **INT** (= Stellen vor dem Dezimalpunkt) und **FRAC** (= Stellen nach dem Dezimalpunkt). Beispiele:
WRITELN (INT (12345.6789));
ergibt theoretisch 12345.0, und
WRITELN (FRAC (12345.6789));
ergibt theoretisch 0.6789.

Abschließend sei darauf hingewiesen, daß eine Reihe der UCSD-Integer- und Real-Funktionen

nur in der sog. System-Library enthalten sind und deshalb über
USES TRANSCEND, APPLESTUFF;
in den Objektcode eingebunden werden müssen. In Turbo-Pascal gibt es normalerweise keine Libraries, so daß alle o.g. Funktionen automatisch vorhanden sind.

7. Verzweigungen und Schleifen

Während bei der Integer- und Fließkomma-Mathematik die Gemeinsamkeiten zwischen Applesoft und Pascal noch sehr groß waren, kommen wir jetzt allmählich zu den pascaltypischen Befehlskonstruktionen.
<, =, >, <=, >=, <>: Hierbei handelt es sich um die von Applesoft her bekannten Vergleichsoperatoren. Man beachte, daß „><", „=>" im Gegensatz zu Applesoft illegal sind.
OR, AND, NOT: Dies sind die ebenfalls in Applesoft verwendeten Wahrheitswert-Operatoren = logischen Operatoren = booleschen Operatoren (benannt nach George Boole, geb. 1815; „boolesche" klein und ohne Apostroph). Daneben gibt es noch den Datentyp BOOLEAN und die Konstanten TRUE und FALSE, worauf an dieser Stelle nicht eingegangen wird.

IF-THEN-ELSE
Der sequentielle Befehlsfluß eines Pascal-Programms wird durch Verzweigungen, Schleifen und Unterprogramme durchbrochen. Beim IF-THEN-(ELSE-)Befehl muß man drei Befehlsteile unterscheiden:
– THEN-Befehle T1, T2...
– ELSE-Befehle E1, E2...
– Weitere Befehle W1, W2...

IF-THEN läßt sich umschreiben mit „Wenn Bedingung erfüllt ist, dann THEN-Befehl(e) T1, T2... ausführen, ansonsten direkt die weiteren Befehle W1, W2... ausführen." Je nachdem, ob von THEN ein einziger Befehl oder mehrere Befehle abhängen, unterscheidet man:

```
IF Bedingung THEN
T1;
W1;...
```

oder

```
IF Bedingung THEN
BEGIN
T1;
T2
END;
W1;...
```

Im letzteren Fall ist die BEGIN-END-Klammerung (= Verbundanweisung) erforderlich. Betrachten wir hierzu folgende Beispiele:

```
IF 1 = 1 THEN
WRITELN ('T1');
WRITELN ('W1'); ...
```
Hier werden T1 und W1 ausgegeben; 1-Befehl-Beispiel, Bedingung erfüllt.

```
IF 1 = 2 THEN
WRITELN ('T1');
WRITELN ('W1');
```
Hier wird nur W1 ausgegeben; 1-Befehl-Beispiel, Bedingung nicht erfüllt.

```
IF 1 = 1 THEN
WRITELN ('T1');
WRITELN ('T2');
WRITELN ('W1');
```
Hier werden T1, T2 und W1 ausgegeben; falsches 2-Befehl-Beispiel, das sich jedoch nicht negativ auswirkt, weil Bedingung erfüllt ist.

```
IF 1 = 2 THEN
WRITELN ('T1');
WRITELN ('T2');
WRITELN ('W1');
```
Hier werden T2 und W1 ausgegeben; falsches 2-Befehl-Beispiel, das sich negativ auswirkt, weil Bedingung nicht erfüllt ist.

```
IF 1 = 2 THEN
BEGIN
WRITELN ('T1');
WRITELN ('T2')
END;
WRITELN ('W1');
```
Hier wird nur W1 ausgegeben; richtiges 2-Befehl-Beispiel, Bedingung nicht erfüllt.

Wir merken uns: Wird die Bedingung erfüllt/nicht erfüllt, dann wird die THEN-Anweisung/THEN-Verbundanweisung ausgeführt/nicht ausgeführt. Nach einem „THEN" können theoretisch beliebig viele Befehle durch „BEGIN-END" verbunden werden. In Applesoft können zum Vergleich nach einem THEN nur so viele Befehle folgen, wie noch in den Rest der Programmzeile passen. Vergißt man die BEGIN-END-Klammerung bei *mehreren* THEN-Befehlen, so fährt Pascal *in jedem Fall mit dem zweiten Befehl* nach dem THEN fort. Die Einklammerung wird vor allen Dingen dann kritisch, wenn verschachtelte IF-THEN-Konstruktionen verwendet werden. Ohne Verbundanweisungen könnte man sich nämlich dann nur noch durch GOTOs retten.

IF-THEN-ELSE läßt sich umschreiben mit „Wenn Bedingung erfüllt ist, dann THEN-Befehl(e) T1, T2... ausführen, andernfalls ELSE-Befehl(e) E1, E2... ausführen, und danach in jedem Fall mit den weiteren Befehlen W1, W2... fortfahren". Man merke sich, daß vor dem Wort „ELSE" die „;"-Leeranweisung verboten ist. Betrachten wir nun folgendes Beispiel:

```
IF A = B THEN
BEGIN
WRITELN ('T1');
WRITELN ('T2')
END {hier kein „;"!}
ELSE
BEGIN
WRITELN ('E1');
WRITELN ('E2')
END;
WRITELN ('W1');
```
Wenn A = B wahr ist, dann werden T1, T2 und W1 ausgegeben. Wenn A = B falsch ist, dann werden E1, E2 und W1 ausgegeben. Würde das Wort „ELSE" fehlen, dann würden in jedem Fall auch E1 und E2 ausgegeben. „ELSE" wird also benötigt, um *vor* den weiteren Befehlen W1, W2... anstelle der Wahr-THEN-Alternative T1, T2... die Falsch-ELSE-Alternative E1, E2... ausführen zu können.

CASE-OF-END
IF-THEN und IF-THEN-ELSE sind Entscheidungen, die von dem Wahrheitsgehalt *einer* Bedingung, d.h. einem einzigen logischen Vergleich, abhängen. Am übersichtlichsten wären Entscheidungen, die von *mehreren* Bedingungen abhängen könnten, z.B.
1: A = B ...
2: A > B ...
usw.
Eine ähnliche Konstruktion gibt es in Pascal in Form der CASE-Anweisung, bei der jedoch *keine* logischen Bedingungen, sondern nur *feste Werte* in bezug auf eine vorgegebene Datentyp-Variable – *Selektor* genannt – zur Auswahl stehen. Beispiel:

```
PROGRAM FAELLE;
VAR I: INTEGER;
BEGIN
WRITE ('ZAHL 1-3:');
READLN (I);
CASE I OF {CASE-Anfang}
1: WRITELN (1);
2: WRITELN (2);
3: WRITELN (3)
END {CASE-Ende}
END. {Programm-Ende}
```

Die allgemeine Form lautet:
„CASE" + Selektor + „OF"
Wert1 + „:" + Befehl(e);
Wert2 + „:" + Befehl(e); usw.
END;

In Turbo-Pascal (*nicht* in UCSD-Pascal) kann man vor dem „END" der CASE-Anweisung mit einer ELSE-Anweisung fortfahren. Eine detaillierte Explikation des CASE-Befehls ist an dieser Stelle noch nicht sinnvoll. Wichtig ist dagegen bereits hier der allgemeine Hinweis, daß im Gegensatz zu Applesoft die logischen Operatoren in der Prioritätsskala *vor* den Vergleichsoperatoren rangieren, so daß man bei kombinierten Ausdrücken stets einklammern muß, also in Pascal
IF (1 = 1) AND (2 = 2) THEN...
und nicht wie in Applesoft
IF 1 = 1 AND 2 = 2 THEN ...
denn dies würde in Pascal
IF 1 = (1 AND 2) = 2 THEN...
bedeuten, was prompt zu einer Compiler-Fehlermeldung führen würde.

FOR-TO/DOWNTO-DO
Die von Applesoft her bekannte Aufwärtszählschleife, z.B.
FOR I = 1 TO 10 : ... : NEXT I
lautet in Pascal
FOR I := 1 TO 10 DO ...;
Und die Applesoft-Abwärtszählschleife, z.B.
FOR I = 10 TO 1 STEP -1: ...: NEXT I
lautet in Pascal
FOR I := 10 DOWNTO 1 DO ...;
In Applesoft ist der FOR-NEXT-Verbund deutlicher erkennbar als in Pascal, weshalb sich der Applesoft-Programmierer bei z.B.
FOR I := 1 TO 5 DO
WRITELN (I); WRITELN (I+1);
mit Recht fragen wird, wieviele Male hier I und I+1 ausgegeben werden. Der Schlüssel zum Verständnis liegt wiederum in der Verbundan-

weisung: Wenn die FOR-Schleife nur *einen einzigen* Befehl „tun" (DO) soll, dann folgt unmittelbar nach DO der Befehl. Wenn hingegen *mehrere* Befehle ausgeführt werden sollen, so ist die BEGIN-END-Verbundanweisung erforderlich. Beispiele:

```
FOR I := 1 TO 3 DO
WRITELN ('A');
WRITELN ('B');
```
gibt dreimal „A" und nur einmal „B" aus.

```
FOR I := 1 TO 3 DO
BEGIN
WRITELN ('A');
WRITELN ('B')
END;
```
gibt dreimal „A" *und* dreimal „B" aus.

Im Gegensatz zu Applesoft darf in Pascal die Laufvariable (hier I) *nie* eine Real-Variable sein. Außerdem gibt es kein NEXT, keinen STEP und keine von +/-1 abweichende Schrittweite. FOR-Schleifen mit Real-Variablen und/oder (Integer-)Schrittweiten ungleich +/-1 muß man mit WHILE- oder REPEAT-Schleifen simulieren. Weitere Sonderfälle:

```
FOR I := 1 TO 10 DO;
```
ist eine Leerschleife (nacktes „;") und entspricht der Applesoft-Befehlsfolge
```
FOR I = 1 TO 10: NEXT I
```

```
FOR I := 1 TO 3 DO
FOR J := 1 TO 5 DO
WRITELN (I, ' ', J);
```
entspricht der Applesoft-Befehlsfolge
```
FOR I = 1 TO 3:
FOR J = 1 TO 5:
PRINT I; " "; J:
NEXT J, I
```

```
FOR CV := 'A' TO 'Z' DO
WRITE (CV);
```
ist in Pascal möglich und gibt hier die Buchstaben von A bis Z in einer Zeile aus. Allgemein formuliert läßt die Pascal-FOR-Schleife als Laufvariablen sog. skalare Variablen zu. Dies sind u.a. Integer- und Char-Variablen.

```
FOR I := 11 TO 10 DO
WRITELN (I);
```
bewirkt gar nichts, denn im Gegensatz zu Applesoft wird eine FOR-Schleife übersprungen, wenn der Anfangswert den Endwert übersteigt.

WHILE-DO
Diese Schleife heißt in Worten „Solange die WHILE-Bedingung noch erfüllt ist, führe die DO-BEGIN-END-Befehle aus". Beispiel:

```
A := 1.0;
WHILE A < 10.5 DO
BEGIN
WRITELN (A);
A := A + 0.5
END;
```
(Dieses Demo gibt die Zahlen 1.0, 1.5...10.0 aus.)

Zwischen WHILE und DO steht wie zwischen IF und THEN ein logischer Ausdruck. Da sich dieser logische Ausdruck zur Vermeidung einer Endlos-Schleife ändern sollte (1 Befehl) und da die WHILE-Schleife auch noch etwas Sinnvolles tun sollte (noch 1 Befehl), folgen auf DO mindestens 2 Befehle, so daß eine WHILE-Schleife praktisch immer als BEGIN-END-Verbundanweisung konstruiert wird.
Man beachte, daß die WHILE-Schleife übersprungen, d.h. kein einziges Mal durchlaufen wird, wenn die Bedingung bereits beim ersten Durchlauf nicht erfüllt ist, z.B.

```
I := 2;
WHILE I = 1 DO...
```
wird übersprungen.

REPEAT-UNTIL
Diese Schleife heißt in Worten „Führe die REPEAT-Befehle solange aus, bis die UNTIL-Bedingung nicht mehr erfüllt ist". Beispiel:

```
A := 1.0;
REPEAT
WRITELN (A);
A := A + 0.5
UNTIL A >= 10.5;
```
(Dieses Demo gibt die Zahlen 1.0, 1.5...10.0 aus.)

Der logische Ausdruck steht hier am Ende, d.h. nach UNTIL, so daß die REPEAT-Schleife im Gegensatz zu den FOR- und WHILE-Schleifen *mindestens einmal* durchlaufen wird. Wie das Demo zeigt, läßt sich eine WHILE-Schleife durch eine REPEAT-Schleife ausdrücken und umgekehrt. Im Gegensatz zur WHILE-Schleife ist bei der REPEAT-Schleife *niemals* eine (äußere) BEGIN-END-Verbundanweisung erforderlich, weil REPEAT...UNTIL selbst bereits eine (spezielle) Verbundanweisung darstellt. Wie *vor* dem END eines BEGIN-END-Verbundes muß auch *vor* dem UNTIL eines REPEAT-UNTIL-Verbundes kein „;" stehen; es ist aber hier wie dort als Leeranweisung zulässig.

Exkurs: Das Semikolon, oder wie man zum Puristen wird
Die Semikolon-Regel für „vor und nach END" ist für Anfänger und Fortgeschrittene gleichermaßen verwirrend. Allgemein gilt:
1. *Vor* einem „END" sollte nie (für Puristen: darf nie) ein „;" stehen.
2. *Nach* einem „END" muß meist ein Semikolon stehen. Ausnahmen: „END." am Programmende, „END" vor „ELSE" u.a.
3. Was passiert jedoch, wenn zwei „ENDs" aufeinanderstoßen? Beispiel:

```
PROGRAM FINALE;
VAR I: INTEGER;
BEGIN
FOR I := 1 TO 10 DO
BEGIN
WRITELN (I + I);
WRITELN (I * I) {hier!}
END {hier!}
END.
```
Die 3. Regel lautet dann: *Vor* einem „END" steht kein Semikolon, wenn vor einem „END" ein „END" steht. Ferner gilt der „Spezialfall": *Nach* einem „END" steht kein Semikolon, wenn davor und danach das gleiche steht, wie das, was dazwischen steht. Gemeint ist das mittlere von drei „ENDs"...

Wenn Ihnen das alles zu „wirr" ist, dann setzen Sie das Semikolon „aus Trotz" und trösten Sie sich mit der Erkenntnis, daß selbst der „Pascal-Papst" Niklaus Wirth in seinem „Pascal User Manual und Report", 2. Aufl. 1978, S. 44, 54 und 123 jeweils vor dem „END." ein Semikolon zuviel gesetzt hat. Auch in der Schweiz gibt es noch Willensfreiheit!

8. String-Verarbeitung

Einzelzeichen

PRED und SUCC: Die von Integer-Zahlen bereits bekannten Vorgänger- und Nachfolger-Funktionen gelten auch für den Char-Typ, z.B.

```
WRITELN (PRED ('Z'));
```
ergibt „Y".
```
WRITELN (SUCC ('Y'));
```
ergibt „Z".
ORD und CHR: ORD ermittelt die „Ordnungszahl" eines ASCII-Zeichens, während CHR das ASCII-Zeichen zu einer vorgegebenen „Ordnungszahl" liefert, z.B.
```
WRITELN (ORD ('A'));
```
ergibt die Integer-Zahl 65.
```
WRITELN (CHR (65));
```
ergibt das Zeichen „A".

Gesamtstring

LENGTH ermittelt die Länge eines Strings, z.B.
```
WRITELN (LENGTH ('Peeker'));
```
ergibt die Integer-Zahl 6.
CONCAT in der Form CONCAT (S1, S2, ...) dient der Verschmelzung von zwei oder mehreren Strings, z.B.
```
S := CONCAT ('PAS', 'CAL');
WRITELN (S);
```
ergibt „PASCAL".
In Turbo-Pascal ist auch die Applesoft-Form
```
S := 'PAS' + 'CAL'
```
zulässig („+"-Zeichen).

Teilstring

Eine String-Variable wird, wie wir bereits erfahren haben, mit einer Längenangabe in eckigen Klammern definiert, z.B.
```
VAR S: STRING[100];
```
Die einzelnen Zeichen eines Strings sind wie in Applesoft von 1 bis LENGTH durchnumeriert und haben damit eine Position, z.B.
```
S := 'Peeker'; {123456}
```
POS in der Form POS (T, G) ermittelt die Position P des Teilstrings T im Gesamtstring G, z.B.
```
WRITELN (POS ('CAL', 'PASCAL'));
```
ergibt die Integer-Zahl 4, weil „C" das 4. Zeichen von „PASCAL" ist. Wird der Teilstring nicht gefunden, so liefert POS 0.
COPY in der Form COPY (G, P, A) ergibt den Teilstring T als Zeichenanzahl A ab Position P im Gesamtstring G, z.B.
```
T := COPY ('Peeker', 5, 2);
WRITELN (T);
```
ergibt „er", weil dies die 2 Zeichen ab dem 5. Zeichen von „Peeker" sind.
S[P] := C ersetzt das Zeichen in der Position P des Strings S durch das Einzelzeichen C vom Typ Char, z.B.
```
S := 'lle';
S[3] := 'c';
WRITELN (S);
```
ergibt „llc".

Die Umkehrung C := S[P] ist ebenfalls möglich (Einzelzeichen-COPY-Befehl).
FILLCHAR in der Form FILLCHAR (G[P], A, C) ersetzt im Gesamtstring G ab Position P die Zeichenanzahl A durch das Char-Zeichen C (Beispiel s. Listing).

INSERT und DELETE: Dies sind keine Funktionen, sondern eingebaute Prozeduren, so daß *keine* Wertzuweisung in der Form S := ... erfolgen darf.
INSERT (T, G, P) fügt den Teilstring T in den Gesamtstring G ab Position P ein. DELETE (G, P, A) entfernt aus dem Gesamtstring G die Zeichenanzahl A ab der Position P. Beispiele:

```
T := 'PP';
G := 'ALE';
INSERT (T, G, 2);
WRITELN (G);
```

ergibt „APPLE".

```
DELETE (G, 2, 4);
WRITELN (G);
```

ergibt „A".
Abschließend sei bemerkt, daß die String-Operationen nur teilweise gegen illegale Parameter abgesichert sind, so daß man entsprechende Vorsicht walten lassen sollte.

IV. Pascal-Bücher

Aus der riesigen Fülle der Pascal-Literatur greifen wir einige wichtige Titel heraus.

1. Standard-Pascal

Pascal User Manual and Report
von Kathleen Jensen und Niklaus Wirth
2. Aufl. 1978, 167 S., Manuskriptbuch, kart.
Springer-Verlag, Heidelberg
Obwohl denkbar primitiv in der herstellerischen Ausstattung, ist dieser Titel ein „Muß" für all diejenigen, die sich über die genaue Syntax von Standard-Pascal informieren wollen. Der erste Teil enthält in der Art eines anspruchsvollen und mehr für Informatik-Studenten gedachten Tutorials einen Überblick über Pascal mit zahlreichen Programmbeispielen, während der zweite Teil eine abstrakte, stark formalistische Beschreibung der Sprache und ihrer Elemente bietet. Für Anfänger nicht geeignet.

Algorithmen und Datenstrukturen
von Niklaus Wirth
3. Aufl. 1983, 320 S., kart.
Teubner-Verlag, Stuttgart
Eigentlich handelt es sich bei diesem Titel um ein Informatik-Lehrbuch. Da jedoch alle Programmbeispiele in Pascal geschrieben sind, kann es auch als Pascal-Lektüre für fortgeschrittene Programmierer mit Informatik-Background benutzt werden. Für Anfänger nicht geeignet.

Pascal: Einführung – Programmentwicklung – Strukturen
von Jürgen Plate und Paul Wittstock
1982, 387 S., geb.
Franzis-Verlag, München
Dieses Buch stellt eine umfangreiche Einführung in Standard-Pascal dar und eignet sich für diejenigen Apple-Besitzer, die zusätzlich auf einem Großrechner programmieren wollen oder müssen. Für reine Apple-Programmierer wäre es dagegen eher verwirrend, weil man bei zu vielen Befehlen umdenken müßte. Insgesamt ein gutes und ansprechendes Buch, wenngleich nicht immer präzis genug definiert wird. Beispiel: „Als Seiteneffekt werden Zuweisungen an globale Variable bezeichnet" (S. 129). In dem Buch von Gruber, S. 230, s.u. wird besser definiert: „Man spricht von Seiteneffekt, wenn einer globalen Variablen in einem *untergeordneten* Block ein Wert zugewiesen wird."

2. UCSD-Apple-Pascal

Um es vorweg zu sagen: Es gibt zwar viele Bücher über USCD-Pascal und zahlreiche über Apple-UCSD-Pascal, aber leider kein einziges Buch, das Apple-Pascal systematisch bis in die letzten Details behandelt. Die meisten Titel gehen über die Anfangsgründe nicht hinaus.

Apple Pascal
1. Operating System Reference Manual, 1980, 298 S., spiralgeb., mit 16seitigem Addendum, geh.
2. Language Reference Manual, 1980, 208 S., spiralgeb., mit 16seitigem Addendum, geh.
3. 1.2 Update Manual, 1983, 99 S., spiralgeb.
Apple, München
Deutsche Übersetzung bei Tewi, München
Dies sind die offiziellen Handbücher, die beim Kauf von Apple-Pascal mitgeliefert werden. Das Operating-Manual enthält eine gründliche und gut verständliche Übersicht über das Betriebssystem (Filer, Editor, Compiler, Assembler, Linker, Utility Programs u.a.). Dagegen ist das Language-Manual ein Stückwerk, weil es nur auf diejenigen Besonderheiten eingeht, in denen sich Apple-UCSD-Pascal von „Standard-UCSD-Pascal" unterscheidet. Wie wir jedoch gesehen haben, sind viele Dinge nicht standardisiert, so daß man sich in der berüchtigten Versuch-und-Irrtum-Methode vortasten muß. Nachteilig ist ferner die nicht-integrierte Darstellung durch Addenda und Supplement. Eigentlich ist das Manual in drei Auflagen erschienen. Anstatt daß man, wie dies bei Buchverlagen üblich ist, die Änderungen in die verschiedenen Kapitel der jeweiligen Neuauflage einarbeitete, schuf man zunächst die Addenda zu den Manuals und später das Supplement zu den Addenda, also quasi den Appendix zum Appendix. Auch der Tewi-Verlag hat die Addenda und das Supplement nicht integriert. Trotz alledem sind diese Handbücher unverzichtbar.

Apple II Pascal
von Arthur Luehrmann und Herbert Peckham
1982, ca. 450 S. (kapitelweise paginiert), kart.
Tewi-Verlag, München
Dieses Buch wird zusätzlich kostenlos mitgeliefert, wenn man das Apple-Pascal-Betriebssystem erwirbt, kann aber auch separat gekauft werden. Meines Erachtens ist diese Einführung viel zu weitschweifig und aufgebläht. Jedes der Mini-Programme – meist 5-10-Zeiler – wird seitenlang erläutert. Wem diese Darstellungform zusagt, möge zu diesem Buch greifen. Viele der spezielleren Befehle wie LABEL, GOTO, FILLCHAR, UNITREAD, BLOCKREAD, WSTRING usw. werden jedoch nicht behandelt. Ferner wurde das Library-Konzept (Units, Assembler usw.) ausgespart. Für Anfänger geeignet.

UCSD Pascal 1
von Anton Gruber und Silvia Gutschmidt
1981, 294 S., kart.
Interface-Verlag, Haar
Dieses Buch, dessen Programme noch auf dem ITT geschrieben wurden, ist eine auffallend gut gegliederte, zügig von den leichteren zu den schwierigeren Befehlen voranschreitende Darstellung von Apple-UCSD-Pascal mit präzisen Definitionen. Manche Anfänger werden das Buch jedoch wegen der etwas spartanischen Beispielprogramme für zu formalistisch halten, denn die Darstellungsform ist dem obigen Titel von Luehrmann/Peckham diametral entgegengesetzt. Leider ist nur der erste Band erschienen, so daß beispielsweise die HGR-Befehle oder 6502-Assembler nicht behandelt werden.

Grundkurs in Pascal
von Karl-Herrmann Rollke
Band 1, 4. Aufl. 1985, 221 S., kart.
Band 2, 1. Aufl. 1985, 217 S., kart.
Obwohl es der Titel nicht verrät, wird in diesem zweibändigen Lehrbuch nur Apple-UCSD-Pascal behandelt. Von allen mir bekannten Pascal-Büchern ist dieser Grundkurs mit Abstand die didaktisch am besten gelungene Einführung in Apple-Pascal. Das Werk ist primär für den Schulunterricht an Gymnasien gedacht, eignet sich jedoch auch vorzüglich zum Selbststudium. Der Autor hat sich im übrigen nicht gescheut, im zweiten Band auch auf die 6502-Programmierung einzugehen, obwohl die Programmbeispiele hier trivialer Natur sind. Erwähnenswert ist auch die vorbildliche typographische Gestaltung, was angesichts der vielen auf die Schnelle von mehr schlechten als rechten Manuskripten abphotographierten „Bücher" von Verlagen wie Data-Becker, Vieweg, Luther, Teubner u.a. heute leider keine Selbstverständlichkeit mehr ist. Solche Bücher darf man teilweise nicht einmal postalisch als Büchersendung verschicken; dies sagt wohl alles. Programmlistings in Büchern wird man wohl immer in Matrix- oder Typenraddruckerform akzeptieren müssen, denn nur beim Peeker sind wir infolge der Spezialisierung auf einen Gerätetyp in der technischen Lage, Listings normal im Lichtsatz zu produzieren. Ferner wird man bei wissenschaftlichen Monographien, die nur in kleiner Auflage gedruckt werden, Manuskriptbücher auch in Zukunft in Kauf nehmen müssen. Doch wenn ein *Lehrbuch* wie etwa der Grundkurs von Kaier/Rudolfs (s.u.) aus „schreibmaschinentechnischen" Gründen auf kursive und halbfette Hervorhebungen verzichtet und eckige Klammern und sonstige Sonderzeichen „von Hand" vermerkt, dann ist wohl die Grenze des didaktisch-pädagogisch Zumutbaren erreicht. Man lege einmal das Rollke- und das Kaier-Buch nebeneinander. Bei Rollke halbfette und kursive Merksätze und Marginalien und saubere technische Zeichnungen, bei Kaier schwer lesbare Sperrschrift und „Freihandzeichnungen".

Um den uns überaus zusagenden Grundkurs in späteren Auflagen noch besser zu machen, sei auf einige Fehler aufmerksam gemacht:
Band 1, S. 47: PWROFTEN hat nur Integer-Zahl als Argument (dto. S. 204).
S. 57: Ersatzsymbole „(. .)" gelten nicht für Apple-Pascal.
S. 71: „12." oder „15." sind keine gültigen Real-Zahlen.
S. 112: „Ausdruck" für Selektor bei CASE-Definition nicht ganz korrekt.
S. 178: „A<=x<=E", also x statt i.
Band 2, S. 10: VAR I statt VAR 1.
S. 22: Produkt.Name := 'Buecher' funktioniert.
S. 26: „(TRUE: Art: Strahlung)" ohne Klammer vor TRUE.
S. 30: „SET OF INTEGER" falsch.
S. 184: „READ.../" und „WRITE.../" falsch

All About Pascal
Überwiegend Reprints von alten „Call-A.P.P.L.E."-Aufsätzen
1982, 183 S., kart.
Dieses Buch enthält viele nützliche Anregungen für fortgeschrittene Pascal- und besonders für Assembler-Programmierer mit zahlreichen Utilities und Interna (Systemadressen usw.), die sich allerdings noch auf Apple-Pascal 1.1 beziehen. Der Reader ist leider auf braunem Papier gedruckt und daher schlecht lesbar.

3. Turbo-Pascal

Zu Turbo-Pascal gibt es bislang keine Bücher, die auf die Besonderheiten unter Apple-CP/M eingehen, geschweige denn die apple-spezifischen Hardware-Gegebenheiten wie HGR-Grafik usw. in die Darstellung mit einbeziehen.

Turbo Pascal Handbuch 3.0
1985, 374 S., kart.
Heimsoeth, München
Dies ist das offzielle Manual, das in Verbindung mit der Systemdiskette geliefert wird. Das Buch ist recht formalistisch aufgebaut und damit für Anfänger nicht geeignet, doch ist es zum Nachschlagen unverzichtbar. Ab S. 159 werden die IBM-PC-Extras behandelt, und ab S. 259 wird auf die CP/M-80-Besonderheiten (apple-unspezifisch) eingegangen. Leider kann man sich nicht des Eindrucks erwehren, daß der Übersetzer von der Materie offenbar keine Ahnung hatte. So wird z.B. auf S. 311 von den „nach abnehmender Wichtigkeit" gruppierten Operatoren gesprochen. „*" ist jedoch nicht wichtiger als „+", sondern hat bei der Auswertung von Formelausdrücken *Vorrang* vor „+". Als Pascal-Kenner kann man sich in diesen und ähnlichen Fällen zusammenreimen, was gemeint ist, als Pascal-Neuling jedoch leider nicht.

Turbo Tutor
1985, ca. 200 S. (kapitelweise paginiert), kart.
Heimsoeth, München
Dies ist das Tutorial zu dem obigen Handbuch. Leider enthält dieses Buch, das offenbar ein „Schnellschuß" ist, erschreckend viele Tippfehler. Einer meiner Freude, der sich die Mühe machte, etliche der gelisteten Programme abzutippen, mußte leider feststellen, das sie wegen der Druckfehler teilweise nicht lauffähig sind. Außerdem scheint der Übersetzer beim Deutschunterricht in der Schule gefehlt zu haben. Es empfiehlt sich daher, auf eine hinsichtlich der Orthographie und Interpunktion verbesserte Neuauflage zu warten, zumal das Buch auch sachliche Fehler aufweist, z. B. „Klingel: Char = Chr (7)" funktioniert nicht (s. 18–22).

Turbo Pascal-Wegweiser Grundkurs
von Ekkehard Kaier und Edwin Rudolfs
1985, 262 S., Manuskriptbuch, kart.
Vieweg-Verlag, Braunschweig
Dieser erste Band – nach dem Grundkurs wird noch ein Aufbaukurs folgen – stellt eine gründliche und gut aufbereitete, jedoch durch die Schreibmaschinentypographie didaktisch entwertete Einführung in Turbo-Pascal dar. Die Programmbeispiele sind überwiegend aus dem kaufmännischen Schulunterricht entnommen, was jedoch den Nutzen des Buches nicht schmälert. Wenn ein Peeker-Leser noch das Heft 1/1984 besitzt, so möge er einmal den Aufsatz „Turbo-Pascal – Schritt für Schritt" von Dr. Ekkehard Kaier mit dem „Turbo Pascal-Wegweiser", S. 43 ff. vergleichen („Programmerstellung in 11 Schritten"). Er wird dann feststellen, daß aus dem Peeker ohne Quellenangabe „entlehnt" wurde. Plagiarius te salutat...

V. Pascal-Wortregister

Die nachfolgende Liste faßt die reservierten Wörter und vordefinierten Bezeichner von UCSD- und Turbo-Pascal in einem Gesamtalphabet zusammen. Außerdem sind die Bezeichner der normalen UCSD-SYSTEM.LIBRARY berücksichtigt worden; auf die Namen von Variablen und Konstanten, z.B. REVERSE, wurde dabei allerdings verzichtet. Die kleingeschriebenen Wörter sind in diesem Kurs noch nicht behandelt worden. Manche Befehle können Funktionen und Prozeduren sein, z.B. BLOCKREAD in Turbo-Pascal. In solchen und ähnlichen Fällen mußte der Kürze wegen eine willkürliche Kategorisierung vorgenommen werden.
Wenn der Vermerk „UCSD" oder „Turbo" fehlt, so handelt es sich um gemeinsame Befehlswörter beider Dialekte. Man beachte jedoch, daß „gleiche Befehlswörter = gleiche Befehle" nicht immer gilt (s. READ usw.), so daß man dem Register natürlich keine Feinheiten entnehmen kann.
Der Anfänger möge sich nicht durch die riesige Befehlsliste abschrecken lassen, denn zahlreiche Befehle werden praktisch kaum benötigt (z.B. AUX) oder funktionieren nicht (z.B. INSLINE bei normaler 80-Zeichenkarte) oder sind nicht in den Handbüchern erwähnt, z.B. CBREAK für Ctrl-C-Test.

UCSD = *nur* in UCSD-Apple-Pascal 1.1 vorhanden (1.2 gesondert vermerkt)
Turbo = *nur* in CP/M-Turbo-Pascal 3.0 vorhanden
Const = Konstante
File = Datei (einschließlich der Peripheriegeräte)
Function = Funktion
Procedure = Prozedur
Reserved = reserviertes Wort
Type = Datentyp
UCSD-Lib = UCSD-Library

Wörter
ABS (Function)
absolute (Reserved, Turbo)
addr (Function, Turbo)
AND (Reserved)
append (Procedure, Turbo)
applestuff (UCSD-Lib-Unit)
ARCTAN (Function, Turbo)
array (Reserved)
assign (Procedure, Turbo)
ATAN (Function, UCSD-Lib)
aux (File, Turbo)
auxinptr (Function, Turbo)
auxoutptr (Function, Turbo)
bdos (Procedure, Turbo)
bdoshl (Function, Turbo)
BEGIN (Reserved)
bios (Procedure, Turbo)
bioshl (Function, Turbo)
blockread (Function)
blockwrite (Function)
boolean (Type)
buflen (Function, Turbo)
button (Function, UCSD-Lib)
byte (Type, Turbo)
CASE (Reserved)
cbreak (Function, Turbo)
chain (Procedure, Turbo)
chainstuff (UCSD-Lib-Unit)
char (Type)
chartype (Procedure, UCSD-Lib)
CHR (Function)
close (Procedure)
clreol (Procedure, Turbo)
CLRSCR (Procedure, Turbo)
code (Reserved, UCSD)
con (File, Turbo)
CONCAT (Function)
coninptr (Function, Turbo)
conoutptr (Function, Turbo)
console: (File, UCSD)
CONST (Reserved)
constptr (Function, Turbo)
COPY (Function)
COS (Function, UCSD-Lib + Turbo)
crtexit (Procedure, Turbo)
crtinit (Procedure, Turbo)
data (Reserved, UCSD)
delay (Procedure, Turbo)
DELETE (Procedure)
delline (Procedure, Turbo)
dispose (Procedure, Turbo)
DIV (Reserved)
DO (Reserved)
DOWNTO (Reserved)
drawbock (Procedure, UCSD-Lib)
ELSE (Reserved)
END (Reserved)
eof (Function)
eoln (Function)
erase (Procedure, Turbo)
errorptr (Function, Turbo)
execute (Procedure, Turbo)
exit (Procedure)
EXP (Function, UCSD-Lib + Turbo)
external (Reserved)
false (Const)
file (Reserved)
filepos (Function, Turbo)
filesize (Function, Turbo)
FILLCHAR (Procedure)
fillscreen (Procedure, UCSD-Lib)

flush (Procedure, Turbo)
FOR (Reserved)
forward (Reserved)
FRAC (Function, Turbo)
freemem (Procedure, Turbo)
function (Reserved)
get (Procedure, UCSD)
getcval (Procedure, UCSD-Lib)
getmem (Procedure, Turbo)
GOTO (Reserved)
GOTOXY (Procedure)
grafmode (Procedure, UCSD-Lib)
halt (Procedure)
heapptr (Type, Turbo)
hi (Function, Turbo)
highvideo (Procedure, Turbo)
idsearch (Procedure, UCSD)
IF (Reserved)
implementation (Reserved, UCSD)
in (Reserved)
initturtle (Procedure, UCSD-Lib)
inline (Reserved, Turbo)
input (File)
INSERT (Procedure)
insline (Procedure, Turbo)
INT (Function, Turbo)
INTEGER (Type)
interactive (Type)
interface (Reserved, UCSD)
intrinsic (Reserved, UCSD)
ioresult (Function)
KBD (File, Turbo)
KEYBOARD (File, UCSD)
keypress (Function, UCSD-Lib)
keypressed (Function, Turbo)
LABEL (Reserved)
LENGTH (Function)
LN (Function, UCSD-Lib + Turbo)
lo (Function, Turbo)
LOG (Function, UCSD-Lib)
LOWVIDEO (Procedure, Turbo)
lst (File, Turbo)
lstoutptr (Function, Turbo)
mark (Procedure)
maxavail (Function, Turbo)
MAXINT (Constant)
mem (Type, Turbo)
MEMAVAIL (Function)
MOD (Reserved)
move (Procedure, UCSD-Lib)
move (Procedure, Turbo)
moveleft (Procedure, UCSD)
moveright (Procedure, UCSD)
moveto (Procedure, UCSD-Lib)
new (Procedure)
NORMVIDEO (Procedure, Turbo)
nil (Const)
NOT (Reserved)
note (Procedure, UCSD-Lib)
odd (Function)
OF (Reserved)
openold (Procedure, UCSD)
opennew (Procedure, UCSD)
OR (Reserved)
ORD (Function)
output (File)
overlay (Reserved, Turbo)
ovrdrive (Procedure, Turbo)
packed (Reserved, UCSD)
paddle (Function, UCSD-Lib)
PAGE (Procedure, UCSD)
paramcount (Function, Turbo)
paramstr (Function, Turbo)
pencolor (Procedure, UCSD-Lib)
PI (Const, Turbo)
port (File, Turbo)
POS (Function)
PRED (Function)
printer: (File, UCSD)
PROCEDURE (Reserved)
PROGRAM (Reserved)
ptr (Function, Turbo)
put (Procedure, UCSD)
PWROFTEN (Function, UCSD)
RANDOM (Function, UCSD-Lib + Turbo)
randomize (Procedure, UCSD-Lib + Turbo)
READ (Procedure)
READLN (Procedure)
REAL (Type)
record (Reserved)
recurptr (Type, Turbo)
release (Procedure)
remin: (File, UCSD)
remout: (File, UCSD)
remstatus (Function, UCSD, nur 1.2)
rename (Procedure, Turbo)
REPEAT (Reserved)
reset (Procedure)
rewrite (Procedure)
ROUND (Function)
scan (Function, UCSD)
screenbit (Function, UCSD-Lib)
screencolor (Type, UCSD-Lib)
seek (Procedure)
seekeof (Function, Turbo)
seekeoln (Function, Turbo)
segment (Reserved, UCSD)
set (Reserved)
setchain (Procedure, UCSD-Lib)
setcval (Procedure, UCSD-Lib)
shl (Reserved, Turbo)
shr (Reserved, Turbo)
SIN (Function, UCSD-Lib + Turbo)
sizeof (Function)
SQR (Function)
SQRT (Function, UCSD-Lib + Turbo)
stackptr (Type, Turbo)
str (Function; UCSD-Long-Integer)
str (Function; Turbo-Real/Integer)
STRING (Type)
SUCC (Function)
swap (Function, Turbo)
swapgpon (Procedure, UCSD-Lib, nur 1.2)
swapon (Procedure, UCSD-Lib)
swapoff (Procedure, UCSD-Lib)
systerm: (File, UCSD)
text (Type)
textmode (Procedure, UCSD-Lib)
THEN (Reserved)
time (Procedure, UCSD)
TO (Reserved)
transcend (UCSD-Lib-Unit)
treesearch (Function, UCSD)
trm (File, Turbo)
true (Const)
TRUNC (Function)
ttlout (Procedure, UCSD-Lib)
turn (Procedure, UCSD-Lib)
turnto (Procedure, UCSD-Lib)
turtleang (Function, UCSD-Lib)
turtlegraphics (UCSD-Lib-Unit)
turtlex (Function, UCSD-Lib)
turtley (Function, UCSD-Lib)
type (Reserved)
unit (Reserved, UCSD)
unitbusy (Function, UCSD)
unitclear (Procedure, UCSD)
unitread (Procedure, UCSD)
unitstatus (Procedure, UCSD)
unitwait (Procedure, UCSD)
unitwrite (Procedure, UCSD)
UNTIL (Reserved)
upcase (Function, Turbo)
USES (Reserved, UCSD)
usr (File, Turbo)
usrinptr (Function, Turbo)
usroutptr (Function, Turbo)
val (Procedure, Turbo)
VAR (Reserved)
viewport (Procedure, UCSD-Lib)
wchar (Procedure, UCSD-Lib)
WHILE (Reserved)
with (Reserved)
wstring (Procedure, UCSD-Lib)
WRITE (Procedure)
WRITELN (Procedure)
xor (Reserved, Turbo)

Hinweis zum Teil 2

Der 2. Teil des Kompaktkurses wird sich mit den pascaltypischen Befehlen, Datentypen und Strukturen befassen.

```
{KURS1}
{Pascal fuer Applesoft-Programmierer
 von U. Stiehl/8.8.85
 PA = Pascal; AS = Applesoft
 Laenge TEXT: 36, CODE: 18 Blocks,
 Compilierungszeit: 130 Sek.

 1.0. Kopf/Konstanten/Variablen
 ------------------------------
 PROGRAM, CONST, VAR;
 INTEGER, REAL, CHAR, STRING;
 USES, TRANSCEND, APPLESTUFF}

PROGRAM BEISPIELE;

{Unit TRANSCEND fuer SIN, COS, EXP,
      ATAN, LN, LOG und SQRT;
 Unit APPLESTUFF fuer RANDOM}

USES TRANSCEND, APPLESTUFF;

{Konstantendefinition
 --------------------

 Namen:
 PA: max. 8 Buchstaben/Ziffern
     Typzusatz entfaellt;
 AS: max. 2 Buchstaben/Ziffern
     Typzusatz % + $ erforderlich}

CONST
IK = 100;       {AS: IK% = 100}
RK = 100.5;     {AS: RK  = 100.5}
CK = 'P';       {AS: CK$ = "P"}
SK = 'Peeker'; {AS: SK$ = "Peeker"}

{Variablendefinition
 -------------------

 Integer:
  PA: Integer ca. +/-32767
  AS: ditto
 Real:
  PA: ca. +/-1.23456E-37
  AS: ca. +/-1.23456789E-37
 Char:
  PA: ein EINZELNES Zeichen
  AS: nur normaler String
 String:
  PA: 1-255 Zeichen mit Laenge
  AS: 1-255 Zeichen ohne Laenge}

VAR

IV: INTEGER;       {AS: IV%}
RV: REAL;          {AS: RV}
CV: CHAR;          {AS: CV$ !}
SV: STRING [255]; {AS: SV$}

{Hilfsvariablen fuer Beispiele}

IH: INTEGER;       {AS: IH%}
RH: REAL;          {AS: RH}
CH: CHAR;          {AS: CH$}
SH: STRING [255]; {AS: SH$}

R:  CHAR; {Return R$}
T:  CHAR; {Taste T$}

{2.0. Beispielprogramme
 ----------------------
 PROCEDURE, BEGIN, END}

{Die Befehlsfolge in PA:
  PROCEDURE NAME;
  BEGIN
  ...
  END;
 lautet in AS:
  1000 REM Beginn Subroutine
  1010 ...
  1090 RETURN.
 Aufruf erfolgt in PA mit:
  NAME;
 und in AS mit:
  GOSUB 1000}

{2.1. Bildschirm-Ausgabe
 -----------------------
 CHR(12) Home, CHR(13) Return;
 CHR(15) Inverse, CHR(14) Normal;
 WRITE, WRITELN;
 GOTOXY}

PROCEDURE BILDSCHIRMAUSGABE;
BEGIN

{AS: HOME
 PA:}
WRITE (CHR(12));

{AS: PRINT
 PA:}
WRITELN;

{AS: PRINT CHR$(13);
 PA:}
WRITE (CHR(13));

{AS: PRINT "Hans"; " Meier"
 PA:}
WRITELN ('Hans', ' Meier');

{AS: PRINT "Hans";  R$; "Meier"; R$
 PA:}
WRITELN ('Hans', R, 'Meier', R);

{AS: INVERSE:
     PRINT "Den Apostroph ' so"; R$:
     NORMAL
 PA:}
WRITELN (CHR (15),
         'Den Apostroph '' so',
          CHR (14), R);

WRITE ('WRITE ohne Return: ');
{AS: PRINT 1;:   PRINT " ";:
     PRINT 2.3;: PRINT " ";:
     PRINT "A";:
     PRINT: PRINT
 PA:}
WRITE (1);   WRITE (' ');
WRITE (2.3); WRITE (' ');
WRITE ('A');
WRITELN; WRITELN;
{Returns hier nachgeholt}

{AS: PRINT IK%;" "; RK; " ";
     CK$; " "; SK$
 PA:}
WRITELN (IK, ' ', RK, ' ',
CK, ' ', SK);

{AS: HTAB 30: VTAB 20:
     PRINT "Hier";:
     HTAB 30: VTAB 19:
     PRINT "Daueber";:
     HTAB 1: VTAB 1:
     PRINT "Oben links";:
     HTAB 1: VTAB 20:
     PRINT "Zeile 20"
 PA:}
GOTOXY (29, 19);
WRITE ('Hier');
GOTOXY (29, 18);
WRITE ('Darueber');
GOTOXY (0,0);
WRITE ('Oben links');
GOTOXY (0, 19);
WRITELN ('Zeile 20')

END;


{2.2. Formatierung
 ------------------
 WRITELN (AUSDRUCK:G:N)}

PROCEDURE FORMATIERUNG;
BEGIN

WRITELN ('Pascaltypische, ',
'rechtsbuendige Formatierung');
{'AUSDRUCK:G:N' fehlt in AS,
 'AUSDRUCK:G' durch SPC simulierbar}

WRITELN (R, 'AUSDRUCK:G:N');

WRITELN (R, 'Fliesskommazahl mit ',
 '10 Gesamtstellen G und 1, 2, 3 ',
 'Nachkommastellen N', R);

WRITELN ('/', 123.4567, ' -> ',
         '/', 123.4567:10:1,
         '/', 123.4567:10:2,
         '/', 123.4567:10:3, '/');

WRITELN (R, 'AUSDRUCK:G', R);

{AS: PRINT "Zehn"; SPC (10); "Spaces"
 PA:}
WRITELN ('Zehn', '':10, 'Spaces');

WRITELN (R, 'Integerzahl mit ',
 '6, 7, 8 Gesamtstellen', R);

{AS: PRINT "/"; -1; " -> ";
     "/"; SPC (6-LEN (STR$ (-1))); -1;
     "/"; SPC (7-LEN (STR$ (-1))); -1;
     "/"; SPC (8-LEN (STR$ (-1))); -1;
     "/"
 PA:}
WRITELN ('/', -1, ' -> ',
         '/', -1:6,
         '/', -1:7,
         '/', -1:8,
         '/');

WRITELN (R, 'String mit ',
 '10, 15, 20 Gesamtstellen', R);

{AS: PRINT "/"; SK$; " -> ";
     "/"; SPC (10-LEN (SK$)); SK$;
     "/"; SPC (15-LEN (SK$)); SK$;
     "/"; SPC (20-LEN (SK$)); SK$
 PA:}

WRITELN ('/', SK, ' -> ',
         '/', SK:10,
         '/', SK:15,
         '/', SK:20, '/')

END;

{2.3. Wertzuweisungen
 --------------------
 PA: ':='; AP: '=';
 MAXINT}

PROCEDURE WERTZUWEISUNG;
BEGIN

WRITELN ('''Literals''', R);

{AS: RV  = 765.432: IV% = 333:
     CV$ = "Z":     SV$ = "abc":
     PRINT RV:      PRINT IV%:
     PRINT CV$:     PRINT SV$
 PA:}
RV:= 765.432;  IV:= 765;
CV:= 'Z';      SV:= 'abc';
WRITELN (RV);  WRITELN (IV);
WRITELN (CV);  WRITELN (SV);

WRITELN (R, 'Konstanten', R);

{AS: RV  = RK:  IV% = IK%:
     CV$ = CK$: SV$ = SK$:
     PRINT RV;  " - "; IV%; " - ";
           CV$; " - "; SV$; " - "
 PA:}
RV:= RK; IV:= IK;
CV:= CK; SV:= SK;
WRITELN (RV, ' - ', IV, ' - ',
         CV, ' - ', SV, ' - ');

WRITELN (R, 'Systemkonstanten',
         ', z.B MAXINT:',R);

{AS: IV% =  32767: PRINT IV%:
     IV% = -32767: PRINT IV%
PA:}
IV:=  MAXINT; WRITELN (IV);
IV:= -MAXINT; WRITELN (IV)

END;

{2.4. Tastatur-Eingabe
 ---------------------
 READ, READLN, KEYBOARD}

PROCEDURE TASTATUREINGABE;
BEGIN

WRITE (R, 'READ', R,
          'Mit Anzeige der Taste: ');

{AS: PRINT "Char: ";: GET CV$:
     PRINT CV$;:
     PRINT R$; "Eingegeben: "; CV$
```

```
 PA:}
WRITE ('Char: '); READ (CV);
{Kein WRITE!}
WRITELN (R, 'Eingegeben: ', CV);

WRITE (R, 'READ (KEYBOARD, Taste)',
       R, 'Ohne Anzeige der Taste: ');

{AS: PRINT "Char: ";: Get CV$:
     PRINT R$; "Eingegeben: "; CV$;
     ", ASC "; ASC (CV$);
 PA:}
WRITE ('Char: '); READ (KEYBOARD, CV);
WRITELN (R, 'Eingegeben: ', CV,
', ORD.', ORD (CV)); {s.u. 2.8.}

WRITE (R, 'READLN von Strings: ', R);

{AS: INPUT "String: "; SV$:
     PRINT "Eingegeben: "; SV$
     (REM Oder analog zu PA:
     PRINT "String: ";:
     INPUT ""; SV$)
 PA:}
WRITE   ('String: '); READLN (SV);
WRITELN ('Eingegeben: ', SV);

WRITELN
(R, CHR (15), 'Keine Stringeingabe',
 R, 'bei Zahlen-READLN!!', CHR (14));

{AS: HTAB 1Ø: VTAB 18:
      INPUT "1. Real: "; RV:
     HTAB 3Ø: VTAB 18:
      INPUT "2. Integer: "; IV%:
     PRINT: PRINT "Eingegeben ";
      "Real: "; RV; ", ";
      "Integer: "; IV%
 PA}
GOTOXY (9, 17);
 WRITE  ('1. Real: '); READLN (RV);
GOTOXY (29, 17);
 WRITE  ('2. Integer: '); READLN (IV);
WRITELN; WRITELN ('Eingegeben ',
 'Real: ', RV:1Ø:2, ', ',
 'Integer: ', IV:6)

END;

{2.5. Integer-Mathematik
 ---------------------
 +, -, *;
 DIV, MOD;
 ABS, SQR;
 PRED, SUCC;
 ROUND, TRUNC;
 RANDOM
 PWROFTEN}

PROCEDURE INTEGERMATHE;
BEGIN

WRITELN ('+, - und *', R);

{AS: IV% = 1Ø + 2Ø:
     PRINT "1Ø + 2Ø = "; IV%:
     IV% = 2Ø - 1Ø:
     PRINT "2Ø - 1Ø = "; IV%:
     IV% = 1Ø * 1Ø:
     PRINT "1Ø * 1Ø = "; IV%:
 PA:}
IV:= 1Ø + 2Ø;
WRITELN ('1Ø + 2Ø = ', IV:6);
IV:= 2Ø - 1Ø;
WRITELN ('2Ø - 1Ø = ', IV:6);
IV:= 1Ø * 1Ø;
WRITELN ('1Ø * 1Ø = ', IV:6);

WRITELN (R, 'Quotient und Rest: ');
{Kein Modulus-Befehl in AS!}


{AS: PRINT IK%;: REM 1ØØ
     PRINT " geteilt durch ";:
     IH% = 7:
     PRINT IH%;:
     PRINT " = ";:
     IV% = INT (IK% / IH%):
     PRINT IV%;:
     PRINT " Rest ";:
     REM MOD-Formel
     IV% = INT ( (IK% / IH%
               - INT (IK% / IH%))
               * IH% + Ø.Ø5 )
               * SGN (IK% / IH%):
     PRINT IV%
 PA:}
 WRITE (IK); {1ØØ}
 WRITE (' geteilt durch ');
 IH:= 7;
 WRITE (IH);
 WRITE (' = ');
 IV:= IK DIV IH;
 WRITE (IV);
 WRITE (' Rest ');
 IV:= IK MOD IH;
 WRITELN (IV);

WRITE (R, 'Absolutbetrag /-I/: ');

{AS: IH% = -7:
     IV% = ABS (IH%):
     PRINT IH%; " -> "; IV%
 PA:}
IH:= -7;
IV:= ABS (IH);
WRITELN (IH, ' -> ', IV);

WRITE (R, 'Quadrat: I * I: ');
{Kein eigener Befehl in AS!}

{AS: IH%:= 2Ø:
     IV%:= IH% * IH%:
     PRINT "Quadrat von "; IH%;:
     PRINT " ist "; IV%
 PA:}
IH:= 2Ø;
IV:= SQR (IH);
WRITE ('Quadrat von ', IH);
WRITELN (' ist ', IV);

WRITE (R, 'Vorgaenger/Nachfolger: ');
{In AS simulierbar}

{AS: IH% = 2Ø:
     PRINT IH%;:
     PRINT ": ";:
     PRINT "Vorgaenger "; IH% - 1;:
     PRINT ", ";:
     PRINT "Nachfolger "; IH% + 1
 PA}
IH:= 2Ø;
WRITE (IH);
WRITE (': ');
WRITE ('Vorgaenger ', PRED (IH));
WRITE (', ');
WRITELN ('Nachfolger ', SUCC (IH));

WRITE (R, 'Runden: ');
{In PA ROUND und TRUNC
 nur bis max. 32767.
 In AS INT bis ca. 8388608}

{AS: RV = -666.666:
     IV% = INT (RV):
     PRINT "Real "; RV;:
     PRINT " auf Integer gerundet ";
     IV%;
 PA:}
RV:= -666.666;
IV:= ROUND (RV);
WRITE ('Real ', RV);
WRITE (' auf Integer gerundet ',
IV);

{TRUNC = Nachkommastellen abhacken
 nicht in AS implementiert}
WRITELN (' und abgehackt ',
TRUNC (RV));


WRITE (R, 'Zufallszahl: ');

{AS: RV = RND (1): REM Ø - Ø.999999999
     PRINT RV;:    REM Fließkommazahl!
     PRINT " Bereich 1Ø-1ØØ: ";:
     PRINT INT (1ØØ * RND (1)) + 1
 PA:}
IV:= RANDOM;        {Ø - 32767}
WRITE (IV);         {Integerzahl!}
WRITE (' Bereich 1Ø-1ØØ: ');
WRITELN (1Ø + RANDOM MOD (1ØØ-1Ø+1));

WRITE (R, 'PWROFTEN: 1Ø ↑ Integer: ');

{AS: PRINT "1Ø ↑ 3 = ";
     1Ø ↑ 3
 PA:}
WRITELN ('1Ø ↑ 3 = ',
PWROFTEN (3):6:2)

END;

{2.6. Fliesskomma-Mathematik
 ------------------------
 +, -, *, /;
 ↑ (fehlt);
 ABS, SQR, SQRT;
 SIN, COS, ATAN;
 TAN (fehlt);
 LOG, LN, EXP}

PROCEDURE FLIESSKOMMAMATHE;
BEGIN

WRITELN ('+, -, *, /', R);

{AS:  RV = 15.7: RH = 5.7777:
      PRINT RV; " + "; RH; " = ";
       RV + RH:
      PRINT RV; " - "; RH; " = ";
       RV - RH:
      PRINT RV; " * "; RH; " = ";
       RV * RH:
      PRINT RV; " / "; RH; " = ";
       RV / RH
 PA:}
RV:= 15.7; RH:= 5.7777;
WRITELN (RV, ' + ', RH, ' = ',
 RV + RH:6:2);
WRITELN (RV, ' - ', RH, ' = ',
 RV - RH:6:2);
WRITELN (RV, ' * ', RH, ' = ',
 RV * RH:6:2);
WRITELN (RV, ' / ', RH, ' = ',
 RV / RH:6:2);

WRITELN;
WRITE ('X ↑ Y = ',
'EXP (Y * LN (X)): ');

{AS: RV = 3.Ø: RH = 5.Ø:
     PRINT RV; " ↑ "; RH; " = ";
     RV ↑ RH
 PA: X hoch Y fehlt in PA}

RV:= 3.Ø; RH:= 5.Ø;
WRITELN (RV, ' ↑ ', RH, ' = ',
EXP (RH * LN (RV)):6:2);

WRITELN (R, 'Punkt vor Strich, ',
            'Klammer vor alles', R);

{AS: PRINT " 3 + 4  *  5 - 13  = ";
           3 + 4  *  5 - 13:
     PRINT "(3 + 4) * (5 - 13) = ";
           (3 + 4) * (5 - 13)
 PA:}
WRITELN  (' 3 + 4  *  5 - 13  = ',
          3 + 4  *  5 - 13);
WRITELN  ('(3 + 4) * (5 - 13) = ',
         (3 + 4) * (5 - 13))
END;

PROCEDURE FLIESSREST;
{Aufsplittung}

BEGIN

WRITELN (R, 'Diverse Funktionen', R);
{Beachte, dass
 - Befehlswoerter teils abweichen,
 - TAN in PA fehlt,
 - SQR und LOG in AS fehlen!}

{AS: PRINT "Von Ausgangszahl ";:
      RV = 5.6789:
     PRINT RV; " -> ":
      PRINT
      " ABS: "; ABS (RV);
      " Quad:"; RV * RV;
      " SQR: "; SQR (RV);
      " SIN: "; SIN (RV); R$;
      " COS: "; COS (RV);
      " ATN: "; ATN (RV);
      " TAN: "; TAN (RV);
      " EXP: "; EXP (RV); R$;
```

```
      " Ln:  "; LOG (RV);
      " Log: "; LOG (RV) / LOG (1Ø)
 PA:}
WRITE ('Von Ausgangszahl ');
 RV:= 5.6789;
WRITELN (RV:7:4, ' -> ');
 WRITELN
(' ABS: ', ABS  (RV):7:2,
 ' SQR: ', SQR  (RV):7:2,
 ' SQRT:', SQRT (RV):7:2,
 ' SIN: ', SIN  (RV):7:2, R,
 ' COS: ', COS  (RV):7:2,
 ' ATAN:', ATAN (RV):7:2,
 ' Tan: ', SIN  (RV)/COS (RV):7:2,
 ' EXP: ', EXP  (RV):7:2, R,
 ' LN:  ', LN   (RV):7:2,
 ' LOG: ', LOG  (RV):7:2 )

END;

{2.7. Verzweigungen und Schleifen
 ---------------------------------

 OR, AND, NOT;
 IF...THEN...ELSE...;
 FOR...TO/DOWNTO...DO...;
 WHILE...DO...;
 REPEAT...UNTIL...}

PROCEDURE VERZWEIGUNGEN;
BEGIN

WRITELN ('IF-Vergleiche:', R);
{Beachte: Im Geg. zu AS in PA:
 Erst 'AND, OR', dann '<, >, ='}

{AS: IF 1 = 1 AND 1 <> 2 THEN
     PRINT "1 = 1 ja":
     PRINT ", ";
     REM Else fehlt in AS!
        IF NOT (2 < 1 OR 1 > 2)
        THEN PRINT "2 < 1 nein"
           IF "B" > "A" THEN
           PRINT "'B' > 'A'"

 PA:}
IF (1 = 1) AND (1 <> 2) THEN
BEGIN
 WRITE ('1 = 1 ja')
 WRITE (', ')
END
ELSE WRITE ('So?');
   IF NOT ((2 < 1) OR (1 > 2)) THEN
   WRITE ('2 < 1 nein, ') {kein ';'!}
   ELSE WRITE ('Ach?');
      IF ('B' > 'A') THEN
      WRITELN ('"B" > "A"')
      ELSE WRITELN ('Wie?');

WRITELN (R, 'FOR nur mit Integer-Step 1',
 R, 'Fuer Real-Step WHILE + REPEAT', R);

{AS: FOR RH = 1 TO 5:
     PRINT RH; " * "; RH; " = ";
     RH * RH; ", ";: NEXT:
     PRINT:
       FOR RH = 5 TO 1 STEP -1:
       PRINT RH; " + "; RH; " = ";
       RH + RH; ", ";: NEXT:
       PRINT
 PA}
FOR IH:= 1 TO 5 DO
WRITE (IH, ' * ', IH, ' = ',
IH * IH, ', ');
WRITELN;
  FOR IH:= 5 DOWNTO 1 DO
  WRITE (IH, ' + ', IH, ' = ',
  IH + IH, ', ');
  WRITELN;

{AS: FOR RH = Ø.5 TO 5.Ø STEP Ø.5:
     PRINT RH; " ";: NEXT: PRINT
 PA: 2 Versionen statt FOR}

WRITELN (R, 'Version mit WHILE',
' prueft vorher', R);
RH:= Ø.5;
WHILE RH < 5.5 DO
BEGIN
 WRITE (RH:5:2, ' ');
 RH:= RH + Ø.5
END;
WRITELN;

WRITELN (R, 'Version mit REPEAT',
' prueft nachher', R);
RH:= Ø.5;
REPEAT
 WRITE (RH:5:2, ' ');
 RH:= RH + Ø.5
UNTIL RH >= 5.5;
WRITELN

END;

{2.8. String-Verarbeitung
 -------------------------

 PRED, SUCC;
 ORD, CHR;
 LENGTH, CONCAT;
 String [x], FILLCHAR, POS;
 COPY;
 DELETE, INSERT
 Kein VAL und STR$ in PA!}

PROCEDURE STRINGVERARBEITUNG;
BEGIN

WRITELN ('   Cy:= PRED (Cx), ',
            'Cy:= SUCC (Cx), ',
            'Cx:= CHR (Ix), ',
            'Ix:= ORD (Cx):');

{AS: PRINT "Vor B ist  ";
     CHR$ (ASC ("B") - 1);:
     PRINT ", nach B ist ";
     CHR$ (ASC ("B") + 1): ", ";:
       PRINT "ASC B: "; ASC ("B");:
       PRINT ", CHR 65: "; CHR$ (65)
 PA:}
WRITE ('Vor B ist ',
PRED ('B'));
WRITE (', nach B ist ',
SUCC ('B'), ', ');
  WRITE ('ASC B: ' , ORD ('B'));
  WRITELN (', CHR 65: ', CHR (65));

WRITELN ('   Ix:= LENGTH (Sx):');

{AS: SV$ = "Apple": PRINT SV$; " hat ";:
     IV% = LEN (SK$): PRINT "Len: "; IV%
 PA:}
SV:= 'Apple'; WRITE (SV, ' hat ');
IV:= LENGTH (SV); WRITELN ('Len: ', IV);

WRITELN ('   Sx:= CONCAT (S1, S2...):');

{AS: SV$ = SK$ + "-" + SK$ + "-" + SK$:
     PRINT SV$
 PA:}
SV:= CONCAT (SK, '-', SK, '-', SK);
WRITELN (SV)

END;

PROCEDURE STRINGREST;
{Aufsplittung}

BEGIN

{Nachfolgende Befehle nur noch
 umstaendlich in AS simulierbar}

WRITELN ('   Cx:= Sx [POS], ',
 'Sx [POS]:= Cx, ',
 'FILLCHAR (Sx [POS], Ix, Cx):');

SV:= '*Peeker*'; SH:= '-******-';
WRITE (SV, '; ', SH, ', ');
  FOR IH:= 2 TO 7 DO
  SH [IH]:= SV [IH];
  WRITE (SH, ', ');
    CH:= SV [1]; {also '*'}
    SH [1]:= CH; SH [8]:= CH;
    WRITE (SV, ', ');
      FILLCHAR (SH [2], 6, '$');
      WRITELN (SH);

WRITELN ('   Ix:= POS (von Sy, in Sx):');

SV:= 'Ich lese Peeker';
IH:= POS (SK, SV);
WRITELN ('Sy ', '"', SK, '"',
         ' ist in Sx ', '"', SV, '"',
         ' ab dem ', IH, '.',
         ' Zeichen enthalten');

WRITELN
      ('Sy:= COPY (von Sx, ab POS, Ix):');

SV:= 'Lerne zu peeken ohne zu quieken';
SH:= COPY (SV, 1Ø, 6);
WRITELN ('"',SH, '" entnommen aus: "',
         SV, '"');

WRITELN (' DELETE (Sx, ab POS, Ix), ',
          'INSERT (Sx, in Sy, ab POS):');

{Sind Prozeduren, keine Funktionen}

WRITE (SV, ' DELETE: ');
DELETE (SV, POS ('ohne', SV), 5);
WRITELN (SV);

WRITE (SV, ' INSERT: ');
INSERT ('und ', SV, POS ('zu q', SV));
WRITELN (SV)

END;

{3.Ø. Hauptprogramm
 -------------------

 CASE...OF... END;
 BEGIN, END. mit Punkt;
 MEMAVAIL;
 (AS-Menue nur angedeutet)}

BEGIN  {Programm-Start}

REPEAT {Menue-Start}

R:= CHR(13); {Return}
T:= ' ';     {Taste}

WRITE (CHR(12)); {Ctrl-L}

WRITELN ('***** PASCAL-KURS1 *****',
       R,'=========================',
       R, R,
'1. Bildschirm-Ausgabe', R,
'2. Formatierte Ausgabe', R,
'3. Wertzuweisung', R,
'4. Tastatur-Eingabe', R,
'5. Integer-Mathematik', R,
'6. Fliesskomma-Mathematik', R,
'7. Verzweigungen und Schleifen', R,
'8. String-Verarbeitung', R,
'9. Ende', R);

WHILE (T < '1') OR (T > '9') DO
  READ (KEYBOARD, T);
WRITELN (CHR(12));

CASE T OF      {AS: ON T GOSUB ...)
'1': BILDSCHIRMAUSGABE;
'2': FORMATIERUNG;
'3': WERTZUWEISUNG;
'4': TASTATUREINGABE;
'5': INTEGERMATHE;
'6': BEGIN
     FLIESSKOMMAMATHE;
     FLIESSREST
     END;
'7': VERZWEIGUNGEN;
'8': BEGIN
     STRINGVERARBEITUNG;
     STRINGREST
     END;
'9': WRITELN ('Ende...')
END;

IF T <> '9' THEN
BEGIN
  WRITE (R, 'Taste druecken ');
  READ (CH)
END;

UNTIL T = '9'; {Menue-Ende}

WRITELN ('Freier Speicher: ',
MEMAVAIL)  {AS: FRE (Ø)}

END.  {Programm-Ende}
```

Apple-IIe/c-Screendump in USCD-Pascal

Apple-IIe/c-Screendump in USCD-Pascal

von Ulrich Stiehl

Das folgende kleine Maschinenprogramm gibt den 80-Zeichenbildschirm des Apple IIe/c auf beliebige Drucker aus. Apple-Pascal 1.1 oder 1.2. Getestet auf Epson und Imagewriter. Nicht für Videxkarte geeignet!

1. DUMPTEST.TEXT C(ompilieren.
2. DUMP.TEXT A(ssemblieren.
3. L(inker aufrufen.
   Bei Host File? DUMPTEST
   Bei Lib File? DUMP
   Bei erneut Host File? Return
   Bei erneut Lib File? Return
   Bei Output File? DUMPTEST
   eingeben
4. Drucker einschalten und
   E(xecute DUMPTEST

DUMPTEST.TEXT

```
PROGRAM DUMPTEST;
VAR I: INTEGER;
{so definieren in Host-Programm:}
PROCEDURE DUMP; EXTERNAL;
BEGIN
FOR I:= 1 TO 24 DO
 BEGIN
 WRITE ('1234567890');
 WRITE ('abodefghijklmnopqrstuvwxyz');
 WRITE ('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
 WRITELN;
 END;
{so aufrufen in Host-Programm:}
DUMP;
END.
```

DUMP.TEXT

```
; Pascal-Screendump/us/1.8.85
;
; Druckt 80-Z/Z-Bildschirm
; IIc/IIe über PWRITE.
; Von Pascal-Hostprogramm mit
; einfachem "DUMP;" aufrufen.

          .PROC DUMP
;          Zeile/Spalte änderbar
FRSTCOL   .EQU 1.     ;Werte 1-80
LASTCOL   .EQU 80.    ;Werte 1-80

FRSTLINE  .EQU 1.     ;Werte 1-24
LASTLINE  .EQU 24.    ;Werte 1-24

BASL      .EQU 0028 ;ZP retten!
BASCALC   .EQU 0FBC1 ;ROM!
PWRITE    .EQU 0FF65 ;LC!
PAGE1     .EQU 0C054 ;unten
PAGE2     .EQU 0C055 ;oben
BANKRD    .EQU 0C011 ;BPL:BANK1
ROMRD     .EQU 0C012 ;BPL:ROM
BANK1     .EQU 0C08B ;R/W BANK1
BANK2     .EQU 0C083 ;R/W BANK2
RDROM     .EQU 0C081 ;R/W ROM
SLOTON    .EQU 0C006 ;Slots on
COL80ON   .EQU 0C00D ;Column 80 on
COL80ST   .EQU 0C001 ;Column 80 store

           JMP START
ZEILE     .BYTE 0
SPALTE    .BYTE 0
BANKSTAT  .BYTE 0     ;BPL:BANK1
ROMSTAT   .BYTE 0     ;BPL:ROM
;         Switches speichern
START     LDA BANKRD
          STA BANKSTAT
          LDA ROMRD
          STA ROMSTAT
;         1. Zeile
          LDA #FRSTLINE
          STA ZEILE
          DEC ZEILE
;         1. Spalte
LOOP      LDA #FRSTCOL
          STA SPALTE
          DEC SPALTE
NEXT      LDA ZEILE     ;nächste
          JSR ADR
          LDA SPALTE
          LSR A
          BCC OBEN
UNTEN     STA PAGE1
          BCS CHAR
OBEN      STA PAGE2
;         Zeichen ausgeben
CHAR      TAY
          LDA (BASL),Y
          AND #7F
          CMP #20       ;Ctrl?
          BCS NOCTRL    ;nein
          ADC #40       ;normalisieren
NOCTRL    STA PAGE1     ;stets unten
          JSR PWRITE
          INC SPALTE
          LDA SPALTE
          CMP #LASTCOL
          BCC NEXT
;         CR/LF ausgeben
CR        LDA #0D
          JSR PWRITE
LF        LDA #0A
          JSR PWRITE
          INC ZEILE
          LDA ZEILE
          CMP #LASTLINE
          BCC LOOP
;         Switches normalisieren
          LDA ROMSTAT
          BPL EXIT
          LDA BANK2
          LDA BANK2
          LDA BANKSTAT
          BMI EXIT
          LDA BANK1
          LDA BANK1
EXIT      RTS
;         Bascalc
ADR       LDA RDROM
          LDA RDROM
          LDA ZEILE
          JSR BASCALC
          LDA BANK2
          LDA BANK2
          STA SLOTON    ;wichtig
          STA COL80ON   ;wegen Monitor-
          STA COL80ST   ;Flackern!
          RTS
          .END
```

Anzeigenschluß für Ausgabe 2/86 ist am 20. 12. 85

1.4. Ausführung

NEWSTT ($D7D2) – führt den nächsten A/S-Befehl via ↑GOCMD aus (wenn nicht Ctrl-C gedrückt wurde) und arbeitet als Interpreter-Hauptschleife. Nach Ausführung eines Befehls wird immer wieder NEWSTT aufgerufen, bis das Programmende erreicht ist, um dann nach ↑RESTART zu springen. Diese Routine arbeitet auch im Direktmodus und gibt im Trace-Modus die Zeilennummern aus.
Vorher: TXTPTR = Zeiger auf letztes Zeichen vor neu auszuführendem A/S-Befehl.

**GOCMD ($D828) –** führt den dem Token (siehe **Tokentabelle**) entsprechenden A/S-Befehl aus. Liegt kein Token vor, wird versucht, eine „LET"-Anweisung auszuführen, ist er Null, erfolgt ein RTS. Entspricht der Token einem Funktionsaufruf (z.B. „SIN" oder „MID$"), erfolgt ein „SYNTAX ERROR". Die entsprechenden Parameter müssen im Programmtext folgen (z.B. Zeilennummer nach „GOTO").
Vorher: siehe ↑NEWSTT.
Aufruf: Register durch JSR ↑CHRGET setzen.

GOLINE ($D935) – setzt via ↑GOTO die Programmzeiger und springt nach ↑NEWSTT.
Vorher: TXTPTR = Zeiger auf Zeilenangabe.

1.5. Sonstiges

FNDLIN ($D61A) – sucht die angegebene Zeile im Programmtext ab dem Programmanfang.
Vorher: LINNUM ($0050, $0051) = zu suchende Zeilennummer.
Nachher: C = 1, wenn Zeile existiert; C = 0, wenn Zeile nicht gefunden wurde; LOWTR ($009B, $009C) = Zeilenanfang der gefundenen Zeile oder der nächsten, falls nicht gefunden.

FNDLIN1 ($D61E) – sucht die angegebene Zeile im Programmtext ab der vorgegebenen Stelle.
Vorher: LINNUM ($0050, $0051) = zu suchende Zeilennummer; LOWTR ($009B, $009C) = Zeiger auf Programmtext (Anfang des Suchbereichs).
Nachher: siehe ↑FNDLIN.

MAINLST ($D6CC) – listet den angegebenen Bereich und springt nach ↑NEWSTT. Ist das Bereichsende Zeile 0, wird bis zum Programmende gelistet. Ein Abbruch durch Ctrl-C ist möglich, wobei dann zu ↑RESTART gesprungen wird. Der IIc und neue IIe gibt vor jeder Zeile ein Leerzeichen aus.
Vorher: LOWTR ($009B, $009C) = Zeiger auf erste zu listende Zeile; LINNUM ($0050, $0051) = Zeilennummer der letzten zu listenden Zeile.

NEWVAR ($E09C) – legt eine neue (nicht dimensionierte) Variable in der Variablenliste an und trägt den Wert 0 ein (5mal). Dazu werden alle dimensionierten Variablen um 7 Bytes nach oben verschoben.
Vorher: VARNAM ($0081-$0082) = Variablenname (2 Buchstaben).
Nachher: siehe ↑PTRGET.

2. Programmverwaltung

2.1. Initialisierung

2.1.1. Global

**COLDST ($F128) –** führt einen A/S-Kaltstart aus ($E003-Vektor verweist hierher). Zunächst wird die CHRGET-Routine und der Random-Startwert (aufgrund eines Bugs ein Byte zuwenig) in die Zero-Page übertragen. Nach dem Setzen einiger Variablen und Pointer wird dann die Größe des Systems bestimmt. Danach erfolgt ein Sprung zu ↑RESTART.

RESTART ($D43C) – führt einen Warmstart (Soft-Entry) des A/S-Systems durch. Dabei wird „ONERR" gelöscht, Direktmodus eingestellt und eine Eingabezeile geholt, um diese abzuarbeiten. Diese Stelle wird stets dann angesprungen, wenn der Interpreter ein neues Kommando erwartet (Promptzeichen „Ü" oder „]").

SCRTCH ($D64B) – führt „NEW" aus (beinhaltet ↑CLEARC, ↑RESTORE, ↑STKINI und ↑STXTPT).
Vorher: C = 0.
Nachher: TXTPTR = Zeiger auf Programmanfang - 1.

SETPTRS ($D665) – ruft ↑STXTPT auf und geht in ↑CLEARC über.

CLEARC ($D66C) – führt „CLEAR" aus (beinhaltet ↑RESTORE und ↑STKINI).

STKINI ($D683) – initialisiert den Stack und verhindert „CONT".
Nachher: S = $F8.

2.1.2. Einzelne Zeiger

STXTPT ($D697) – setzt TXTPTR auf Programmanfang - 1.
Nachher: TXTPTR = Zeiger auf Programmanfang - 1.

SETDA ($D853) – trägt den übergebenen Zeiger in den DATA-Zeiger ein.
Vorher: A = DATA-Zeiger, LSB; Y = DATA-Zeiger, MSB.
Nachher: DATPTR ($007D, $007E) = übergebener Zeiger.

**VARTIO ($D8F0) –** setzt die Programmlänge und das LOCK-Byte für READ ($FEFD) und WRITE ($FECD). Diese Routine entfällt beim IIc.
Nachher: A1 ($003C, $003D) = $0050 (Bereichsanfang); A2 ($003E, $003F) = $0052 (Bereichsende).

PROGIO ($D901) – setzt die Programmstart- und -endadresse für READ und WRITE (siehe ↑VARTIO). Diese Routine entfällt beim IIc.
Nachher: A1 ($003C, $003D) = TXTTAB ($0067, $0068); A2 ($003E, $003F) = VARTAB ($0069, $006A).

2.2. Programmänderung

NXTLIN ($D45C) – übernimmt die Zeile im Eingabepufffer in den Programmtext und führt „CLEAR" aus. Falls die Eingabezeile nur aus der Zeilennummer besteht, wird diese Zeile gelöscht. Danach erfolgt ein Sprung zu ↑RESTART.
Vorher: TXTPTR = IN - 1 ($01FF); IN ($0200-$02FF) = Eingabezeile im ASCII-Format (Bit 7 = 0) mit $00 als Endmarker.
Aufruf: Register durch JSR ↑CHRGET setzen.

LINKSET ($D4F2) – setzt die Link-Felder im A/S-Text neu. (Wird z.B. von DOS 3.3 nach Laden eines A/S-Programms aufgerufen). Führt „CLEAR" und „RESTORE" aus und springt zu ↑RESTART.
Vorher: TXTTAB ($0067, $0068) = Programmanfang; VARTAB ($0069, $006A) = Programmende + 1.

3. Ein/Ausgabe

3.1. Eingabe

**INCHR ($D553) –** liest ein Zeichen via RDKEY ($FD0C) ein und löscht Bit 7.
Nachher: A = eingelesenes Zeichen (ASCII mit Bit 7 = 0).

**INLINE ($D52C) –** übernimmt eine Eingabezeile via GETLINE ($FD6A) ohne Prompt-Zeichen, löscht alle Bit 7 und setzt $00 als Endmarker an Stelle von Return. Akzeptiert nur 240 Zeichen inkl. Return.
Nachher: IN ($0200-$02FF) = abgelegte Eingabezeile; X = $FF, Y = $01.

INLINE1 ($D52E) – wie ↑INLINE, das Prompt-Zeichen kann jedoch gewählt werden.
Vorher: X = Prompt-Zeichen (ASCII mit Bit 7 = 1).
Nachher: siehe ↑INLINE.

GDBUFS ($D539) – löscht in der Eingabezeile alle Bit 7 und setzt $00 als Endmarker an Stelle von Return.
Vorher: IN ($0200-$02FF) = Eingabezeile; X = Länge der Eingabezeile - 1.
Nachher: siehe ↑INLINE.

3.2. Ausgabe

3.2.1. Einzelne Zeichen

**OUTDO ($DB5C) –** gibt das angegebene Zeichen via COUT ($FDED) aus und berücksichtigt dabei die eigene Ausgabemaske (NORMAL/FLASH/INVERSE). Danach wird eine Warteschleife eingelegt (für „SPEED").
Vorher: A = auszugebendes Zeichen (Bit 7 egal); SPEEDZ ($00F1) = Länge der Warteschleife.
Nachher: A = ausgegebenes Zeichen mit Bit 7 = 0; X =; Y =.

CRDO ($DAFB) – gibt ein Carriage Return via ↑OUTDO aus.

OUTSPC ($DB57) – gibt ein Leerzeichen (Space = $20) via ↑OUTDO aus.

OUTQST ($DB5A) – gibt ein Fragezeichen via ↑OUTDO aus.

3.2.2. Zahlen

LINPRT ($ED24) – gibt die angegebene Zahl (2-Byte-Integer) via ↑STROUT aus.
Vorher: X = auszugebende Zahl, LSB; A = auszugebende Zahl, MSB.

PRNTFAC ($ED2E) – gibt die Zahl in FAC aus.
Vorher: FAC ($009D-$00A2) = auszugebende Zahl.

3.2.3. Zeichenketten

STROUT ($DB3A) – gibt den angegebenen String aus, der durch ↑STRLIT ausgewertet und durch ↑FREFAC wieder freigegeben wird. Der String muß mit $00 oder Hochkomma (Bit 7 = 0) enden. Die Ausgabe erfolgt via ↑STRPRT1.
Vorher: A = Zeiger auf String, LSB; Y = Zeiger auf String, MSB.
Nachher: siehe ↑ STRLIT.

STRPRT1 ($DB41) – gibt den angegebenen String via ↑OUTDO aus.
Vorher: INDEX ($005E, $005F) = Zeiger auf String; X = String-Länge.
Nachher: X = 0; Y = String-Länge.

4. String-Verwaltung

STRCMP ($DF7D) – vergleicht zwei Strings mit der durch CMPFLG angegebenen Operation und trägt das Ergebnis (1 oder 0 für wahr oder falsch) in den Hauptfließkomma-Akku ein. Beide Strings werden freigegeben (↑FREFAC, ↑FRETMP).
Vorher: FAC+3 ($00A0, $00A1) = Zeiger auf 1. Deskriptor; ARG+3 ($00A8, $00A9) = Zeiger auf 2. Deskriptor; CMPFLG ($0016) = Flag für Vergleichsoperation (1: >, 2: =, 3: >=, 4: <, 5: <>, 6: <=).
Nachher: FAC ($009D-$00A2) = 1 oder 0; VALTYP ($0011) = 0.

STRLIT ($E3E7) – bestimmt die Länge eines Strings anhand seines Endmarkers ($00 oder X-Registers). Befindet sich der String-Anfang in der Zero-Page oder dem Eingabepuffer ($0200-$02FF), wird er unter den String-Bereich geschrieben (unterhalb von FRETOP). Danach wird der Deskriptor in den Deskriptor-Stack kopiert, falls dort noch Platz ist (ansonsten „FORMULA TOO COMPLEX").
Vorher: A = Zeiger auf String, LSB; Y = Zeiger auf String, MSB; X = Endmarker; TEMPPT ($0052) = Stackpointer im Deskriptor-Stack.
Nachher: TEMPPT = TEMPPT + 3 = X; TEMPST ($0055-$005D) = neuer Deskriptor; FRETOP ($006F, $0070) = FRETOP - Länge; VALTYP ($0011) = $FF (String); STRNG2 ($00AD, $00AE) = Zeiger auf erstes Zeichen hinter String (falls Endmarker = Hochkomma, wird dieses ebenfalls übergangen).

STRSPA ($E3DD) – legt einen neuen String-Deskriptor an und reserviert den entsprechenden Platz unter den vorhandenen Strings. Falls selbst nach einer Garbage-Collection kein Platz mehr vorhanden ist, erfolgt die Meldung „OUT OF MEMORY„.
Vorher: A = Länge des neu einzurichtenden Strings.
Nachher: A =; FAC ($009D-$009F) = String-Deskriptor; X = String-Adresse, LSB; Y = String-Adresse, MSB.

GETSPA ($E452) – schafft Platz für neuen String, d.h. FRETOP und FRESPC werden um die String-Länge nach unten versetzt. Dabei wird ggf. eine Garbage-Collection ausgeführt. Reicht der Speicherplatz trotz G/C nicht aus, erfolgt „OUT OF MEMORY".
Vorher: A = String-Länge.
Nachher: FRETOP ($006F, $0070) = FRESPC ($0071, $0072) = FRETOP - A; A =.

FRETMP ($E604) – gibt den bezeichneten String frei. Wurde der String bereits im String-Bereich abgelegt, wird er dort wieder gelöscht. Deutet der letzte Eintrag im Deskriptor-Stack auf diesen String, wird dieser Deskriptor ebenfalls vom Stack genommen (↑FRETMS).
Vorher: A = Zeiger auf Deskriptor, LSB; Y = Zeiger auf Deskriptor MSB.
Nachher: X = Zeiger auf String, LSB; Y = Zeiger auf String, MSB; A = String-Länge.

FRESTR ($E5FD) – gibt den letzten String mittels ↑FREFAC frei, prüft jedoch vorher, ob der letzte Ausdruck tatsächlich ein String war. Falls nicht, erfolgt ↑TMMERR.
Vorher: VALTYP ($0011) = $FF (String); ansonsten siehe ↑FREFAC.
Nachher: siehe ↑FRETMP.

FREFAC ($E600) – gibt den bezeichneten String mittels ↑FRETMP frei.
Vorher: FAC+3 ($00A0, $00A1) = Zeiger auf String-Deskriptor.
Nachher: siehe ↑FRETMP.

FRETMS ($E635) – entfernt den obersten Eintrag des Deskriptor-Stacks, falls es sich um den angegebenen String handelt.
Vorher: LASTPT ($0052, $0053) = Zeiger auf letzten Deskriptor; A = Zeiger auf derzeitigen Deskriptor, LSB; Y = Zeiger auf derzeitigen Deskriptor, MSB.
Nachher: Z = 1, falls Deskriptor entfernt wurde.

GARBAG ($E484) – führt eine Garbage-Collection aus. Dabei werden alle nicht mehr referierten Strings (sog. String-Leichen) entfernt und die verbleibenden Strings, d.h. alle durch die Variablenliste und den temporären Deskriptor-Stack bezeichneten Zeichenketten, nach oben verschoben.
Vorher: VARLEN ($008F) = 3.
Nachher: FRETOP ($006F, $0070) = Zeiger auf Start der Strings.

5. Mathematik

5.1. Integer-Mathematik

MULT ($E2AD) – führt eine 16-Bit-Multiplikation durch. Wird das Ergebnis größer als 16 Bits, erfolgt die Meldung „OUT OF MEMORY".
Vorher: STRNG2 ($00AD, $00AE) = Multiplikator; LOWTR ($009B, $009C) = Zeiger auf Multiplikand, durch Y indiziert ((LOWTR),Y); Y = Index auf Multiplikand, MSB.
Nachher: X = Ergebnis, LSB; Y = A = Ergebnis, MSB.

MULT1 ($E2B8) – führt 16-Bit-Multiplikation aus wie ↑MULT.
Vorher: STRNG2 ($00AD, $00AE) = Multiplikator; RESULT+2 ($0064, $0065) = Multiplikant.
Nachher: siehe ↑MULT.

5.2. Fließkomma-Mathematik

5.2.1. Einfache Operationen

ZEROFAC ($E84E) – setzt FAC zu null (nur den Exponenten und das Vorzeichen).
Nachher: FAC ($009D, $00A2) = 0.

FALSE ($DF5D) – trägt 0 in den Hauptfließkomma-Akku ein.
Nachher: FAC ($009D) = 0.

TRUE ($DF60) – trägt 1 in den Hauptfließkomma-Akku ein.
Nachher: FAC ($009D-$00A2) = 1.

RND ($EFAE) – bildet eine Zufallszahl, abhängig von FAC (entspricht „RND").
Vorher: FAC ($009D-$00A2) = 0, für alte Zufallszahl; FAC > 0 für neue Zufallszahl; FAC < 0 für definierte Zufallsfolge; RNDSEED ($00C9-$00CD) = Startwert.
Nachher: FAC = neue Zufallszahl im Intervall [0, 1[.

SGN ($EB90) – trägt das Vorzeichen von FAC in FAC ein.
Nachher: FAC ($009D-$00A2) = -1, wenn FAC < 0; FAC = 0, wenn FAC = 0; FAC = 1, wenn FAC > 0.

INT ($EC23) – schneidet die Nachkommastellen ab (entspricht „INT").
Nachher: FAC ($009D-$00A2) = INT (FAC).

ABS ($EBAF) – bildet den Betrag von FAC.
Nachher: FAC ($009D-$00A2) = ABS (FAC).

5.2.2. Funktionen

SQR ($EE8D) – berechnet die Quadratwurzel von FAC. (Ausführung über SQR (X) = X ↑ 0,5.)
Nachher: FAC ($009D-$00A2) = SQR (FAC).

EXP ($EF09) – exponiert FAC zur Basis e = 2,71828. (Ausführung über Potenzreihe.)
Nachher: FAC ($009D-$00A2) = EXP (FAC).

LOG ($E941) – berechnet den natürlichen Logarithmus von FAC und legt das Ergebnis wieder in FAC ab. Ist das Argument <= 0, erfolgt ↑ILQERR. (Ausführung mittels Potenzreihe.)
Nachher: FAC ($009D-$00A2) = LOG (FAC).

SIN ($EFF1) – berechnet den Sinus von FAC. (Ausführung über Potenzreihe.)
Nachher: FAC ($009D-$00A2) = SIN (FAC) .

COS ($EFEA) – berechnet den Cosinus von FAC. (Ausführung über COS (X) = SIN (X + Pi/2).)
Nachher: FAC ($009D-$00A2) = COS (FAC).

TAN ($F03A) – berechnet den Tangens von FAC. (Ausführung über TAN (X) = SIN (X) / COS (X).)
Nachher: FAC ($009D-$00A2) = TAN (FAC).

ATN ($F09E) – berechnet den Arcustangens von FAC. (Ausführung über Potenzreihe; ist X

> 1, so wird folgende Formel angewandt: ATN (X) = Pi/2 - ATN (1 / X).)
Nachher: FAC ($009D-$00A2) = ATN (FAC).

ODDSER ($EF5C) – berechnet eine ungerade Potenzreihe (a * X + b * X↑3 + c * X↑5 + ...). Dabei müssen die Reihenkoeffizienten (in umgekehrter Reihenfolge) als Tabelle abgelegt sein. Das erste Byte gibt die Anzahl - 1 wieder, darauf folgen die Koeffizienten als 5-Byte-Fließkommazahl. Zunächst wird das Argument X quadriert, dann die Routine ↑SERIES (Einsprung) aufgerufen und das Ergebnis noch einmal mit dem Argument multipliziert (X * (a + b * X↑2 + c * X↑4 ...).
Vorher: siehe ↑SERIES.
Nachher: siehe ↑SERIES.

SERIES ($EF72) – berechnet eine Potenzreihe (a + b * X + c * X↑2 + ...) nach dem Horner-Schema: a + X * (b + X * (c + X * (d ...))). Die Koeffizienten-Tabelle muß wie unter ↑ODDSER beschrieben abgelegt sein.
Vorher: FAC ($009D-$00A2) = Argument X; A = Zeiger auf Koeffizienten-Tabelle, LSB; Y = Zeiger auf Koeffizienten-Tabelle, MSB.
Nachher: FAC = Ergebnis der Potenzreihe.

5.2.3. Verknüpfungen

FPWRT ($EE97) – exponiert ARG um FAC und legt das Ergebnis in FAC ab. (Ausführung über X↑Y = EXP (LOG (X) * Y).)
Vorher: ARG ($00A5-$00A9) = Basis; FAC ($009D-$00A2) = Exponent; Z = 1, wenn FAC = 0.
Nachher: FAC = ARG↑FAC.

FADD ($E7BE) – überträgt die angegebene (gepackte) Zahl nach ARG und geht in ↑FADDT über.
Vorher: A = Zeiger auf Zahl, LSB; Y = Zeiger auf Zahl, MSB.
Nachher: siehe ↑FADDT.

FADDT ($E7C1) – addiert FAC und ARG und legt das Ergebnis in FAC ab.
Vorher: FAC ($009D-$00A2) = 1. Summand; ARG ($00A5-$00A9) = 2. Summand; A = FAC ($009D).
Nachher: FAC = FAC + ARG.

FADDH ($E7A0) – addiert 0,5 zum Hauptfließkomma-Akku.
Nachher: FAC ($009D-$00A2) = FAC + 0,5.

FSUB ($E7A7) – überträgt die angegebene (gepackte) Zahl nach ARG und geht in ↑FSUBT über.
Vorher: A = Zeiger auf Zahl, LSB; Y = Zeiger auf Zahl, MSB.
Nachher: siehe ↑FSUBT.

FSUBT ($E7AA) – subtrahiert ARG von FAC und legt das Ergebnis in FAC ab.
Vorher: FAC ($009D-$00A2) = Subtrahend; ARG ($00A5-$00A9) = Minuend.
Nachher: FAC = ARG - FAC.

FMULT ($E97F) – überträgt die angegebene (gepackte) Zahl nach ARG und geht in ↑FMULTT über.
Vorher: A = Zeiger auf Zahl, LSB; Y = Zeiger auf Zahl, MSB.
Nachher: siehe ↑FMULTT.

FMULTT ($E982) – multipliziert FAC und ARG und legt das Ergebnis in FAC ab.
Vorher: FAC ($009D-$00A2) = Multiplikand; ARG ($00A5-$00A9) = Multiplikator; A = FAC ($009D).
Nachher: FAC = FAC * ARG.

MULT10 ($EA39) – multipliziert FAC mit 10.
Nachher: FAC ($009D-$00A2) = FAC * 10.

FDIV ($EA66) – überträgt die angegebene (gepackte) Zahl nach ARG und geht in ↑FDIVT über.
Vorher: A = Zeiger auf Zahl, LSB; Y = Zeiger auf Zahl, MSB.
Nachher: siehe ↑FDIVT.

FDIVT ($EA69) – dividiert ARG durch FAC und legt das Ergebnis in FAC ab. Ist der Divisor = 0, erfolgt ↑DBZERR.
Vorher: ARG ($00A5-$00A9) = Dividend; FAC ($009D-$00A2) = Divisor; A = FAC.
Nachher: FAC = ARG / FAC.

DIV10 ($EA55) – dividiert FAC durch 10.
Nachher: FAC ($009D-$00A2) = FAC / 10.

OR ($DF4F) – verknüpft den Hauptfließkomma-Akku mit dem zweiten Fließkomma-Akku durch logisches ODER. Ist *eines* der beiden Argumente ungleich 0; wird FAC zu 1.
Vorher: FAC ($009D) = 1. Operand; ARG ($00A5) = 2. Operand.
Nachher: FAC ($009D-$00A2) = 1 oder 0.

AND ($DF55) – verknüpft FAC und ARG durch logisches UND. Sind *beide* Argumente ungleich 0, wird FAC zu 1.
Vorher: siehe ↑OR.
Nachher: siehe ↑OR.

5.2.4. Sonstiges

FCOMP ($EBB2) – vergleicht FAC (ungepackt) mit der angegebenen Zahl (gepackt) im Speicher.
Vorher: A = Zeiger auf zu vergleichende Zahl, LSB; Y = Zeiger auf zu vergleichende Zahl, MSB.
Nachher: A = $FF, wenn FAC < Zahl; A = $00, wenn FAC = Zahl; A = $01, wenn FAC > Zahl; Z = 1, wenn FAC = Zahl.

SIGN ($EB82) – überprüft das Vorzeichen von FAC.
Nachher: A = $FF, wenn FAC ($00A2) < 0; A = $00, wenn FAC ($009D) = 0; A = $01, wenn FAC ($00A2) > 0; Z = 1, wenn FAC = 0.

SIGNIF ($E82E) – normiert FAC, d.h. FAC wird so angepaßt, daß Bit 7 von FAC+1 (erstes Mantisse-Byte) eine 1 enthält (i.allg. Links-Shift). Die unnormierte Form kann bei arithmetischen Operationen auftreten, vermindert jedoch die Genauigkeit. Daher wird FAC nach einigen Operationen unter Einbeziehung des 5. Mantisse-Bytes neu normiert.

RNDB ($EB72) – rundet FAC, d.h. es wird unter Einbeziehung des 5. Mantisse-Bytes eine Mantissenbreite von 32 Bits erzeugt.

ADDACC ($ECD5) – addiert die Integer-Zahl im A-Register zu FAC.
Vorher: A = Summand (0-255).
Nachher: FAC ($009D-$00A2) = FAC + A.

FRE1 ($E2E5) – legt die Anzahl der noch freien Speicherplätze in FAC als Fließkommazahl ab. Vorher wird eine Garbage-Collection durchgeführt (siehe ↑GARBAG).
Vorher: VARLEN ($008F) = 3.
Nachher: FAC ($009D-$00A2) = Anzahl freier Speicherstellen.

6. Konvertieren und Übertragen

6.1. Konvertierung

FIN ($EC4A) – wandelt den durch den Textpointer angegebenen String in eine Fließkommazahl um und legt das Ergebnis in FAC ab.
Vorher: TXTPTR = Zeiger auf erstes Zeichen - 1 der ASCII-Folge mit Bit 7 = 0.
Aufruf: Register durch JSR ↑CHRGET setzen.
Nachher: FAC ($009D-$00A2) = ausgewertete Zahl.

FOUT ($ED36) – wandelt die Zahl in FAC in einen String um (Bit 7 = 0) und legt diesen im Stack (ab $0100) ab.
Vorher: FAC ($009D-$00A2) = umzuwandelnde Zahl.
Nachher: STACK ($0100-$010A) = umgewandelte Zahl als String; A = Zeiger auf Stack, LSB ($00); Y = Zeiger auf Stack, MSB ($01).

VAL ($E707) – wandelt den angegebenen String in eine Zahl in FAC um.
Vorher: siehe ↑GETSTR.
Nachher: FAC ($009D-$00A2) = umgewandelte Zahl.

GIVAYF ($E2F2) – wandelt eine vorzeichenbehaftete 15-Bit-Integer-Zahl in eine Fließkommazahl um.
Vorher: A = zu übertragende Zahl, LSB; Y = zu übertragende Zahl, MSB (Bit 7 = Vorzeichen); C = 1.
Nachher: FAC ($009D-$00A2) = Fließkommazahl; VALTYP ($0011) = 0.

SNGFLT ($E301) – wandelt die Zahl im Y-Register in eine Fließkommazahl um.
Vorher: Y = zu übertragende Zahl.
Nachher: siehe ↑GIVAYF.

QINT ($EBF2) – wandelt FAC in eine 31-Bit-Integer-Zahl mit Vorzeichen um (negative Zahlen im Zweierkomplement). Es wird nicht überprüft, ob FAC in diesem Bereich liegt.
Nachher: FAC ($009E-$00A1) = Integer-Zahl; Bit 7 von FAC ($009E) = Vorzeichen.

CONINT ($E6FB) – wandelt den Hauptfließkomma-Akku in eine Integer-Zahl (0-255) um.
Vorher: FAC ($009D-$00A2) = umzuwandelnde Zahl.
Nachher: siehe ↑GETBYT.

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,– zuzügl. ges. MwSt.

Beispiel:

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

nur DM 17,10 inkl. MwSt.

Musteranzeige gewerblich

Die einspaltige Millimeterzeile (42 mm breit) kostet 5,50 zuzügl. ges. MwSt.

Beispiel:

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

nur DM 137,94 inkl. MwSt.

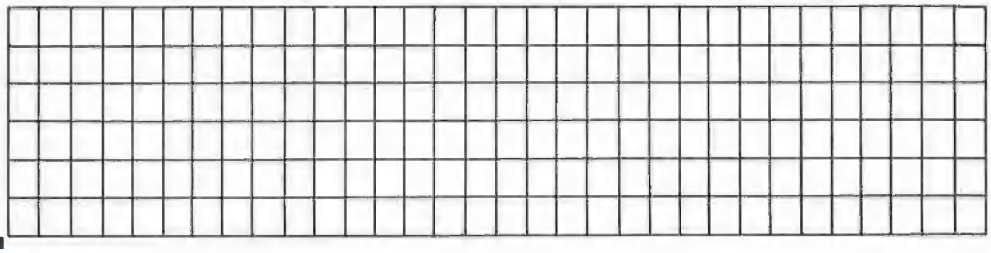
peeker

peeker-Börse

AUFTRAG FÜR KLEINANZEIGEN

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

☐ suche Hardware ☐ suche Software ☐ Tausch ☐ Verschiedenes
☐ biete Hardware ☐ biete Software ☐ Kontakte ☐ Chiffre



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

peeker

Produkt-Karte

Zu der in »**peeker**«, Heft_____, Seite_____ erschienenen
☐ Anzeige ☐ Produktbesprechung

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 Fragen, Ergänzungen oder Anregungen zu Aufsätzen in »**peeker**« haben:

_____

_____

_____

_____

_____

_____

peeker-Börse

Vorname, Name

Beruf

Straße

Wohnort

PLZ

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

ANTWORTKARTE

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

POSTKARTE

Firma

Straße

PLZ/Ort

POSTKARTE

peeker
Redaktion
Postfach 10 28 69
6900 Heidelberg 1

Produkt-Karte

Wünschen Sie weitere Informationen zu einem der im Heft vorgestellten Produkte?

Nichts einfacher als das.
Produkt-Karte ausfüllen, mit 60-Pfennig frankieren und absenden.

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

Damit erleichtern Sie dem Hersteller eine gezielte Beantwortung Ihrer Anfrage.

Zum Schluß tragen Sie auf der Rückseite die genaue Anschrift des Inserenten/Herstellers und Ihre vollständige Firmenanschrift ein.

PEEKER
MAGAZIN FÜR APPLE-COMPUTER

GETADR ($E752) – wandelt den Hauptfließkomma-Akku in eine 2-Byte-Integer-Zahl in LINNUM um.
Vorher: FAC ($009D-$00A2) = umzuwandelnde Zahl.
Nachher: Y = LINNUM ($0050) = umgewandelte Zahl, LSB; A = LINNUM+1 ($0051) = umgewandelte Zahl, MSB.

PARSE ($D56C) – wandelt die BASIC-Schlüsselwörter in der Eingabezeile in Tokens (siehe **Tokentabelle**) um und legt die Eingabezeile neu ab. Der IIc und neue IIe akzeptiert hier auch Kleinbuchstaben.
Vorher: IN ($0200-$02FF) = zu bearbeitende Zeile mit Endmarker $00; X = Zeiger auf erstes Zeichen - 1; Y = $04.
Nachher: IN = umgewandelte Zeile.

6.2. Übertragung

6.2.1. Allgemein

BLTU1 ($D39A) – verschiebt den Bereich von LOWTR bis HIGHTR nach HIGHDS (Ende des Zielbereichs). Der Transfer erfolgt von oben nach unten (Left-Move), daher ist diese Routine nicht geeignet zur Verschiebung nach unten bei Überschneidung von Ziel- und Quellbereich.
Vorher: LOWTR ($009B, $009C) = Quellbereich, Anfang; HIGHTR ($0096, $0097) = Quellbereich, Ende; HIGHDS ($0094, $0095) = Zielbereich, Ende.

6.2.2. Deskriptoren und Strings

STRCPY ($DA9A) – überträgt den angegebenen Deskriptor (3 Bytes) in die durch FORPNT bezeichnete Variable.
Vorher: VPNT ($00A0, $00A1) = Zeiger auf Deskriptor; FORPNT ($0085, $0086) = Zeiger auf Deskriptorfeld der Variablen.

PUTNEW ($E42A) – überträgt den Deskriptor in FAC in den Deskriptor-Stack, falls dort noch Platz vorhanden ist. Ansonsten wird ↑ FTCERR ausgeführt.
Vorher: FAC ($009D-$009F) = Deskriptor; TEMPPT ($0052) = Stackpointer im Deskriptor-Stack.
Nachher: TEMPPT = TEMPPT + 3 = X; TEMPST ($0055-$005D) = neuer Deskriptor; VALTYP ($0011) = $FF (String).

STRINI ($E3D5) – überträgt FAC+3 in den Deskriptor-Pointer, ansonsten siehe ↑ STRSPA.
Vorher: A = Länge des neu einzurichtenden Strings.
Nachher: DSCPTR ($008C, $008D) = FAC+3 ($00A0, $00A1); ansonsten siehe ↑ STRSPA.

MOVINS ($E5D4) – überträgt den angegebenen String mittels ↑ MOVSTR.
Vorher: STRING1 ($00AB-$00AD) = String-Deskriptor; FRESPC ($0071, $0072) = Zeiger auf Zielbereich.
Nachher: siehe ↑ MOVSTR.

MOVSTR ($E5E2) – überträgt den durch die Register angegebenen String und setz FRESPC um die String-Länge nach oben.
Vorher: X = Zeiger auf String, LSB; Y = Zeiger auf String, MSB; A = String-Länge; FRESPC ($0071, $0072) = Zeiger auf Zielbereich.
Nachher: FRESPC = FRESPC + String-Länge.

6.2.3. Fließkommazahlen

CONUPK ($E9E3) – überträgt die angegebene (gepackte) Zahl nach ARG. Diese liegt dann als ungepackte Zahl vor, d.h. das Vorzeichen wird in einem eigenen Byte abgelegt, während in der gepackten Form das Vorzeichen im ersten Mantisse-Byte enthalten ist. Darüber hinaus werden die Vorzeichen von FAC und ARG zu einem Gemeinschaftsvorzeichen verknüpft.
Vorher: A = Zeiger auf zu übertragende Zahl, LSB; Y = Zeiger auf zu übertragende Zahl, MSB.
Nachher: ARG ($00A5-$00A9) = ungepackte Zahl; SGNCPR ($00AB) = Gemeinschaftsvorzeichen; A = FAC ($009D); Z = 1, wenn FAC = 0.

MOVFA ($EB53) – überträgt ARG nach FAC und richtet ein 5. Mantisse-Byte ein.
Nachher: FAC ($009D-$00A2) = ARG ($00A5-$00A9).

MOVAF ($EB63) – überträgt den gerundeten Wert von FAC nach ARG.
Nachher: ARG ($00A5-$00A9) = FAC ($009D-$00A2).

MOVMF ($EB2B) – überträgt den gerundeten Wert (5. Mantisse-Byte) von FAC in den angegebenen Speicherbereich (5 Bytes).
Vorher: FAC ($009D-$00A2) = zu übertragende Zahl; X = Zeiger auf Speicherbereich, LSB; Y = Zeiger auf Speicherbereich, MSB.
Nachher: Z = 1, wenn FAC = 0.

MOVFM ($EAF9) – überträgt die angegebene (gepackte) Zahl nach FAC (siehe ↑ CONUPK). Dabei wird in FAC noch ein 5. Mantisse-Byte angelegt, das die Genauigkeit bei der Auswertung von Polynomen erhöht.
Vorher: A = Zeiger auf Zahl, LSB; Y = Zeiger auf Zahl, MSB.
Nachher: FAC ($009D-$00A2) = übertragene Zahl; EXTRAFAC ($00AC) = 0; Z = 1, wenn Zahl = 0.

MOV2F ($EB1E) – überträgt FAC via ↑ MOVMF in den 3. Hilfsfließkomma-Akku.
Nachher: FAC3 ($0098-$009C) = FAC ($009D-$00A2).

MOV1F ($EB21) – überträgt FAC via ↑ MOVMF in den 2. Hilfsfließkomma-Akku.
Nachher: FAC2 ($0093-$0097) = FAC ($009D-$00A2).

VARL ($DED5) – überträgt den Wert der folgenden Variablen in den Hauptfließkomma-Akku, falls es sich um eine numerische Variable handelt.
Vorher: siehe ↑ PTRGET.
Nachher: siehe ↑ PTRGET.

VARL1 ($DEE9) – überträgt die durch VPNT bezeichnete 16-Bit-Zahl als FP-Zahl in den Hauptfließkomma-Akku.
Vorher: VPNT ($00A0, $00A1) = Zeiger auf Zahl.
Nachher: FAC ($009D-$00A2) = Fließkommazahl.

7. Grafik

7.1. Lores-Grafik

PLOTFNS ($F1EC) – liest zwei Ausdrücke, die durch ein Komma getrennt sind, ein und wertet sie als Integer-Zahlen zwischen 0 und 47 aus. Stimmt der Bereich nicht überein, erfolgt ↑ ILQERR. Beim IIc werden hier Werte zwischen 0 und 79 zugelassen.
Vorher: TXTPTR = Zeiger auf 1. Zeichen des 1. Parameters.
Nachher: X = H2 ($002C) = V2 ($002D) = zweiter Parameter; FIRST ($00F0) = 1. Parameter.

LINCOOR ($F209) – liest zwei Ausdrücke via ↑ PLOTFNS ein, gefolgt von „AT" (ansonsten ↑ SYNERR) und einem weiteren Ausdruck im Bereich 0-47. Ist der erste Parameter größer als der zweite, werden sie vertauscht. Beim IIc werden hier Werte zwischen 0 und 79 zugelassen.
Vorher: siehe ↑ PLOTFNS.
Nachher: FIRST ($00F0) = kleinerer der beiden ersten Parameter; H2 ($002C) = V2 ($002D) = größerer der beiden ersten Parameter; X = 3. Parameter.

**PLOT ($F225) –** zeichnet einen Block auf den Lores-Bildschirm mittels der Monitor-Routine PLOT ($F800).
Vorher: COLOR ($003C) = Farbmaske; TXTPTR = Zeiger auf 1. Zeichen der Koordinatenangabe.

**HLIN ($F232) –** zeichnet eine horizontale Linie auf den Lores-Bildschirm mittels der Monitor-Routine HLINE ($F819).
Vorher: siehe ↑ PLOT.

**VLIN ($F241) –** zeichnet eine vertikale Linie auf den Lores-Bildschirm mittels der Monitor-Routine VLINE ($F828).
Vorher: siehe ↑ PLOT.

7.2. Hires-Grafik

HFNS ($F6B9) – liest die HGR-Koordinaten ein und prüft deren Wertebereich (0-279 bzw. 0-191). Stimmen die Werte nicht, erfolgt ↑ ILQERR.
Vorher: TXTPTR = Zeiger auf 1. Zeichen der X-Koordinate.
Nachher: X = X-Koordinate, LSB; Y = X-Koordinate, MSB; A = Y-Koordinate.

SHPFNS ($F72D) – liest die Shape-Nummer und die (durch „AT" getrennten) Koordinaten via ↑ HFNS ein und setzt die internen Cursor-Koordinaten entsprechend. Wird kein „AT" angetroffen, erfolgt ↑ SYNERR.
Vorher: TXTPTR = Zeiger auf 1. Zeichen der Shape-Nummer; SHPTAB ($00E8, $00E9) = Zeiger auf Shape-Tabelle.
Nachher: SHAPE ($001A, $001B) = Zeiger auf angegebenes Shape; A = ROT-Wert.

HPOS ($F411) – berechnet die internen HGR-Cursor-Werte. Diese umfassen die Basisadresse der Bildschirmzeile, den Spaltenindex des Zeilen-Bytes und den Bit-Index des Spalten-Bits (Bit-Maske).
Vorher: A = Y-Koordinate; X = X-Koordinate, LSB; Y = X-Koordinate, MSB; HPAG ($00E6) = Seite; HCOLOR0 ($00E4) = Basis-Farbmaske.
Nachher: GBAS ($0026, $0027) = Basisadresse; HGRINDX ($00E5) = Spaltenindex; HMASK ($0030) = Bit-Maske; HCOLOR ($001C) = aktuelle Farbmaske.

HFIND ($F5CB) – berechnet aus den internen Cursor-Daten die ursprünglichen X- und Y-Koordinaten. (Diese Routine wird vom A/S-Interpreter nicht benutzt.)
Vorher: siehe ↑ DECRX.
Nachher: HGRX ($00E0, $00E1) = X-Koordinate; HGRY ($00E2) = Y-Koordinate.

HGR2 ($F3D8) – führt „HGR2" aus und geht dabei in ↑ HCLR über.

HGR ($F3E2) – führt „HGR" aus und geht dabei in ↑ HCLR über.

HCLR ($F3F2) – löscht den angegebenen HGR-Bildschirm zu schwarz.
Vorher: HPAG ($00E6) = HGR-Seite ($20 = Seite 1, $40 = Seite 2).
Nachher: HCOLOR ($001C) = aktuelle Farbmaske = $00.

BKGND ($F3F6) – färbt den Bildschirm mit der angegebenen Farbe ein.
Vorher: HPAG ($00E6) = Seite; HCOLOR ($001C) = aktuelle Farbmaske.

HCOLOR1 ($F6EC) – legt die der übergebenen Farbnummer entsprechende Basis-Farbmaske in HCOLOR0 ab. Ist die Farbnummer zu groß, erfolgt ↑ ILQERR.
Vorher: X = Farbnummer (0-7).
Nachher: HCOLOR0 ($00E4) = entsprechende Basis-Farbmaske.

HPLOT ($F457) – ruft ↑ HPOS auf und setzt einen Punkt in der angegebenen Farbe.
Vorher: siehe ↑ HPOS.
Nachher: siehe ↑ HPOS.

HGLIN ($F53A) – zeichnet eine Verbindungslinie zwischen zwei Punkten in der aktuellen Farbe.
Vorher: HGRX ($00E0, $00E1) = X-Koordinate des 1. Punktes; HGRY ($00E2) = Y-Koordinate des 1. Punktes; A = X-Koordinate, LSB des 2. Punktes; X = X-Koordinate, MSB des 2. Punktes; Y = Y-Koordinate des 2. Punktes; interne Cursor-Daten (siehe ↑ DECRX) beziehen sich auf 1. Punkt.
Nachher: HGRX/HGRY = Koordinaten des 2. Punktes; interne Cursor-Daten beziehen sich nun auf 2. Punkt.

DRAW ($F601) – zeichnet ein Shape in angegebener Farbe, Größe und Richtung. Die Anfangskoordinaten sind durch die internen Cursor-Daten bestimmt (siehe ↑ DECRX).
Vorher: X = Zeiger auf Shape, LSB; Y = Zeiger auf Shape, MSB; A = ROT-Wert; SCALE ($00E7) = SCALE-Wert.

DRAW1 ($F605) – zeichnet wie ↑ DRAW ein Shape.
Vorher: SHAPE ($001A, $001B) = Zeiger auf Shape; ansonsten wie ↑ DRAW.

XDRAW ($F65D) – zeichnet ein Shape in der angegebenen Farbe, Größe und Richtung durch Invertierung aller Bit-Punkte mit den auf dem Bildschirm vorhandenen. Die Anfangskoordinaten sind durch die internen Cursor-Daten bestimmt (siehe ↑ DECRX).
Vorher: siehe ↑ DRAW.

XDRAW1 ($F661) – zeichnet wie ↑ XDRAW ein Shape.
Vorher: siehe ↑ DRAW1.

DECRX ($F467) – setzt die internen Cursor-Daten um einen Punkt nach links. War der Punkt am linken Bildschirmrand, erfolgt ein Wrap around zum rechten Rand in derselben Zeile.
Vorher: GBAS ($0026, $0027) = Basisadresse; Y = Spaltenindex; HMASK ($0030) = Bit-Maske; HCOLOR ($001C) = aktuelle Farbmaske.

INCRX ($F48A) setzt die internen Cursor-Daten um einen Punkt nach rechts. War der Punkt am rechten Bildschirmrand (279), erfolgt ein Wrap around zum linken Rand in derselben Zeile.
Vorher: siehe ↑ DECRX.

DECRY ($F4D5) – setzt die internen Cursor-Daten um einen Punkt nach oben. War der Punkt am oberen Bildschirmrand, erfolgt ein Wrap around zum unteren Rand in derselben Spalte.
Vorher: siehe ↑ DECRX.

INCRY ($F504) – setzt die internen Cursor-Daten um einen Punkt nach unten. War der Punkt am unteren Bildschirmrand (191), erfolgt ein Wrap around zum oberen Rand in derselben Spalte.
Vorher: siehe ↑ DECRX.

INVMASK ($F47E) – invertiert die aktuelle Farbmaske (wenn es sich nicht um weiß oder schwarz handelt) für den Übergang von gerader zu ungerader Spaltenzahl oder umgekehrt.
Nachher: HCOLOR ($001C) = invertierte Farbmaske.

8. Prüf- und Fehlerroutinen

8.1. Prüfen

CHKSTK ($D3D6) – überprüft, ob auf dem Stack noch genügend Platz für Rücksprungadressen vorhanden ist (mit 18 Adressen Reserve), ansonsten erfolgt „OUT OF MEMORY".
Vorher: A = Anzahl der erwünschten Adressen = Bytes / 2.

REASON ($D3E3) – überprüft, ob die übergebene Adresse unterhalb von FRETOP ($006F, $0070) liegt (führt ggf. Garbage-Collection aus). Falls nicht, erfolgt „OUT OF MEMORY".
Vorher: A = Adresse, LSB; Y = Adresse, MSB.
Nachher: A =; Y =.

CHKNUM ($DD6A) – überprüft, ob der letzte Ausdruck numerisch war, ansonsten erfolgt ein Sprung zu ↑ TMMERR.
Vorher: VALTYP ($0011) = Flag für Ausdruckstyp ($00 = Zahl, $FF = String).

CHKSTR ($DD6C) – überprüft, ob der letzte Ausdruck ein String war, ansonsten wird ↑ TMMERR durchgeführt.
Vorher: siehe ↑ CHKNUM.

ISLETC ($E07D) – überprüft, ob ein Großbuchstabe vorliegt.
Vorher: A = zu prüfendes Zeichen (Bit 7 = 0).
Nachher: A =; C = 1 bei Buchstabe; C = 0, wenn nicht.

ISCNTC ($D858) – überprüft, ob Ctrl-C gedrückt wurde, und führt dementsprechend ↑ HANDLERR (falls „ONERR" angegeben wurde) oder ↑ STOP aus. Ansonsten erfolgt ein RTS.
Nachher: ERRNUM ($00DE) = $FF, falls Ctrl-C gedrückt wurde.

SYNCHR ($DEC0) – überprüft die Übereinstimmung zwischen angegebenem Zeichen und dem, auf das der Textpointer zeigt. Bei Gleichheit erfolgt ein Rücksprung über ↑ CHRGET, ansonsten die Meldung „SYNTAX".
Vorher: TXTPTR = Zeiger auf zu prüfendes Zeichen; A = erwartetes Zeichen.
Nachher: siehe ↑ CHRGET.

CHKCOM ($DEBE) – überprüft via ↑ SYNCHR, ob ein Komma folgt.

CHKOPN ($DEBB) – überprüft via ↑ SYNCHR, ob eine geöffnete Klammer folgt.

CHKCLS ($DEB8) – überprüft via ↑ SYNCHR, ob eine geschlossene Klammer folgt.

8.2. Fehler

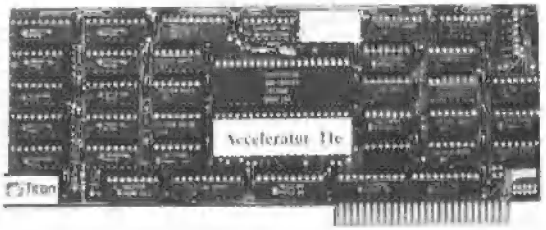
8.2.1. Allgemeine Fehlerbehandlung

ONERR1 ($F2D0) – setzt alle Zeiger für eine Fehlerbehandlung.
Vorher: TXTPTR = Zeiger auf erstes Zeichen der Zeilennummer der Fehlerbehandlung (nach „GOTO"-Token).
Nachher: OERRPNT ($00F4, $00F5) = TXTPTR; OERRLIN ($00F6, $00F7) = CURLIN ($0075, $0076); ERRFLG ($00D8) >= $80: TXTPTR = Zeiger auf 1. Zeichen des nächsten Befehls.

RESUME ($F318) – führt das Programm bei dem fehlerhaften Befehl fort, nachdem eine Fehlerbehandlung stattgefunden hat.
Vorher: ERRLIN ($00DA, $00DB) = Zeilennummer, bei der der Fehler auftrat; ERRPOS ($00DC, $00DD) = Zeiger auf Befehlsanfang; ERRSTK ($00DF) = Stackpointer vor erster Ausführung der fehlerhaften Zeile.

ERROR ($D412) – gibt die dem Fehlercode entsprechende Fehlermeldung aus (siehe **Fehlertabelle**) oder verzweigt zur Fehlerbehandlung (↑ HANDLERR), abhängig von ERRFLG. Nach der Fehlerausgabe erfolgt ein A/S-Warmstart (↑ RESTART).

Accelerator™ IIe macht Ihren Apple® II, II Plus oder IIe dreieinhalbmal schneller.



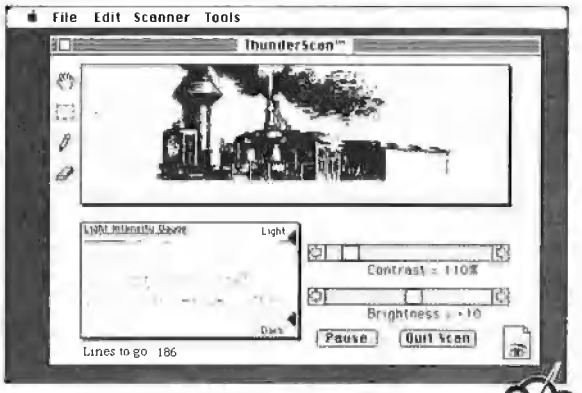
Jetzt laufen VisiCalc®, Apple Writer, PASCAL, BASIC, Datenbanken usw. endlich ohne langen Zeitverlust. Stecken Sie einfach die ACCELERATOR IIe Karte in irgendeinen Slot und beobachten Sie, wir Ihr Apple loslegt!
ACCELERATOR IIe besitzt seinen eigenen schnellen 6502 Prozessor und 80 K-Byte Hochgeschwindigkeitsspeicher, einschließlich einer eingebauten schnellen Sprachkarte und schnellem RAM-Speicherplatz für die ROM-Sprache.
Direkt von Pandasoft (Titan Distributor für Deutschland) oder bei Ihrem Applehändler.

pandasoft Dr.-Ing. Eden

Uhlandstraße 195 · 1000 Berlin 12 · Mo–Fr 10–18 Uhr, Sa 10–13 Uhr
Telefon: 0 30/31 04 23 · Telex: 1 85 859

ThunderScan.™

Ein neues optisches Lesegerät, das beliebige Vorlagen in MacPaint überträgt: Fotos, Zeichnungen, Landkarten und Illustrationen werden in den Apple-Imagewriter eingespannt und von einem Lesekopf, der das Farbband ersetzt, abgetastet.



- 32 Graustufen
- 80 Punkte/cm Auflösung
- Übertragungsmaßstab 25 % – 400 %
- Vorlagen bis 20 x 25 cm
- Nachträgliche Veränderung des Kontrasts und der Helligkeit.

ThunderScan

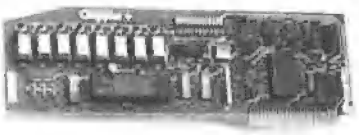
pandasoft Dr.-Ing. Eden

Uhlandstraße 195 · 1000 Berlin 12 · Mo–Fr 10–18 Uhr, Sa 10–13 Uhr
Telefon: 0 30/31 04 23 · Telex: 1 85 859

Orange™ Printer Interface **Druckerinterfaces** für Apple II+/e/c/IIi Interfaces auf dem **neuesten Stand der Technik. Kompatibel** mit allen gängigen Druckern wie: APPLE, EPSON, STAR, NEC, OKIDATA usw. Passende Treibersoftware wird über Dip-Switch ausgewählt.

Grappler + Printer Interface **Grafikfähiges Druckerinterface** das keine Wünsche mehr offen läßt. Über **2 Dutzend Kommandos** ermöglichen die volle Kontrolle über alle Möglichkeiten Ihres Druckers. Jetzt auch mit **IIe Features: Double Hires Grafics** und **80 Zeichen Dump** mittels Druckerpuffer nachrüstbar über Bufferboard.

Buffered Grappler + Besitzt alle Vorzüge des Grappler +, hat aber zusätzlich einen integrierten **16 K Druckpuffer,** der auf **32 oder 64 K aufrüstbar** ist.



Serial Grappler Printer Interface **Serielles Druckerinterface** speziell für den **Apple Imagewriter.**

HOTLINK Seriell-nach-Parallel-Wandler für den IIc im Kabel integriert.

GRAPPLER C wie Hotlink, jedoch zusätzlich Imagewriter Emulation und Grafik Software-Diskette.

pandasoft Dr.-Ing. Eden

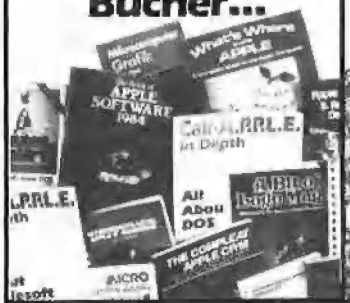
Uhlandstraße 195 · 1000 Berlin 12 · Mo–Fr 10–18 Uhr, Sa 10–13 Uhr
Telefon: 0 30/31 04 23 · Telex: 1 85 859

Sie haben einen Apple...

wir haben die Software...

und die Hardware...

wir haben die Bücher...



und die Zeitschriften*...

***Fordern Sie unseren Gratiskatalog an!**

ALLES FÜR DEN APPLE II+, IIe, IIc UND MACINTOSH

pandasoft Dr.-Ing. Eden

UHLANDSTR. 195 · D-1000 BERLIN 12
TEL.:(030) 310 423 · TELEX: 18 58 59

Autorisierter Apple Fachhandler MICROSOFT Distributor

Ich besitze einen Apple. Bitte schicken Sie mir Ihren kostenlosen Katalog

Name .....

Adresse .....

Vorher: X = Fehlercode; ERRFLG ($00D8) = $00 für Fehlermeldung, ansonsten Sprung nach ↑HANDLERR.

HANDLERR ($F2E9) – setzt die Fehlerbehandlungswerte ein und rettet die Zeiger für ↑RESUME. Danach wird ↑GOTO ausgeführt (Zeilennummer auswerten, siehe ↑ONERR1) und ↑NEWSTT aufgerufen.
Vorher: X = Fehlercode; REMSTK ($00F8) = Stackpointer vor Ausführung der fehlerhaften Anweisung; OERRPNT ($00F4, $00F5) = Zeiger auf Fehlerbehandlungszeile; OERRLIN ($00F6, $00F7) = Zeilennummer der Fehlerbehandlung.
Nachher: ERRNUM ($00DE) = Fehlercode; ERRSTK ($00DF) = Stackpointer vor fehlerhaftem Befehl; ERRLIN ($00DA, $00DB) = Zeilennummer der fehlerhaften Zeile; ERRPOS ($00DC, $00DD) = Zeiger auf fehlerhaften Befehl (durch ↑NEWSTT in OLDTPNT = $0079, $007A gerettet).

ERRMSGE ($D419) – gibt in jedem Fall die dem Fehlercode entsprechende Fehlermeldung aus (siehe Fehlertabelle) und führt dann einen A/S-Warmstart durch (↑RESTART).
Vorher: X = Fehlercode.

INPRT ($ED19) – gibt „ IN " und die in CURLIN abgelegte Integer-Zahl via ↑STROUT aus.
Vorher: CURLIN ($0075, $0076) = auszugebende Zahl.

ERRDIR ($E306) – gibt „ILLEGAL DIRECT" aus, falls kein Programm aktiv ist (Direktmodus).
Vorher: CURLIN+1 ($0076) = $FF bei Direktmodus.
Nachher: A =; Y =.

8.2.2. Die Fehler im einzelnen

BSSERR ($E196) – behandelt „BAD SUBSCRIPT" via ↑ERROR.
DBZERR ($EAE1) – behandelt „DIVISION BY ZERO" via ↑ERROR.
FTCERR ($E430) – behandelt „FORMULA TOO COMPLEX" via ↑ERROR.
ILDERR ($E30B) – behandelt „ILLEGAL DIRECT" via ↑ERROR.
ILQERR ($E199) – behandelt „ILLEGAL QUANTITY" via ↑ERROR.
NWFERR ($DD0B) – behandelt „NEXT WITHOUT FOR" via ↑ERROR.
OODERR ($E1BC) – behandelt „OUT OF DATA" via ↑ERROR.
MEMERR ($D410) – behandelt „OUT OF MEMORY" via ↑ERROR.
OVFERR ($E8D5) – behandelt „OVERFLOW" via ↑ERROR.
RWGERR ($D979) – behandelt „RETURN WITHOUT GOSUB" via ↑ERROR.
SYNERR ($DEC9) – behandelt „SYNTAX" via ↑ERROR.
TMMERR ($DD76) – behandelt „TYP MISMATCH" via ↑ERROR.
UDFERR ($E30E) – behandelt „UNDEF'D FUNCTION" via ↑ERROR.
UNDERR ($D97C) – behandelt „UNDEF'D STATEMENT" via ↑ERROR.

Tabelle der Zero-Page-Variablen (Auszug)

```
VALTYP   - $11 -  17 - Variablentyp: $FF = String
                                     $00 = Zahl
INTFLAG  - $12 -  18 - Integer-Flag: $80 = Integer
                                     $00 = Real
SUBFLAG  - $14 -  20 - Flag für verschiedene Einschränkungen:
                       z.B. keine Integer-Var. bei "FOR"
CMPFLG   - $16 -  22 - Vergleich-Flag: >: 1, = : 2, >=: 3
                                       <: 4, <>: 5, <=: 6
HCOLOR   - $1C -  28 - aktuelle HGR-Farbmaske (wird von
                       Spalte zu Spalte invertiert)
GBAS     - $26 -  38 - Basisadresse einer HGR-Zeile
H2       - $2C -  44 - Endkoordinate bei HLINE ($F819)
V2       - $2D -  45 - Endkoordinate bei VLINE ($F828)
HMASK    - $30 -  48 - Bit-Maske: maskiert Pixel einer
                       HGR-Spalte
INVFLG   - $32 -  50 - AND-Maske für Bildschirmausgabe
COLOR    - $3C -  60 - Farbe für GR (besteht aus 2
                       identischen Nibbles für oberen und
                       unteren Block)
A1       - $3C -  60 - Startadresse bei Tape-Routine
A2       - $3E -  62 - Endadresse bei Tape-Routine
LINNUM   - $50 -  80 - 16-Bit-Integer-Zahl (i.allg.
                       Zeilennummer)
TEMPPT   - $52 -  82 - Deskriptor-Stackpointer
LASTPT   - $53 -  83 - Zeiger auf letzten temporären
                       Deskriptor
TEMPST   - $55 -  85 - Deskriptor-Stack (9 Bytes)
INDEX    - $5E -  94 - allgemeiner Zeiger
RESULT   - $62 -  98 - Ergebnis der letzten Multiplikation
                       oder Division
TXTTAB   - $67 - 103 - Zeiger auf Programmanfang
VARTAB   - $69 - 105 - Zeiger auf Variablenanfang (numerisch)
FRETOP   - $6F - 111 - Ende der String-Variablen
                       (von oben nach unten)
FRESPC   - $71 - 113 - String-Zeiger bei temporären Strings
MEMSIZE  - $73 - 115 - höchste verfügbare Speicherstelle+1
                       ("HIMEM")
CURLIN   - $75 - 117 - Nummer der aktuellen Zeile
                       MSB = $FF bei Direktmodus
OLDLIN   - $77 - 119 - Nummer der letzten Zeile
                       (für Ctrl-C, "RESUME" usw.)
OLDTXT   - $79 - 121 - Zeiger auf letzten Befehl (s.o.)
DATPTR   - $7D - 125 - Zeiger auf aktuelle DATA-Position
VARNAM   - $81 - 129 - letzter Variablenname (2 Buchstaben)
VARPNT   - $83 - 131 - Zeiger auf letzte Variable
FORPNT   - $85 - 133 - allgemeiner Zeiger
CPRTYP   - $89 - 137 - Vergleich-Flag
DSCPTR   - $8C - 140 - Deskriptorenzeiger
VARLEN   - $8F - 143 - Variablenlänge bei Garbage-Collection
FAC2     - $93 - 147 - weiterer FP-Akku neben FAC und ARG
HIGHDS   - $94 - 148 - Ende des Zielbereichs bei Block-Transfer
HIGHTR   - $96 - 150 - Ende des Quellbereichs bei Block-Trans.
FAC3     - $98 - 152 - 4. Fließkomma-Akku
LOWTR    - $9B - 155 - Beginn des Quellbereichs bei Block-Trans.
FAC      - $9D - 157 - Hauptfließkomma-Akku (mit einem
                       zusätzlichen Mantisse-Byte)
FACSIGN  - $A2 - 162 - Vorzeichen von FAC: $00 = positiv
                                           $80 = negativ
ARG      - $A5 - 165 - zweiter Fließkomma-Akku
SGNCPR   - $AB - 171 - gemeinsames Vorzeichen von FAC und ARG
STRING1  - $AB - 171 - Zeiger für String-Verschiebung
EXTRAFAC - $AC - 172 - 6. Mantisse-Byte für FAC
STRNG2   - $AD - 173 - allgemeiner Zeiger bei String-Verwaltung
TXTPTR   - $B8 - 184 - Zeiger auf aktuelle Programmposition
RNDSEED  - $C9 - 201 - Startwert des RND-Generators (5 Bytes)
LOCK     - $D6 - 214 - Auto-RUN-Flag: führt bei $80 jedes
                       Kommando als "RUN" aus
ERRFLG   - $D8 - 216 - ONERR-Flag: führt bei $80
                       Fehlerbehandlung aus
ERRLIN   - $DA - 218 - Nummer der Zeile, bei der ein Fehler
                       auftrat
ERRPOS   - $DC - 220 - Zeiger auf Zeile, bei der ein Fehler
                       auftrat
ERRNUM   - $DE - 222 - Fehlernummer (siehe Fehlertabelle)
ERRSTK   - $DF - 223 - Stackpointer vor Fehlerbehandlung
HGRX     - $E0 - 224 - X-Koordinate für HGR-Routinen
HGRY     - $E2 - 226 - Y-Koordinate für HGR-Routinen
HCOLOR0  - $E4 - 228 - HGR-Farbmaske, auf erste Spalte bezogen
HGRINDX  - $E5 - 229 - HGR-Spaltennummer
                       (Spalte enthält 7 Pixel)
HPAG     - $E6 - 230 - HGR-Seite: $20 = Seite 1
                                  $40 = Seite 2
SCALE    - $E7 - 231 - SCALE-Wert
SHPTAB   - $E8 - 232 - Zeiger auf Shape-Tabelle
FIRST    - $F0 - 240 - PLOT-Parameter (1. Koordinate)
SPEEDZ   - $F1 - 241 - Ausgabeverzögerung
TRCFLG   - $F2 - 242 - Trace-Flag: $80 = Trace-Modus
ORMASK   - $F3 - 243 - OR-Maske für Bildschirmausgabe
OERRPNT  - $F4 - 244 - Zeiger auf ONERR-Zeile
OERRLIN  - $F6 - 246 - Nummer der ONERR-Zeile
REMSTK   - $F8 - 248 - Stackpointer, der vor jedem Befehl
                       gerettet wird
```

Fehlertabelle

Applesoft-Interpreter-Fehlernummern

```
$ØØ (  Ø) - "NEXT WITHOUT FOR"
$1Ø ( 16) - "SYNTAX"
$16 ( 22) - "RETURN WITHOUT GOSUB"
$2A ( 42) - "OUT OF DATA"
$35 ( 53) - "ILLEGAL QUANTITY"
$45 ( 69) - "OVERFLOW"
$4D ( 77) - "OUT OF MEMORY"
$5A ( 9Ø) - "UNDEF'D STATEMENT"
$6B (1Ø7) - "BAD SUBSCRIPT"
$78 (12Ø) - "REDIM'D ARRAY"
$85 (133) - "DIVISION BY ZERO"
$95 (149) - "ILLEGAL DIRECT"
$A3 (163) - "TYPE MISMATCH"
$BØ (176) - "STRING TOO LONG"
$BF (191) - "FORMULA TOO COMPLEX"
$D2 (21Ø) - "CAN'T CONTINUE"
$EØ (224) - "UNDEF'D FUNCTION"
$FE (254) - falsche Eingabe nach INPUT
$FF (255) - Ctrl-C-Unterbrechung
```

(Zur Vervollständigung der Tabelle sind hier auch die Fehlernummern der Disketten-Betriebssysteme aufgeführt)

DOS-3.3-Fehlernummern

```
$Ø1 (  1) - "LANGUAGE NOT AVAILABLE"
$Ø2 (  2) - "RANGE ERROR"
$Ø3 (  3) - "RANGE ERROR"
$Ø4 (  4) - "WRITE PROTECTED"
$Ø5 (  5) - "END OF DATA"
$Ø6 (  6) - "FILE NOT FOUND"
$Ø7 (  7) - "VOLUME MISMATCH"
$Ø8 (  8) - "I/O ERROR"
$Ø9 (  9) - "DISK FULL"
$ØA ( 1Ø) - "FILE LOCKED"
$ØB ( 11) - "SYNTAX ERROR"
$ØC ( 12) - "NO BUFFERS AVAILABLE"
$ØD ( 13) - "FILE TYPE MISMATCH"
$ØE ( 14) - "PROGRAM TOO LARGE"
$ØF ( 15) - "NOT DIRECT COMMAND"
```

ProDOS-BASIC.SYSTEM-Fehlernummern

```
$Ø2 (  2) - "RANGE ERROR"
$Ø3 (  3) - "NO DEVICE CONNECTED"
$Ø4 (  4) - "WRITE PROTECTED"
$Ø5 (  5) - "END OF DATA"
$Ø6 (  6) - "PATH NOT FOUND"
$Ø8 (  8) - "I/O ERROR"
$Ø9 (  9) - "DISK FULL"
$ØA ( 1Ø) - "FILE LOCKED"
$ØB ( 11) - "INVALID PARAMETER"
$ØC ( 12) - "NO BUFFERS AVAILABLE"
$ØD ( 13) - "FILE TYPE MISMATCH"
$ØE ( 14) - "PROGRAM TOO LARGE"
$ØF ( 15) - "NOT DIRECT COMAND"
$1Ø ( 16) - "SYNTAX ERROR"
$11 ( 17) - "DIRECTORY FULL"
$12 ( 18) - "FILE NOT OPEN"
$13 ( 19) - "DUPLICATE FILE NAME"
$14 ( 2Ø) - "FILE BUSY"
$15 ( 21) - "FILE(S) STILL OPEN"
```

Tabelle der Schlüsselwörter (Token)

```
$8Ø (128) - END
$81 (129) - FOR
$82 (13Ø) - NEXT
$83 (131) - DATA
$84 (132) - INPUT
$85 (133) - DEL
$86 (134) - DIM
$87 (135) - READ
$88 (136) - GR
$89 (137) - TEXT
$8A (138) - PR#
$8B (139) - IN#
$8C (14Ø) - CALL
$8D (141) - PLOT
$8E (142) - HLIN
$8F (143) - VLIN
$9Ø (144) - HGR2
$91 (145) - HGR
$92 (146) - HCOLOR=
$93 (147) - HPLOT
$94 (148) - DRAW
$95 (149) - XDRAW
$96 (15Ø) - HTAB
$97 (151) - HOME
$98 (152) - ROT=
$99 (153) - SCALE=
$9A (154) - SHLOAD
$9B (155) - TRACE
$9C (156) - NOTRACE
$9D (157) - NORMAL
$9E (158) - INVERSE
$9F (159) - FLASH
$AØ (16Ø) - COLOR=
$A1 (161) - POP
$A2 (162) - VTAB
$A3 (163) - HIMEM:
$A4 (164) - LOMEM:
$A5 (165) - ONERR
$A6 (166) - RESUME
$A7 (167) - RECALL
$A8 (168) - STORE
$A9 (169) - SPEED=
$AA (17Ø) - LET
$AB (171) - GOTO
$AC (172) - RUN
$AD (173) - IF
$AE (174) - RESTORE
$AF (175) - &
$BØ (176) - GOSUB
$B1 (177) - RETURN
$B2 (178) - REM
$B3 (179) - STOP
$B4 (18Ø) - ON
$B5 (181) - WAIT
$B6 (182) - LOAD
$B7 (183) - SAVE
$B8 (184) - DEF
$B9 (185) - POKE
$BA (186) - PRINT
$BB (187) - CONT
$BC (188) - LIST
$BD (189) - CLEAR
$BE (19Ø) - GET
$BF (191) - NEW
$CØ (192) - TAB(
$C1 (193) - TO
$C2 (194) - FN
$C3 (195) - SPC(
$C4 (196) - THEN
$C5 (197) - AT
$C6 (198) - NOT
$C7 (199) - STEP
$C8 (2ØØ) - +
$C9 (2Ø1) - -
$CA (2Ø2) - *
$CB (2Ø3) - /
$CC (2Ø4) - ↑
$CD (2Ø5) - AND
$CE (2Ø6) - OR
$CF (2Ø7) - >
$DØ (2Ø8) - =
$D1 (2Ø9) - <
$D2 (21Ø) - SGN
$D3 (211) - INT
$D4 (212) - ABS
$D5 (213) - USR
$D6 (214) - FRE
$D7 (215) - SCRN(
$D8 (216) - PDL
$D9 (217) - POS
$DA (218) - SQR
$DB (219) - RND
$DC (22Ø) - LOG
$DD (221) - EXP
$DE (222) - COS
$DF (223) - SIN
$EØ (224) - TAN
$E1 (225) - ATN
$E2 (226) - PEEK
$E3 (227) - LEN
$E4 (228) - STR$
$E5 (229) - VAL
$E6 (23Ø) - ASC
$E7 (231) - CHR$
$E8 (232) - LEFT$
$E9 (233) - RIGHT$
$EA (234) - MID$
```

Tabelle der aufgeführten Routinen

numerisch

```
$ØØB1 -   177 - CHRGET
$ØØB7 -   183 - CHRGOT
$D39A - 5417Ø - BLTU1
$D3D6 - 5423Ø - CHKSTK
$D3E3 - 54243 - REASON
$D41Ø - 54288 - MEMERR
$D412 - 5429Ø - ERROR
$D419 - 54297 - ERRMSGE
$D43C - 54332 - RESTART
$D45C - 54364 - NXTLIN
$D4F2 - 54514 - LINKSET
$D52C - 54572 - INLINE
$D52E - 54574 - INLINE1
$D539 - 54585 - GDBUFS
$D553 - 54611 - INCHR
$D566 - 5463Ø - RUN
$D56C - 54636 - PARSE
$D61A - 5481Ø - FNDLIN
$D61E - 54814 - FNDLIN1
$D64B - 54859 - SCRTCH
$D665 - 54885 - SETPTRS
$D66C - 54892 - CLEARC
$D683 - 54915 - STKINI
$D697 - 54935 - STXTPT
$D6A5 - 54949 - LIST
$D6CC - 54988 - MAINLST
$D7D2 - 5525Ø - NEWSTT
$D828 - 55336 - GOCMD
$D849 - 55369 - RESTORE
$D853 - 55379 - SETDA
$D858 - 55384 - ISCNTC
$D86E - 554Ø6 - STOP
$D87Ø - 554Ø8 - END
$D896 - 55446 - CONT
$D8BØ - 55472 - SAVE
$D8C9 - 55497 - LOAD
$D8FØ - 55536 - VARTIO
$D9Ø1 - 55553 - PROGIO
$D91B - 55579 - RUNLINE
$D921 - 55585 - GOSUB
$D935 - 556Ø5 - GOLINE
$D93E - 55614 - GOTO
$D96B - 55659 - POP
$D979 - 55673 - RWGERR
$D97C - 55676 - UNDERR
```

```
$D995 - 55701 - DATA
$D998 - 55704 - ADDON
$D9A3 - 55715 - DATAN
$D9A6 - 55718 - REMN
$D9C9 - 55753 - IF
$D9DC - 55772 - REM
$D9EC - 55788 - ONGOTO
$DA0C - 55820 - LINGET
$DA46 - 55878 - LET
$DA9A - 55962 - STRCPY
$DAFB - 56059 - CRDO
$DB3A - 56122 - STROUT
$DB41 - 56129 - STRPRT1
$DB57 - 56151 - OUTSPC
$DB5A - 56154 - OUTQST
$DB5C - 56156 - OUTDO
$DD0B - 56587 - NWFERR
$DD67 - 56679 - FRMNUM
$DD6A - 56682 - CHKNUM
$DD6C - 56684 - CHKSTR
$DD76 - 56694 - TMMERR
$DD7B - 56699 - FRMEVL
$DE60 - 56928 - GETVAL
$DE81 - 56961 - STRTXT
$DEB2 - 57010 - PARCHK
$DEB8 - 57016 - CHKCLS
$DEBB - 57019 - CHKOPN
$DEBE - 57022 - CHKCOM
$DEC0 - 57024 - SYNCHR
$DEC9 - 57033 - SYNERR
$DED5 - 57045 - VARL
$DEE9 - 57065 - VARL1
$DF4F - 57167 - OR
$DF55 - 57173 - AND
$DF5D - 57181 - FALSE
$DF60 - 57184 - TRUE
$DF7D - 57213 - STRCMP
$DFE3 - 57315 - PTRGET
$E07D - 57469 - ISLETC
$E09C - 57500 - NEWVAR
$E196 - 57750 - BSSERR
$E199 - 57753 - ILQERR
$E1BC - 57788 - OODERR
$E2AD - 58029 - MULT
$E2B8 - 58040 - MULT1
$E2E5 - 58085 - FRE1
$E2F2 - 58098 - GIVAYF
$E301 - 58113 - SNGFLT
$E306 - 58118 - ERRDIR
$E30B - 58123 - ILDERR
$E30E - 58126 - UDFERR
$E3D5 - 58325 - STRINI
$E3DD - 58333 - STRSPA
$E3E7 - 58343 - STRLIT
$E42A - 58410 - PUTNEW
$E430 - 58416 - FTCERR
$E452 - 58450 - GETSPA
$E484 - 58500 - GARBAG
$E5D4 - 58836 - MOVINS
$E5E2 - 58850 - MOVSTR
$E5FD - 58877 - FRESTR
$E600 - 58880 - FREFAC
$E604 - 58884 - FRETMP
$E635 - 58933 - FRETMS
$E6F5 - 59125 - GTBYTC
$E6F8 - 59128 - GETBYT
$E6FB - 59131 - CONINT
$E707 - 59143 - VAL
$E746 - 59206 - GETNUM
$E752 - 59218 - GETADR
$E7A0 - 59296 - FADDH
$E7A7 - 59303 - FSUB
$E7AA - 59306 - FSUBT
$E7BE - 59326 - FADD
$E7C1 - 59329 - FADDT
$E82E - 59438 - SIGNIF
$E84E - 59470 - ZEROFAC
$E8D5 - 59605 - OVFERR
$E941 - 59713 - LOG
$E97F - 59775 - FMULT
$E982 - 59778 - FMULTT
$E9E3 - 59875 - CONUPK
$EA39 - 59961 - MULT10
$EA55 - 59989 - DIV10
$EA66 - 60006 - FDIV
$EA69 - 60009 - FDIVT
$EAE1 - 60129 - DBZERR
$EAF9 - 60153 - MOVFM
$EB1E - 60190 - MOV2F
$EB21 - 60193 - MOV1F
$EB2B - 60203 - MOVMF
```

```
$EB53 - 60243 - MOVFA
$EB63 - 60259 - MOVAF
$EB72 - 60274 - RNDB
$EB82 - 60290 - SIGN
$EB90 - 60304 - SGN
$EBAF - 60335 - ABS
$EBB2 - 60338 - FCOMP
$EBF2 - 60402 - QINT
$EC23 - 60451 - INT
$EC4A - 60490 - FIN
$ECD5 - 60629 - ADDACC
$ED19 - 60697 - INPRT
$ED24 - 60708 - LINPRT
$ED2E - 60718 - PRNTFAC
$ED36 - 60726 - FOUT
$EE8D - 61069 - SQR
$EE97 - 61079 - FPWRT
$EF09 - 61193 - EXP
$EF5C - 61276 - ODDSER
$EF72 - 61298 - SERIES
$EFAE - 61358 - RND
$EFEA - 61418 - COS
$EFF1 - 61425 - SIN
$F03A - 61498 - TAN
$F09E - 61598 - ATN
$F128 - 61736 - COLDST
$F1EC - 61932 - PLOTFNS
$F209 - 61961 - LINCOOR
$F225 - 61989 - PLOT
$F232 - 62002 - HLIN
$F241 - 62017 - VLIN
$F26D - 62061 - TRACE
$F26F - 62063 - NOTRACE
$F273 - 62067 - NORMAL
$F277 - 62071 - INVERSE
$F280 - 62080 - FLASH
$F28C - 62092 - HIMEM1
$F2AC - 62124 - LOMEM1
$F2D0 - 62160 - ONERR1
$F2E9 - 62185 - HANDLERR
$F318 - 62232 - RESUME
$F331 - 62257 - DEL
$F39F - 62367 - STORE
$F3BC - 62396 - RECALL
$F3D8 - 62424 - HGR2
$F3E2 - 62434 - HGR
$F3F2 - 62450 - HCLR
$F3F6 - 62454 - BKGND
$F411 - 62481 - HPOS
$F457 - 62551 - HPLOT
$F467 - 62567 - DECRX
$F47E - 62590 - INVMASK
$F48A - 62602 - INCRX
$F4D5 - 62677 - DECRY
$F504 - 62724 - INCRY
$F53A - 62778 - GLIN
$F5CB - 62923 - HFIND
$F601 - 62977 - DRAW
$F605 - 62981 - DRAW1
$F65D - 63069 - XDRAW
$F661 - 63073 - XDRAW1
$F6B9 - 63161 - HFNS
$F6EC - 63212 - HCOLOR1
$F72D - 63277 - SHPFNS
$F775 - 63349 - SHLOAD
```

alphabetisch

```
ABS      - $EBAF - 60335
ADDACC   - $ECD5 - 60629
ADDON    - $D998 - 55704
AND      - $DF55 - 57173
ATN      - $F09E - 61598
BKGND    - $F3F6 - 62454
BLTU1    - $D39A - 54170
BSSERR   - $E196 - 57750
CHKCLS   - $DEB8 - 57016
CHKCOM   - $DEBE - 57022
CHKNUM   - $DD6A - 56682
CHKOPN   - $DEBB - 57019
CHKSTK   - $D3D6 - 54230
CHKSTR   - $DD6C - 56684
CHRGET   - $00B1 -   177
CHRGOT   - $00B7 -   183
CLEARC   - $D66C - 54892
COLDST   - $F128 - 61736
CONINT   - $E6FB - 59131
```

```
CONT     - $D896 - 55446
CONUPK   - $E9E3 - 59875
COS      - $EFEA - 61418
CRDO     - $DAFB - 56059
DATA     - $D995 - 55701
DATAN    - $D9A3 - 55715
DBZERR   - $EAE1 - 60129
DECRX    - $F467 - 62567
DECRY    - $F4D5 - 62677
DEL      - $F331 - 62257
DIV10    - $EA55 - 59989
DRAW     - $F601 - 62977
DRAW1    - $F605 - 62981
END      - $D870 - 55408
ERRDIR   - $E306 - 58118
ERRMSGE  - $D419 - 54297
ERROR    - $D412 - 54290
EXP      - $EF09 - 61193
FADD     - $E7BE - 59326
FADDH    - $E7A0 - 59296
FADDT    - $E7C1 - 59329
FALSE    - $DF5D - 57181
FCOMP    - $EBB2 - 60338
FDIV     - $EA66 - 60006
FDIVT    - $EA69 - 60009
FIN      - $EC4A - 60490
FLASH    - $F280 - 62080
FMULT    - $E97F - 59775
FMULTT   - $E982 - 59778
FNDLIN   - $D61A - 54810
FNDLIN1  - $D61E - 54814
FOUT     - $ED36 - 60726
FPWRT    - $EE97 - 61079
FRE1     - $E2E5 - 58085
FREFAC   - $E600 - 58880
FRESTR   - $E5FD - 58877
FRETMP   - $E604 - 58884
FRETMS   - $E635 - 58933
FRMEVL   - $DD7B - 56699
FRMNUM   - $DD67 - 56679
FSUB     - $E7A7 - 59303
FSUBT    - $E7AA - 59306
FTCERR   - $E430 - 58416
GARBAG   - $E484 - 58500
GDBUFS   - $D539 - 54585
GETADR   - $E752 - 59218
GETBYT   - $E6F8 - 59128
GETNUM   - $E746 - 59206
GETSPA   - $E452 - 58450
GETVAL   - $DE60 - 56928
GIVAYF   - $E2F2 - 58098
GLIN     - $F53A - 62778
GOCMD    - $D828 - 55336
GOLINE   - $D935 - 55605
GOSUB    - $D921 - 55585
GOTO     - $D93E - 55614
GTBYTC   - $E6F5 - 59125
HANDLERR - $F2E9 - 62185
HCLR     - $F3F2 - 62450
HCOLOR1  - $F6EC - 63212
HFIND    - $F5CB - 62923
HFNS     - $F6B9 - 63161
HGR      - $F3E2 - 62434
HGR2     - $F3D8 - 62424
HIMEM1   - $F28C - 62092
HLIN     - $F232 - 62002
HPLOT    - $F457 - 62551
HPOS     - $F411 - 62481
IF       - $D9C9 - 55753
ILDERR   - $E30B - 58123
ILQERR   - $E199 - 57753
INCHR    - $D553 - 54611
INCRX    - $F48A - 62602
INCRY    - $F504 - 62724
INLINE   - $D52C - 54572
INLINE1  - $D52E - 54574
INPRT    - $ED19 - 60697
INT      - $EC23 - 60451
INVERSE  - $F277 - 62071
INVMASK  - $F47E - 62590
ISCNTC   - $D858 - 55384
ISLETC   - $E07D - 57469
LET      - $DA46 - 55878
LINCOOR  - $F209 - 61961
LINGET   - $DA0C - 55820
LINKSET  - $D4F2 - 54514
LINPRT   - $ED24 - 60708
LIST     - $D6A5 - 54949
LOAD     - $D8C9 - 55497
LOG      - $E941 - 59713
LOMEM1   - $F2AC - 62124
```

```
MAINLST  - $D6CC - 54988
MEMERR   - $D410 - 54288
MOV1F    - $EB21 - 60193
MOV2F    - $EB1E - 60190
MOVAF    - $EB63 - 60259
MOVFA    - $EB53 - 60243
MOVFM    - $EAF9 - 60153
MOVINS   - $E5D4 - 58836
MOVMF    - $EB2B - 60203
MOVSTR   - $E5E2 - 58850
MULT     - $E2AD - 58029
MULT1    - $E2B8 - 58040
MULT10   - $EA39 - 59961
NEWSTT   - $D7D2 - 55250
NEWVAR   - $E09C - 57500
NORMAL   - $F273 - 62067
NOTRACE  - $F26F - 62063
NWFERR   - $DD0B - 56587
NXTLIN   - $D45C - 54364
ODDSER   - $EF5C - 61276
ONERR1   - $F2D0 - 62160
ONGOTO   - $D9EC - 55788
OODERR   - $E1BC - 57788
OR       - $DF4F - 57167
OUTDO    - $DB5C - 56156
OUTQST   - $DB5A - 56154
OUTSPC   - $DB57 - 56151
OVFERR   - $E8D5 - 59605
PARCHK   - $DEB2 - 57010
PARSE    - $D56C - 54636
PLOT     - $F225 - 61989
PLOTFNS  - $F1EC - 61932
POP      - $D96B - 55659
PRNTFAC  - $ED2E - 60718
PROGIO   - $D901 - 55553
PTRGET   - $DFE3 - 57315
PUTNEW   - $E42A - 58410
QINT     - $EBF2 - 60402
REASON   - $D3E3 - 54243
RECALL   - $F3BC - 62396
REM      - $D9DC - 55772
REMN     - $D9A6 - 55718
RESTART  - $D43C - 54332
RESTORE  - $D849 - 55369
RESUME   - $F318 - 62232
RND      - $EFAE - 61358
RNDB     - $EB72 - 60274
RUN      - $D566 - 54630
RUNLINE  - $D91B - 55579
RWGERR   - $D979 - 55673
SAVE     - $D8B0 - 55472
SCRTCH   - $D64B - 54859
SERIES   - $EF72 - 61298
SETDA    - $D853 - 55379
SETPTRS  - $D665 - 54885
SGN      - $EB90 - 60304
SHLOAD   - $F775 - 63349
SHPFNS   - $F72D - 63277
SIGN     - $EB82 - 60290
SIGNIF   - $E82E - 59438
SIN      - $EFF1 - 61425
SNGFLT   - $E301 - 58113
SQR      - $EE8D - 61069
STKINI   - $D683 - 54915
STOP     - $D86E - 55406
STORE    - $F39F - 62367
STRCMP   - $DF7D - 57213
STRCPY   - $DA9A - 55962
STRINI   - $E3D5 - 58325
STRLIT   - $E3E7 - 58343
STROUT   - $DB3A - 56122
STRPRT1  - $DB41 - 56129
STRSPA   - $E3DD - 58333
STRTXT   - $DE81 - 56961
STXTPT   - $D697 - 54935
SYNCHR   - $DEC0 - 57024
SYNERR   - $DEC9 - 57033
TAN      - $F03A - 61498
TMMERR   - $DD76 - 56694
TRACE    - $F26D - 62061
TRUE     - $DF60 - 57184
UDFERR   - $E30E - 58126
UNDERR   - $D97C - 55676
VAL      - $E707 - 59143
VARL     - $DED5 - 57045
VARL1    - $DEE9 - 57065
VARTIO   - $D8F0 - 55536
VLIN     - $F241 - 62017
XDRAW    - $F65D - 63069
XDRAW1   - $F661 - 63073
ZEROFAC  - $E84E - 59470
```

Richtlinien für Autoren

1. Rechtliches

Der eingereichte Beitrag muß frei von Rechten Dritter sein, d.h. er darf noch nicht anderweitig veröffentlicht oder zur Veröffentlichung angeboten worden sein. Bei Kleinzitaten aus fremden Aufsätzen und/oder Programmen muß die Quelle deutlich vermerkt werden. Bei absichtlich oder fahrlässig verschwiegenen Zitaten liegt die urheberrechtliche Haftung bei Ihnen. Beiträge mit Großzitaten, z.B. kompletten Fremdprogrammen, werden nicht angenommen.
In ein Programm darf kein amerikanischer Copyright-Vermerk und keine Anschrift eingefügt werden. Verwenden Sie statt dessen die deutsche Form „Von Vorname Name, Jahr".
Sie übertragen uns bezüglich Ihres Beitrages das ausschließliche Recht der Vervielfältigung und Verbreitung in jeder Form als Schriftwerk und Datenträger. Bei Programmen bedeutet dies konkret, daß sie nicht nur in Druck-, sondern auch in Diskettenform (Peeker-Sammeldiskette) vertrieben werden. Die Rechtsübertragung findet selbstverständlich erst mit unserer schriftlichen Zusage der Inverlagnahme statt. Solange diese Zusage noch nicht erfolgt ist, kann ein Beitrag jederzeit zurückgezogen werden.

Das Honorar beträgt einheitlich DM 200,- pro Druckseite für Texte, Listings sowie reproduktionsfähige Bildmaterialien, was die Honorierung für die Sammeldiskette einschließt. Von uns erstellte plakative Einstiegsseiten mit Zierbildern usw. werden nicht honoriert. Im Falle eines Reprint-Sammelheftes vergüten wir nochmals DM 50,- pro Druckseite. Bei sehr umfangreichen Beiträgen oder bei Programmen, die aus Platzgründen nur auf die Sammeldiskette aufgenommen werden können, erfolgt die Honorierung nach Vereinbarung.

Bei der Einreichung des Manuskriptes vermerken Sie bitte, ob Sie mit diesen „Richtlinien für Autoren" einverstanden sind. Es genügt, wenn Sie uns den Aufsatz zunächst in Papierform zusenden. Ein Vorspann, aus dem Inhalt und Zweck des Themas hervorgehen, beschleunigt die Prüfung. Zur endgültigen Entscheidung über die Annahme benötigen wir allerdings die kompletten Unterlagen. Im Falle eines Programms spezifizieren Sie bitte sehr genau diejenige Konfiguration (II+/e/c, Nachbau XYZ; FX-80, Imagewriter; DOS 3.3, ProDOS, Pascal 1.2, Turbo 3.0; Videx, 64K-Karte usw.), auf der Sie das Programm persönlich getestet haben. Geben Sie niemals eine Ihnen nicht verfügbare Konfiguration an.

2. Satzverfahren

Alle Aufsatztexte und Programmlistings des Peekers werden komplett auf dem Apple IIe/c für die Satzanlage als kodierte ASCII-Dateien aufbereitet und direkt auf Film belichtet. Wie Sie einem Peeker-Heft unschwer entnehmen können, gibt es zwei Arten von Texten:

Aufsätze
Hierzu benötigt die Satzanlage Texte, bei denen *nur die Absätze* durch Returns abgeschlossen werden, denn die Silbentrennung und das Ausschließen auf Zeilenbreite (= Zeilenumbruch) erfolgt vollautomatisch durch den Satzcomputer. In Aufsatztexten haben die Schriftzeichen unterschiedliche Dicken (= Buchstabenbreiten).

Listings
Hierzu benötigt die Satzanlage Texte, bei denen *alle Zeilen* durch Returns abgeschlossen werden, denn das Silbentrenn- und Ausschließprogramm wird hier vom Satzcomputer nicht aktiviert. In Listings haben die Schriftzeichen gleiche Dicken (= Buchstabenbreiten). Obwohl die Listings auf den ersten Blick wie Schreibmaschinentexte aussehen, werden sie in Wirklichkeit in einer speziellen Druckschrift gesetzt, die ein gestochen scharfes Schriftbild liefert.

Aufsatztexte und Listings sollten als DOS-3.3-formatierte Disketten (35 Spuren) eingereicht werden. Macintosh-Disketten müssen zuvor konvertiert werden. Nur in Sonderfällen, etwa bei Grundsatzaufsätzen branchenfremder Spezialisten, genügt das Papiermanuskript.

2.1. Aufsatz

Für den Aufsatztext benötigen wir einen DOS-3.3-Textfile, der mit einem beliebigen Textverarbeitungsprogramm erstellt werden kann. Da viele Autoren den (zeilenorientierten) Pascal-Editor verwenden, akzeptieren wir auch solche Textfiles; die unerwünschten Returns entfernen wir hier selbst. Für die DOS-Konvertierung von CP/M-, ProDOS- und Pascal-Textfiles können Sie bei Bedarf unsere Konvertierungsprogramme verwenden. Beachten Sie, daß wir in der Redaktion nur den Applewriter besitzen, d.h. wir haben keinen Wordstar usw.
Schreiben Sie den Text so, wie ihn die Satzanlage liebt, d.h. völlig schmucklos ohne Unterstreichungen, ohne Einrückungen von Überschriften, ohne Steuerzeichen für den Matrixdrucker usw. Dies ist im übrigen für Sie auch viel einfacher und schneller. Textstellen, die später halbfett oder kursiv erscheinen sollen, markieren Sie mit Farbstiften auf dem Papierausdruck. Die Kodierung ist dann unsere Angelegenheit. Legen Sie statt dessen Ihr Augenmerk auf Orthographie, Interpunktion, Grammatik, Stilistik, (dezimalklassifikatorische) Gliederung und terminologische Einheitlichkeit.
Überschlägige Umfangsberechnung: Eine Druckseite umfaßt etwa 6.000 Zeichen brutto und unter Berücksichtigung der Zwischenüberschriften und Leerzeilen zwischen größeren Absätzen etwa 5.000 Zeichen netto.

Tabellen und sonstige tabellarische Übersichten sind für uns nach wie vor ein Problem. Beachten Sie folgende Logik: Der Satzcomputer verwandelt bei Aufsatztexten – im Gegensatz zu Listingtexten – jede Folge von *mehreren* Leertasten in *eine* Leertaste, wodurch die Spalten einer Tabelle alle nach links rutschen würden. Wegen der unterschiedlichen Dicken wären nämlich Mehrfach-Leertasten zwecklos. Schreiben Sie deshalb Tabellen so, wie sie aussehen sollen, d.h. mit mehrfachen Leertasten zwischen den Spalten, und speichern Sie sie als gesonderte Textfiles ab. Diese Tabellen werden dann von uns entweder mit Tabulatoren kodiert oder in Schreibmaschinenschrift belichtet.

Technische Zeichnungen legen Sie so maßstabsgetreu wie möglich an. Sie werden von uns jedoch im Atelier umgezeichnet, so daß gut lesbare Freihandzeichnungen genügen.

2.2. Listing

Für die Listings benötigen wir lediglich die Programmdiskette, da wir die Aufbereitung für die Satzanlage selbst vornehmen. Programme mit englischen Menü-Texten und Kommentaren müssen leider zurückgewiesen werden.
Listings werden von uns normalerweise zweispaltig gesetzt (maximal 65 Zeichen/Zeile), gelegentlich dreispaltig (maximal 39 Zeichen/Zeile). Aus Platzgründen können Listings mit mehr als 65 Zeichen/Zeile in der Regel nicht mehr angenommen werden.
Überschlägige Umfangsberechnung: Eine Spalte enthält maximal 90 Zeilen, beim üblichen zweispaltigen Satz also maximal 180 Zeilen.

Für die verschiedenen Programmiersprachen gilt:
Basic-Programme: Verstecken Sie keine Ctrl-Zeichen in REM-Zeilen usw., denn diese müssen von uns dann manuell entfernt werden. Verwenden Sie wegen der beim Satz automatisch vorgenommenen Freistellung der Zeilennummern nur solche mit gleicher Stellenanzahl (bei kleinen Programmen Renumber ab Zeile 10, bei mittelgroßen ab Zeile 100, bei großen ab Zeile 1000). Falls möglich, bitte REM-Kommentare und Menü-Texte in Großkleinschrift.
Assembler-Programme: Schreiben Sie am besten mit einem 40-Z/Z-Editor, da dann einschließlich der Opcodes eine Zeilenbreite von maximal 65 Zeichen/Zeile erreicht werden kann. Wir bevorzugen den Big-Mac- bzw. Merlin-Assembler, weil wir hierzu spezielle Programme für das Assemblieren auf Diskette (statt auf den Drucker) entwickelt haben. Im Interesse der Kompatibilität mit anderen Assemblern sollten Macros vermieden werden. Kommentare möglichst in Großkleinschrift, alles übrige in Großschrift.
Pascal-Programme: Auch für Pascal-Programme gilt eine Zeilenbreite von maximal 65 Zeichen/Zeile. Falls möglich, bitte eckige und geschweifte Klammern verwenden und statt dessen auf Umlaute verzichten. Bei Programmen für Anfänger Schlüsselwörter bitte in Großbuchstaben schreiben.

CP/M-56K-RAM-Disk unter Turbo-Pascal

Ergänzung zum Aufsatz „Apple-CP/M“ aus Peeker, Heft 6/1985

von Bernd Eichinger-Wieschmann

Das nachfolgende Programm soll dem Turbo-Pascal-Besitzer eine einfache und unkomplizierte Installierung des RAM-Disk-Drivers aus dem o.g. Aufsatz ermöglichen. Das Turbo-Programm RAMDISK-.PAS erzeugt einen COM-File, der aus dem CP/M-Betriebssystem heraus mit
> RAMDISK Return
direkt gestartet werden kann und eine RAM-Disk „C:“ auf der 64K-Karte des Apple IIe anlegt.
Es wurden von mir keine Veränderungen auf maschinensprachlicher Ebene vorgenommen. Daher sind die in dem Artikel von K.-W. Bott getroffenen Aussagen nach wie vor gültig. Insbesondere beachte man, daß der Driver nur für CP/M-2.2-56K, also nicht für 60K- oder 64K-CP/M-Versionen, gedacht ist, weil die eine Bank der unteren Language-Card in den RAM-Disk-Speicher mit einbezogen wird.

Von den Listings wurde nur der Hexcode, jeweils die zweite Spalte im Listing, übernommen und in Turbo-Pascal eingebettet. Für einen erfahrenen Programmierer dürfte es kein Problem sein, dieses Pascal-Programm direkt mit einem Debugger in einen Assembler-File umzuwandeln. Mir ging es vielmehr darum, die Möglichkeiten eines modernen Compilers aufzuzeigen. In den meisten „Hochsprachen“ ist es nämlich normalerweise schwierig oder gar unmöglich, Maschinenprogramme in den Quelltext einzubinden. Turbo-Pascal macht hier trotz der Inline-Prozedur keine Ausnahme. Es bietet dafür aber andere Befehle, die diese Einschränkung wieder wettmachen.

Typisierte Konstanten

Und nun zum Programm selbst: Zunächst wird der gesamte Hexcode in jeweils getrennten Feldern abgelegt. Da in Turbo-Pascal die sog. *typisierten Konstanten* implementiert sind, bereitet dies keine Schwierigkeiten. Darunter sind Variablen zu verstehen, die nach „CONST“ definiert und mit einem bestimmten Wert initialisiert werden.

Der *erste Grund* für den Einsatz der typisierten Konstanten liegt in der Syntax: Da sie im Vereinbarungsteil stehen müssen, sind am Anfang eines Programms oder einer Prozedur schon wichtige Variablen bekannt. Ein Beispiel zeigt das Programm **TEST1**. Beim Programmstart wird der typisierten booleschen Konstanten WAHR der Wert TRUE zugewiesen. Daher hat sie schon beim Start einen bekannten Inhalt und muß diesen nicht erst im eigentlichen Programm erhalten.
Der *zweite Grund* ist die Einsparung von Programmcode: Während TEST1 (im RAM compiliert) eine Größe von 116 Bytes einnimmt, bringt es **TEST2** auf 119 Bytes. Bei einer großen Zahl von Zuweisungen macht sich dies bemerkbar.

Da es sich bei den Feldinhalten von RamInit, AuxDriver, MoveDriver und Formatter um Objektcode handelt, müssen diese als Hexwerte in den Feldern abgelegt werden, also mit führendem „$“.
Nun folgt die einzige Prozedur des Programms. Die Prozedur „Brun“ startet ein 6502-Programm, das an einer beliebigen Stelle im Speicher steht. An „Adresse“ wird die Startadresse des auszuführenden Programms übergeben. Da es in Turbo-Pascal keinen Assembler gibt, muß diese Routine zum Starten als Inline-Prozedur geschrieben werden. Inline akzeptiert nur Hexzahlen oder – wie in diesem Fall – typisierte Konstanten. Da an die Prozedur der Wert „Adresse“ nicht als VAR-Parameter übergeben werden kann, erhält „Adresse“ nur einen Zeiger auf den tatsächlichen Inhalt. Der Inhalt, auf den „Adresse“ zeigt, wird nun dem Z80-Register HL zugewiesen. Der Rest entspricht dem SWITCHDEMO von K.-W. Bott (Peeker 6/85, S. 57).

Das Hauptprogramm

Zunächst wird der Bildschirm gelöscht und eine Meldung ausgegeben, um dem Anwender zu zeigen, daß die RAM-Disk als Laufwerk „C:“ installiert wird. Nun werden mittels „Move“ die entsprechenden Felder an ihren Platz im RAM „geschoben“. Da zwischen CP/M- und DOS-Adressen ein Versatz von $1000 Bytes besteht, müssen die Adressen entsprechend angepaßt werden (s. Peeker 6/85, S. 56).
Mit Brun werden nun diese „Felder“ gestartet. Das letzte Feld wird, da es ein CP/M-Programm darstellt, mit der letzten Inline-Prozedur ab 0100h gestartet, wie dies bei CP/M-Programmen üblich ist. Da aber auch der Turbo-Runtime-Kern ab 0100h beginnt, wird dieser zwangsläufig zerstört. Dies ist nicht weiter tragisch, da RamInit von sich aus zu CP/M und seinem CCP zurückkehrt.

Compilierung von RAMDISK.PAS

Nach dem Starten von Turbo-Pascal und der Eingabe des Listings von RAMDISK-.PAS wähle man über „O“ für Compiler-Optionen „C“ für COM-File (= direkte Compilierung auf Diskette). Den fertigen COM-File starte man dann aus dem CP/M-Betriebssystem heraus mit
> RAMDISK Return.

Kurzhinweise

1. Zweck:
Installierung einer RAM-Disk unter CP/M
2. Konfiguration:
IIe mit erweiterter 80-Zeichenkarte;
CP/M 2.20 56K mit Turbo-Pascal 2.0 oder 3.0
3. Test:
RAMDISK.PAS direkt auf Diskette compilieren und später unter CP/M mit
RAMDISK
starten.
4. Sammeldisk:
RAMDISK.PAS
Vorkodierter Quelltext zur Übertragung mit APDOS:
a) APDOS von Drive A starten
b) Sammeldisk in Drive B einlegen
c) RAMDISK.PAS=RAMDISK.PAS eingeben
d) Danach unter Turbo-Pascal direkt compilierfähig

TEST 1

```
Program Testl;

Const Wahr: Boolean = True; {typisierte Konstante}
Var   Ch:   Char;
Begin
  Repeat
    Write ('Weiter J/N ');
    Readln (Ch);
      If Ch = 'N'
        Then Wahr := Not Wahr;
  Until Not Wahr;
End.
```

TEST 2

```
Program Test2;

Var Wahr: Boolean;
    Ch:   Char;
Begin
  Wahr := True;
  Repeat
    Write ('Weiter J/N ');
    Readln (Ch);
      If Ch = 'N'
        Then Wahr := Not Wahr;
  Until Not Wahr;
End.
```

RAMDISK.PAS

```
Program RAMDISK;

{Installiert RAM-Disk unter CP/M als Drive C:}
{Die Felder entsprechen den Routinen aus Heft 6/85, S. 55 ff.}

Const

RamInit: Array [1..173] of Byte =

($21,$83,$DA,$22,$5D,$DA,$11,$83,$DA,$01,$9E,$01,$26,$0F,$7C,$B7
,$CA,$1B,$01,$25,$0A,$12,$03,$13,$C3,$0E,$01,$11,$03,$FE,$3E,$4C
,$12,$13,$3E,$70,$12,$13,$3E,$0C,$12,$11,$B8,$F3,$3E,$03,$12,$11
,$70,$FC,$01,$44,$01,$26,$59,$7C,$B7,$CA,$00,$00,$25,$0A,$12,$03
,$13,$C3,$37,$01,$AD,$83,$C0,$AD,$AE,$FE,$C9,$02,$F0,$0A,$08,$78
,$20,$10,$0E,$AD,$81,$C0,$28,$60,$8D,$09,$C0,$AD,$83,$C0,$AD,$83
,$C0,$4C,$00,$FF,$8D,$08,$C0,$AD,$81,$C0,$60,$A5,$CE,$A6,$CF,$A4
,$06,$8C,$C8,$0C,$A4,$07,$8D,$08,$C0,$85,$CE,$86,$CF,$84,$07,$AD
,$C8,$0C,$85,$06,$AD,$8B,$C0,$AD,$8B,$C0,$A0,$00,$B1,$CE,$91,$06
,$88,$D0,$F9,$A9,$00,$8D,$EA,$03,$AD,$81,$C0,$60,$00,$FF,$20,$00
,$03,$07,$00,$3F,$00,$1F,$00,$80,$00,$00,$00,$00,$00);

AuxDriver: Array [1..228] of Byte =

($A9,$00,$85,$CE,$85,$06,$AD,$E0,$03,$8D,$E3,$FF,$20,$68,$FF,$AD
,$C0,$FF,$D0,$06,$8E,$EA,$03,$4C,$4F,$FF,$48,$20,$C1,$FF,$AD,$EB
,$03,$6A,$B0,$34,$68,$8D,$02,$C0,$8D,$05,$C0,$85,$07,$A9,$08,$85
,$CF,$A0,$00,$AD,$E3,$FF,$C9,$0F,$D0,$09,$8D,$02,$C0,$8D,$04,$C0
,$4C,$97,$0C,$B1,$CE,$91,$06,$88,$D0,$F9,$A9,$00,$8D,$EA,$03,$8D
,$02,$C0,$8D,$04,$C0,$4C,$90,$0C,$68,$8D,$03,$C0,$8D,$04,$C0,$85
,$CF,$A9,$08,$85,$07,$4C,$31,$FF,$A2,$01,$A9,$00,$8D,$BF,$FF,$AD
,$E0,$03,$C9,$0B,$10,$0E,$0A,$0A,$0A,$0A,$09,$08,$18,$6D,$E1,$03
,$8D,$C0,$FF,$60,$C9,$0F,$F0,$10,$C9,$0E,$F0,$19,$C9,$0D,$F0,$10
,$C9,$0C,$F0,$07,$C9,$0B,$D0,$1E,$8E,$BF,$FF,$A9,$D0,$4C,$7C,$FF
,$A9,$E0,$4C,$7C,$FF,$AD,$E1,$03,$C9,$0F,$D0,$05,$A9,$03,$4C,$80
,$FF,$A9,$F0,$4C,$7C,$FF,$A9,$00,$8D,$BF,$FF,$8D,$C0,$FF,$60,$00
,$00,$AD,$C0,$FF,$29,$F0,$C9,$D0,$F0,$01,$60,$AD,$BF,$FF,$F0,$0C
,$A9,$00,$8D,$BF,$FF,$AD,$8B,$C0,$AD,$8B,$C0,$60,$AD,$83,$C0,$AD
,$83,$C0,$60,$00);

MoveDriver: Array [1..52] of Byte =

($8D,$09,$C0,$A9,$00,$85,$00,$A9,$90,$85,$01,$A9,$00,$85,$02,$A9
,$FF,$85,$03,$AD,$8B,$C0,$AD,$8B,$C0,$AE,$33,$95,$A0,$00,$B1,$00
,$91,$02,$88,$D0,$F9,$E6,$01,$E6,$03,$CA,$D0,$F2,$8D,$08,$C0,$AD
,$81,$C0,$60,$01);

Formatter: Array [1..117] of Byte =

($8D,$05,$C0,$A9,$08,$85,$CF,$A9,$00,$85,$CE,$A2,$B8,$20,$66,$90
,$A9,$03,$85,$CF,$A9,$00,$85,$CE,$A2,$01,$20,$66,$90,$AD,$8B,$C0
,$AD,$8B,$C0,$A9,$D0,$85,$CF,$A9,$00,$85,$CE,$A2,$10,$20,$66,$90
,$AD,$8B,$C0,$AD,$8B,$C0,$20,$3F,$90,$AD,$83,$C0,$AD,$83,$C0,$8D
,$09,$C0,$A9,$D0,$85,$CF,$A9,$00,$85,$CE,$A2,$10,$20,$66,$90,$A9
,$E0,$85,$CF,$A9,$00,$85,$CE,$A2,$1F,$20,$66,$90,$8D,$04,$C0,$AD
,$81,$C0,$8D,$08,$C0,$60,$A0,$00,$A9,$E5,$91,$CE,$88,$D0,$FB,$E6
,$CF,$CA,$D0,$F6,$60);
```

```
Procedure Brun (Adresse: Integer);
 {ruft Unterprogramm ab Adresse auf }
Begin
Inline ($2A/Adresse/  {LD HL,(Adresse)}
       $22/$D0/$F3/   {LD (Vec),HL}
       $2A/$DE/$F3/   {LD HL,(CPU)}
       $77);          {LD (HL),A}
End;

Begin
   Clrscr;
   Writeln
    ('Pseudo-Disk wird als Laufwerk C: installiert');

   Move (AuxDriver, Mem [$8000],228);
       {AuxDriver nach $9000, DOS}
   Move (MoveDriver, Mem [$8500],52);
       {MoveDriver nach $9500, DOS}
   Brun ($9500);
       {Run MoveDriver}
   Move (Formatter, Mem [$8000],117);
       {Formatter nach $9000, DOS}
   Brun ($9000);
       {Run Formatter}
   Move (RamInit,Mem [$100],173);
       {RamInit nach 0100h, CP/M}
   Inline ($C3/$00/$01);
       {JP 0100h startet RamInit}
       {Kann nicht mehr zurueck!}
End.
```

Kleine Mikrocomputer-Chronik 1964–1985

zusammengestellt von Ulrich Stiehl

Die nachfolgende Chronik beschreibt die wichtigsten Ereignisse der Mikrocomputer-Ära unter besonderer Berücksichtigung der Apple-Computer-Entwicklung. Die zeitliche Zuordnung ist nicht immer problemlos, da zwischen der Ankündigung und dem Erscheinen eines Mikrocomputers Monate, bei den Prozessoren sogar Jahre verstreichen können. Hinzu kommt noch der „Time-Lag" zwischen Amerika und Europa (in der Regel sechs Monate). Beispielsweise ist der 65816 angeblich bereits 1983 entwickelt worden. Doch bis Mitte 1985 hatte offenbar noch keine Firma diesen Prozessor eingesetzt.

1964

BASIC: Die Programmiersprache „BASIC" („Beginner's All-purpose Symbolic Instruction Code"), die später zur meistbenutzten Mikrocomputersprache werden soll, wird von John G. Kemeny und Thomas E. Kurtz für Großanlagen entwickelt.

1968

Pascal: Die Programmiersprache „Pascal", die später zur wichtigsten Mikrocomputersprache für den Ausbildungsbereich werden soll, wird von Niklaus Wirth in Anlehnung an Algol konzipiert und 1970 erstmals auf einem Großrechner implementiert.

1971

4004: Intel entwickelt den 4-Bit-Prozessor „4004". Ähnliche 4-Bit-Prozessoren findet man heute noch in vielen Industrie- und Haushaltsgeräten, z.B. in Waschmaschinen, Taschenrechnern usw.

1973

8080: Intel konstruiert den 8-Bit-Prozessor „8080", der später als Vorläufer zum Z80 gelten wird.

1974

CP/M: Gary Kildall konzipiert als Berater von Digital Research die Urversion von „CP/M" (= „Control Program for Microprocessors"), eines der in den nachfolgenden Jahren meistbenutzten Betriebssysteme auf 8080- bzw. Z80-Basis.

1975

6502/Z80: In diesem Jahr entstehen die zwei wirtschaftlich wichtigsten 8-Bit-Prozessoren, nämlich der „Z80" von Zilog und der „6502" von MOS Technology, einer von ehemaligen Motorola-Mitarbeitern gegründeten Firma, die später von Commodore übernommen werden wird. Von Motorola stammt in jener Zeit der wenig bedeutsame „6800" als Vorläufer zum späteren „68000".
Allgemein sei hier angemerkt, daß das Schicksal der verschiedenen Prozessoren selten richtig prognostiziert wird. Als einer der besten 8-Bit-Prozessoren gilt beispielsweise der „6809", doch wird er fast nie eingesetzt (Ausnahme z.B. „Dragon"). Ein ähnliches Schicksal ereilt den 16-Bit-Prozessor „Z8000" von Zilog, der praktisch nie verwendet wird.

KIM: Ferner werden in dieser Zeit die ersten Bausätze bzw. Einplatinencomputer („Kits") konstruiert, z.B. der berühmte „KIM" (= „Keyboard Input Monitor") auf 6502-Basis. Das Nachfolgemodell „AIM" ist noch heute von Rockwell erhältlich.

Mikrocomputer-BASIC: Bill Gates, der Begründer von Microsoft, schreibt 1975 den ersten BASIC-Interpreter für Mikrocomputer. Die Firma Microsoft wird in der Folgezeit zum größten Programmiersprachen- und Betriebssystem-Lieferanten für die Mikrocomputer-Industrie.

Während das Integer-BASIC des alten Apple II noch von Steve Wozniak selbst entwickelt wird, ist „Applesoft" ein um Hires-Routinen erweitertes Microsoft-BASIC, das etwa Mitte 1978 zunächst als „Applesoft I" und später als „Applesoft II" auf Kassette, Diskette oder Sprachkarte geliefert werden wird. Was man heute als „FPBASIC" von der System-Master-Diskette her kennt, ist bereits mit der ROM-Applesoft-Version des II+/e identisch, also nicht mit dem in den unteren 48K-Bereich einzuladenden alten „Applesoft II" zu verwechseln.

1976

Apple I: Eine völlig unbekannte „Garagenfirma" namens Apple wird von Steve Jobs und Steve Wozniak im April 1976 gegründet. Wozniak, der das College verlassen und vorübergehend bei Hewlett Packard gearbeitet hat, ist gerade 25 Jahre alt. Der etwas jüngere Steve Jobs, der kurze Zeit bei Atari angestellt gewesen ist, hat ebenfalls nie sein Studium beendet und ist mehr am Geldverdienen denn an der Technik interessiert. Jobs gilt als „Antreiber" und Wozniak als „Arbeiter".

Ende 1975 erblickt der 6502-Bausatz-Computer namens „Apple I" das Licht der Welt. Es werden jedoch nur einige hundert Exemplare verkauft, da Jobs vorausschauend auf eine professionellere Weiterentwicklung mit Gehäuse und Tastatur drängt. Die Garagenfirma Apple bietet den bereits gestandenen Firmen HP und Atari die Entwicklung eines Personal-Computers an: ohne Erfolg, denn niemand erahnt damals den kometenhaften Aufstieg von Apple.

TI-59: Von Texas Instruments wird der „TI-59" entwickelt, der in den nachfolgenden Jahren zu dem meistbenutzten programmierbaren Taschenrechner werden wird (bis zur Produktionseinstellung 1983). Ähnlich erfolgreich ist der HP-41 von Hewlett Packard. Neben diesen anspruchsvollen wissenschaftlich-technischen „Pocket-Computern" macht die seit 1976 einsetzende Flut billiger Taschenrechner (u.a. von Commodore und später namentlich von japanischen Produzenten wie Sharp usw.) dem Rechenschieber, der Logarithmentafel und anderen Rechenhilfsmitteln den Garaus. Die ersten Mikrocomputer-Zeitschriften sind noch überwiegend den programmierbaren Taschenrechnern gewidmet. Das „Kopfrechnen" wird zu einer nur noch von wenigen Schülern beherrschten Fähigkeit verkümmern. Dafür wird in der Schule die „Mengenlehre" eingeführt, von der man sich jedoch Ende 1984 wieder distanziert.

5,25-Disk-Drive: Shugart entwickelt das erste 5,25-Zoll-Disk-Drive (für „flexible Platten" = „floppy discs"), das später die Grundlage der Apple-Diskettenlaufwerke werden soll. Die noch heute üblichen „DISK II"-Laufwerke (140K) erscheinen jedoch erst Mitte 1978. Die Urversion des dazugehörigen Betriebssystems heißt „DOS 3" bzw. „DOS 3.1" und wird von Steve Wozniak und Randy Wiggington entwickelt. Wozniak konstruiert den Controller und schreibt die Nibble-Routinen, während der File-Manager von Wiggington stammt. Die letzte und noch heute übliche Version „DOS 3.3" er-

scheint im Herbst 1980. Anfang 1984 wird „DOS 3.3" durch „ProDOS" ersetzt werden, ein UNIX-ähnliches Betriebssystem mit hierarchischer Directory-Struktur.

1977

Apple II: Im April wird der 6502-Rechner „Apple II" in dem noch heute üblichen beige Plastikgehäuse mit Integer-BASIC im ROM in der Grundversion mit 4K RAM (!), jedoch bereits ausbaufähig auf 48K RAM, vorgestellt. Als Datenspeicher dient ein externer Kassettenrecorder, als Drucker ein ausgedienter Fernschreiber.
Die Entwicklung des Apple II fällt in die zweite Jahreshälfte von 1976. Zu diesem Zeitpunkt wird auch Mike Markkula, der zuvor Marketing-Manager bei Intel gewesen ist, als kaufmännischer Leiter eingestellt, der nach seiner Pensionierung von John Sculley, vormals Manager in der Getränke-Industrie (Pepsi-Cola), Ende 1983 abgelöst werden wird.
Zu den technischen Helfern der ersten Stunde gehören Allen Baum (u.a. Monitor-ROM, Slot-Konzept), Randy Wiggington (u.a. DOS 3.3), Rod Holt (u.a. Netzteil) und Chris Espinosa, der das „Red Manual" und später das „Technical Reference Manual" schreiben wird.
Vom Apple II und dem späteren Apple II+ (Mitte 1978) werden bis Mitte 1983 insgesamt über 1 Million Exemplare verkauft worden sein. Nimmt man noch den Apple IIe (Anfang 1983) und den Apple IIc (Mai 1984) hinzu, so gilt der Apple II in all seinen Spielarten als der einzige Mikrocomputer, der die ersten 10 Jahre der Mikrocomputer-Ära mit hohem Verkaufserfolg überdauert, was auf sein ungewöhnlich flexibles Konzept zurückzuführen ist.

PET: Fast gleichzeitig mit dem Apple II wird der „PET" von Commodore – ebenfalls ein 6502-Mikrocomputer – vorgestellt. Beim PET ist der Kassettenrecorder direkt eingebaut. Weitverbreitete Nachfolgemodelle zum PET sollten später die Rechner der 4000er und 8000er Serie werden.
Andere Mikrocomputer der ersten Stunde stammen von den Firmen Ohio Scientific („OSI", 1977), North Star („Horizon", 1977), Tandy Radio Shack („TRS-80", 1977) und Exidy („Sorceror", 1978), von denen man jedoch inzwischen – mit Ausnahme von TRS – nichts mehr hört.

Dreiergestirn: Verallgemeinert kristallisiert sich in der Anfangszeit der Mikrocomputer-Ära das zunächst gleichwertige Dreiergestirn Apple-Tandy-Commodore heraus. Commodore hat jedoch bald Schwierigkeiten auf dem amerikanischen Markt und verlagert deshalb vorübergehend seine Aktivitäten nach Europa, wo es allmählich zum Marktführer wird. Wegen einer verfehlten Modellpolitik gerät Tandy ins Hintertreffen und muß in den USA schon früh die Marktführerschaft an Apple abtreten, das seine Führungsrolle bis zum Eintritt von IBM in den PC-Markt behauptet.
Für das aktuelle Bilanzjahr 1984/85 gilt in den USA die Reihenfolge
1. IBM
2. Apple
3. Commodore,
während in Deutschland die umgekehrte Reihenfolge gilt
1. Commodore
2. IBM
3. Apple.

68000: Motorola beginnt 1977 mit der Entwicklung des 16-Bit-Prozessors „68000", der jedoch erst 1979 auf den Markt kommen und später in zahlreichen Home- und Personal-Computern eingesetzt werden wird (z.B. Amiga, Macintosh).

1978

Apple II+: In Verbindung mit den Mitte 1978 ausgelieferten Disk-II-Laufwerken wird der Apple II mit einem „Autostart-ROM" versehen, das zum automatischen Booten von DOS erforderlich ist. Außerdem wird anstelle von Integer-BASIC das Applesoft-BASIC im ROM geliefert. Der im übrigen unveränderte Apple II wird in „Apple II+" oder „Apple II Plus" umbenannt.

UCSD-Pascal: Es werden erste Mikrocomputer-Versionen von „UCSD-Pascal" konzipiert, z.B. Apple-Pascal 1.0, das dann 1979 erscheint und kurz darauf durch Version 1.1 sowie 1983 durch Version 1.2 abgelöst werden wird. Überhaupt hat Apple bei mehreren Mikrocomputer-Programmiersprachen die Nase vorn, z.B. bei Logo (1982).

Wordstar: Micropro arbeitet an einem CP/M-8080-Programm namens „Wordmaster", später „Wordstar" genannt, das zu dem meistbenutzten Textverarbeitungsprogramm der nächsten Jahre werden sollte. Für den Apple selbst wird jedoch der „Applewriter" von Paul Lutus zum bekanntesten Textprogramm, dessen Urversion 1.0 im Jahre 1978 entsteht und später durch die Versionen 1.1 und 2.0 abgelöst werden wird.

MX-80: Die japanische Firma Epson kündigt den Drucker „MX-80" an. Dieses Modell und die Weiterentwicklungen MX-82, FX-80 usw. werden sich in der Folgezeit zu den meistgekauften Matrixdruckern entwickeln, wie überhaupt japanische Firmen den Druckermarkt beherrschen werden.

8086: Von Intel wird der 16-Bit-Prozessor „8086" angekündigt, der später in zahlreichen „MS-DOS"-Rechnern (MS = Microsoft) der gehobenen Preisklasse Verwendung finden wird.

1979

Visicalc: Dan Bricklin von Software Arts (später Visicorp) entwickelt ein innovatives Programm namens „Visicalc", zunächst für den Apple II und danach für eine Vielzahl anderer Mikrocomputer. Visicalc gilt als eines der genialsten und im übrigen wirtschaftlich erfolgreichsten „Tabellenkalkulationsprogramme". Alle nachfolgenden „Visi-Clones" wie „Multiplan" von Microsoft usw. sind nur Nachahmungen und Erweiterungen der ursprünglichen Tabellenidee. Visicorp wird später (Mai 1985) aus „Marktbereinigungsgründen" von Lotus aufgekauft und dann aufgelöst werden.

dBASE: Wayne Ratliff entwickelt ein CP/M-Z80-Programm namens „Vulcan", das später unter dem noch heute üblichen Namen „dBASE II" von Ashton-Tate vermarktet wird und sich als eines der wichtigsten Datenbankprogramme für Mikrocomputer erweisen sollte. Inzwischen gibt es dBASE II und III für eine Vielzahl von Rechnern und Betriebssystemen (beim Apple II unter CP/M).

1980

ZX80: Sinclair kündigt den Z80-Rechner „ZX80" an. Dieses Modell und die Nachfolgemodelle ZX81, Spectrum usw. machen als Billigst-Homecomputer die Mikrocomputertechnik auch für Schüler erschwinglich. 1985 gerät Sinclair, nachdem der neue 68008-Rechner „QL" („Quantum Leap") angekündigt worden ist, in finanzielle Schwierigkeiten und wird von einem britischen Zeitungsverleger übernommen.

VC20: Eine noch größere Verbreitung als der ZX80 erreicht der ebenfalls 1980 vorgestellte 6502-Rechner „VC20" von Commodore, von dem bis Anfang 1983 bereits über 1 Million Exemplare verkauft sein werden. Mit diesem Mikrocomputer und dem späteren „C64" zementiert Commodore seine Vorherrschaft auf dem Hobby- und Homecomputer-Sektor.

Apple III: Der 6502-Rechner „Apple III", der im Mai 1980 vorgestellt wird, soll das Nachfolgemodell zum Apple II werden. Anfängliche Hardware-Mängel führen zu umfangreichen Umtauschaktionen. Danach wird der Apple III nur noch wenig gekauft und im Herbst 1984 eingestellt.

Z80-Softcard: Microsoft entwickelt die „Z80-Softcard" für den Apple II sowie eine angepaßte CP/M-Version (in Sublizenz von Digital Research). Ferner entwickelt Videx eine 80-Zeichenkarte namens „Videoterm" für den nach wie vor ohne 80-Zeichendarstellung ausgelieferten Apple II+. Da nunmehr auch CP/M-Programme wie dBASE und Wordstar „gefahren" werden können, erlebt der Apple II in der Folgezeit seine größte Wachstumsperiode. Gleichzeitig beginnen „Nachbauer" aktiv zu werden. Zunächst werden nur Zusatzkarten „abgekupfert", später der Apple-II-Computer selbst.

1981

Wozniak-Unfall: Im Februar hat Steve Wozniak einen Flugzeugabsturz mit seiner eigenen Sportmaschine und zieht sich nach einem längeren Krankenhausaufenthalt aus dem Tagesgeschäft bei Apple zurück und nimmt vorübergehend ein wirtschaftswissenschaftliches Studium auf.

Osborne 1: Adam Osborne, der jahrelang unter seinem Namen Bücher schrieb und schreiben ließ, entwickelt den ersten tragbaren Mikrocomputer namens „Osborne 1" auf Z80-Basis. Nach anfänglichen Erfolgen muß die Firma jedoch im Herbst 1983 Konkurs anmelden. Zu diesem Zeitpunkt gerät auch die Firma Sirius, deren gleichnamiger Mikrocomputer von „Chip" noch als „Computer des Jahres" apostrophiert wird, in wirtschaftliche Bedrängnis.

Ikonen und Maus: Xerox entwikkelt im Palo Alto Research Center (= PARC) einen Rechner namens „Star", der wohlgemerkt kein PC, sondern ein Computer in der 200.000-DM-Preisklasse ist. Neuartig am „Star" ist die „Benutzerschnittstelle", die sich durch die Verwendung symbolhafter Bilder („Ikonen") und eines Zeigegerätes („Maus") auszeichnet. Damit soll dem EDV-unkundigen Laien der Umgang mit der EDV-Anlage erleichtert werden. Diese Bildersprache wird später die Grundlage des Finder- und des GEM-Betriebssystems werden. Den Finder wird man in den Computern Lisa und Mac von Apple antreffen, und den wesentlich später entwickelten bzw. „abgekupferten" GEM (= Graphics Environment Manager) wird man in Rechnern der Firma Digital Research finden, die dann im Herbst 1985 wegen Plagiats in die Zahlung von Lizenzen an die Firma Apple einwilligt.

PC: IBM steigt im Spätherbst 1981 mit dem 8088-Mikro „PC" in das Mikrocomputer-Geschäft ein, nachdem zuvor Microsoft das „MS-DOS" (später PC-DOS) als Auftragsproduktion für IBM entwikkelt hat. Obwohl der „PC" technisch so schlicht wie sein Name ist, wird er rasch vom kommerziellen Anwender (Büros in Industriefirmen) akzeptiert. Bis Ende 1984 gibt es dann noch ein Kopf-an-Kopf-Rennen zwischen IBM und Apple auf dem amerikanischen Markt. 1985 setzte sich IBM jedoch klar als Marktführer mit neuen Modellen wie „PC-XT" (noch 8088) und „PC-AT" (80286) durch. Neben dem „Apple II" gehört der „PC" zu den am meisten nachgebauten Mikrocomputern.

1982

Franklin-Prozeß: Franklin bringt einen Computer namens „Ace" als einen Apple-II-Nachbau heraus und verwickelt sich in der Folgezeit in einen fast zweijährigen Urheberrechtsprozeß mit der Firma Apple wegen der Kopie des Monitor-ROMs und zahlt schließlich nach einem außergerichtlichen Vergleich 2,5 Mill. Dollar Schadenersatz. Danach verschwindet Franklin von der Bildfläche.

Basis-Prozeß: Ähnlich unerquicklich ist die Situation für die Firma Basis in Münster, die ursprünglich Alleinauslieferer für Apple in Deutschland ist und infolge eines Urheberrechtsprozesses in Sachen Micropro-Basis wegen einer exorbitanten Schadenssumme Konkurs anmelden muß.

C64: Commodore bringt als Nachfolgemodell zum VC20 den „C64" heraus, der besonders in Europa eine große Verbreitung erreichen wird.

Integrierte Software: Lotus kündigt „1-2-3" für den IBM-PC an, das als integriertes Programmpaket quasi Visicalc, Visiplot und Visifile in einem darstellt. In den darauffolgenden Jahren werden weitere integrierte Software-Pakete (z.B. 1984 „Symphony") als „Universallösungen" auf den Markt gebracht. Man wird jedoch im Jahre 1985 erkennen, daß Programme, die „alles in einem" bieten, sich für „alle und keinen" eignen.

1983

Lisa: Apple stellt anläßlich der Jahreshauptversammlung im Januar den 68000-Rechner „Lisa" mit Maus und Ikonen als die größte „Revolution" in der Mikrocomputer-Geschichte vor. Der hohe Preis (anfangs ca. 30.000,- DM, später ca. 18.000,-) und die fehlende Großrechner-Kompatibilität schreckt jedoch viele Firmen vom Kauf ab, so daß die Lisa Anfang 1985 wieder aus dem Produktionsprogramm genommen werden wird.

Apple IIe: Anfang 1983 erscheint außerdem der „Apple IIe" (e = extended), der sich gegenüber dem Apple II und II+ durch eine Verbesserung des Motherboards, eine DIN-Tastatur und eine 80-Zeichenkarte auszeichnet, so daß nunmehr Textverarbeitung ohne zusätzliche Hardware möglich ist.

MSX: In Verbindung mit japanischen Firmen kündigt Microsoft das sog. „MSX-Standard-BASIC" für Homecomputer auf Z80-Basis an. Doch weder in den USA noch in Europa wird dieser Standard übernommen werden (MSX = Microsoft Super Extended).

65816: Western Design Center entwickelt den 16-Bit-Prozessor „65816", doch geht er erst im Herbst 1985 in Produktion. Wegen der Kompatibilität zum 6502 wird der 65816 sowie die zum 6502 pin-kompatible Spezialversion 65802 für alte 6502-Rechner noch große Bedeutung erlangen.

Turbo-Pascal: Borland kündigt „Turbo-Pascal" für den IBM-PC und später für CP/M-Rechner an. Infolge des ungewöhnlich niedrigen Preises erlangt es rasch eine große Verbreitung. Microsoft muß lernen, daß gute Programmiersprachen auch billig sein können.

1984

Macintosh: Im Januar stellt Apple den 68000-Rechner „Macintosh" als abgemagerte Version der bereits kränkelnden Lisa vor. Wegen des „geschlossenen" Konzepts (zunächst keine Slots, keine technische Dokumentation und keine apple-eigenen Programmiersprachen) erscheinen anfangs nur wenige fremde Soft- und Hardware-Produkte, und der Verkauf bleibt erheblich hinter den Erwartungen zurück. Nach einem Jahr kündigt sich ein Preisverfall an: Der „Fat Mac" von Ende 1985 ist billiger als der „Thin Mac" von Anfang 1984. Universitäten werden zu Sonderpreisen beliefert, weshalb der Macintosh dort mehr Verbreitung findet als im ursprünglich anvisierten Bürobereich. Ende 1985 umfaßt der deutsche Mac-Club als Ableger der AUGE etwa 150 Mitglieder. Die Zahl der Macintosh-Käufer selbst ist allerdings erheblich größer, doch muß man nicht nur der Firma Apple, sondern auch der Presse anlasten, daß noch bei keinem anderen Gerät derart „großkotzig" übertrieben wurde.

Apple IIc: Im Mai 1984 wird der 65C02-Mikrocomputer „Apple IIc" (c = compact), der sich vom Apple IIe durch das Fehlen der Slots auf der einen und die Integrierung verschiedener Zusatzkarten auf der anderen Seite auszeichnet, in einer von vollem Optimismus geprägten Werbeshow präsentiert. Dieses „Mir-gehört-die-Welt"-Gebaren („aut malum aut nihil") äußert sich noch bis zum Jahresende auf verschiedenen Messen. So wird etwa anläßlich der Orgatechnik-Messe in Köln anstelle eines Großstandes gleich eine ganze Halle gemietet.

1985

Apple-Querelen: Nicht das Orwell-Jahr 1984, sondern das Jahr 1985 wird für die meisten Mikrocomputer-Hersteller zum Horror-Trip. Von den zwei- bis dreistelligen Zuwachsraten der vergangenen Jahre verwöhnt, fahren jetzt viele Firmen satte Verluste ein. Der Pleite-Geier kreist über dem Silicon-Valley. Nach dem ersten Quartal häufen sich die Meldungen über Kurzarbeiten, Entlassungen, Betriebsstillegungen und Konkurse.
Die Firma Apple, die 1985 keinen neuen Computer herausbringt, wird besonders gebeutelt (mehrere Werksschließungen). Steve Wozniak, der seit seinem Flugzeugunglück keinen leitenden Posten mehr übernommen hatte, verläßt wegen Meinungsverschiedenheiten über die Weiterentwicklung des Apple II im Februar die Firma Apple. Steve Jobs wird Mitte des Jahres von John Sculley seiner Entscheidungsbefugnisse für die Macintosh-Division enthoben und verkündet im September sein Ausscheiden aus der Firma Apple. Jobs gründet noch Ende September eine „Gegenfirma" namens „Next" („The Next Apple"), die Konkurrenzprodukte für den Universitätsbereich entwickeln will. Wie seinerzeit gegen den Nachbauer Franklin (1982) kündigt Apple ein Gerichtsverfahren gegen Jobs an, nunmehr jedoch wie bei einem Scheidungsprozeß „in Sachen Apple gegen Apple".

Amiga/512 ST: Ende 1985 erregen zwei neue, ungewöhnlich preiswerte 68000-Mikrocomputer in der 2000-3000-DM-Preisklasse Aufsehen, nämlich der „Amiga" von Commodore und der „512 ST" von Atari. Ob man diese riskante Tiefstpreispolitik durchstehen wird, muß das Jahr 1986 erweisen.

Und 1986?

Für 1986 werden zwei Neuheiten erwartet:
1. Nachfolgemodell zum Apple IIe auf 65816- bzw. 65802-Basis mit partieller Kompatibilität zum alten Apple IIe.
2. Nachfolgemodell zum Macintosh unter Wiederbelebung des Slot-Konzepts.

DAS APPLE II HANDBUCH

Die rasche Orientierung für APPLE IIplus, IIe, IIc

Schnelle Antwort auf Alltagsfragen am APPLE II – leicht nachzuschlagen, praxisbezogen, für IIplus, IIe, IIc in nur 1 Buch!
Unterschiede IIplus/IIe, DOS 3.3/ProDOS, E/A-Interfacekarten/Ports, 40/80 Zeichendarstellung, US/DTS-Tastatur, 48K/128K Systeme etc.
Grafik/Soundmöglichkeiten, eine der APPLE-Stärken, in stark erweiterter Beschreibung.
Kurzführer „Steckkartenerweiterungen" mit Fotos; „Sofortbetrieb von Disketten/Cassettengeräten"; „Druckerbetrieb"; „Direktbefehle"; „Tastaturbedienung" etc.

Backgroundwissen: BASIC für Beginner/Professionelle; MC/BASIC-Kombination; MC-Entwicklung mit MONITOR/MINIASSEMBLER; APPLE-PASCAL-BS; Disketten/Plattenspeicherung; Dateiformate etc.
Ausführlicher Anhang zu Editor, Speicherbelegung, Codes des APPLESOFT-Interpreters etc.

DAS APPLE II HANDBUCH für IIplus, IIe, IIc, 472 Seiten, Softcover, DM 66,–

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

Weiterführende Literatur...

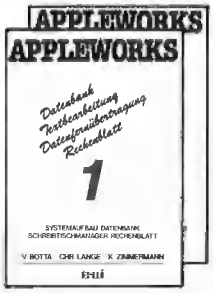
Reparaturanleitung Computer: Apple II, IIplus NEU
Einzigartige Serviceunterläge für Reparaturen und Entwicklungsarbeiten am Apple II. Enthält Schaltpläne, Bauteile- und Vergleichstypenliste; Prüfpunkte mit Oszillogrammen der Signalformen, Logiktabellen, Spannungsangaben; schnelle Servicetests; Anleitung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

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 für zu Hause und im Lehrbereich.
384 Seiten, A4, DM 59,–

NEU

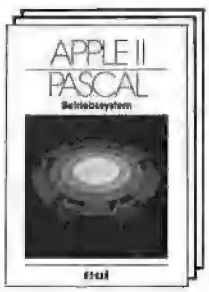
APPLE II – Bewegte 3D-Graphik
(Phil Cohen)
Selbstentworfene Graphiken und Diagramme – animiert oder als Standbilder – eben oder räumlich: alle erforderlichen BASIC-Programme mit Erklärung finden Sie in diesem Buch.
200 Seiten, Softcover, DM 49,–

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,–



NEU

APPLE WORKS + APPLE II, IIe, IIc = Elektronischer Schreibtischmanager
Dieses Programmsystem vereinigt die Funktion Texterstellung, Datenarchivierung, Formblattkalkulation, Datenfernübertragung. Das System mit den höchsten Verkaufsziffern. Sämtliche System/Anwendungsfragen in 2 Bänden. Beispiele aus der Wirtschaft u. v. m.
je 264 Seiten, je DM 49,–



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 4439

Noch im Programm:
6502 – Programmieren in Assembler DM 59,–
VisiCalc, 50 Programme auf Diskette, DM 79,–
Computer für Kinder, APPLE II, DM 29,80

In Vorbereitung:
Macintosh Programmier-Handbuch mit Microsoft BASIC 2.0 DM 59,–

Pascal-Leserbriefe

Zu dem Ergebnis des *Pascal-Preisausschreibens* (s. Heft 9/85, S. 58) sowie in diesem Zusammenhang zu Pascal selbst ging eine wahre Flut von Briefen ein, von denen wir nachfolgend eine Auswahl abdrucken. Allgemein möchte ich anmerken, daß der Peeker in Zukunft mehr Aufsätze über Pascal veröffentlichen wird, und zwar sowohl Grundlagenbeiträge wie der Einführungskurs in diesen Heft als auch Spezialbeiträge in der Art der „Tips und Tricks"-Serie von Dieter Geiß, die großen Zuspruch gefunden hat.
Ihre Reaktion auf das Preisausschreiben zwingt mich, Ihnen reinen Wein einzuschenken. Von den seinerzeit eingereichten Lösungen war *keine einzige Lösung korrekt*. Die meisten Programme hatten bereits die einfachsten Tests nicht überstanden. Aber auch die Programme von U. Allgeier und G. Müller laufen nicht völlig einwandfrei. Das Programm von U. Allgeier funktioniert nicht unter der 64K-Version von Pascal 1.2, und das Programm von G. Müller bringt den Editor gelegentlich zum Absturz, wenn man den konvertierten Textfile einlädt. Weder das Allgeier- noch das Müller-Programm nehmen zudem auf die Blockorganisation des Pascal-Editors Rücksicht, so daß man einen konvertierten Textfile normalerweise nicht direkt – ohne den Umweg über das vorherige Einladen in den Editor – compilieren kann. (Den gleichen Fehler hat auch das GETDOS-Programm von Dieter Geiß aus Heft 1/1985, S. 70.)
Nun stelle man sich einmal plastisch mein Dilemma vor: Hätte ich im September-Heft verkündet, daß kein einziger Gewinner ermittelt werden konnte, so hätten mir wohl einige Leser „Schiebung" unterstellt. Also mußte ich wohl oder übel wenigstens zwei „Lösungen" für die Hauptgewinne auswählen. Und wenn ich verärgert die Allgeier-„Lösung" auf 1/4 Druckseite zusammengequetscht habe, dann deshalb, weil ein nicht völlig fehlerfreies Programm keinen größeren Raum beanspruchen darf.
Noch einen anderen Punkt, der in den nachfolgenden Leserbriefen aus Ihrer Sicht mit Recht kritisiert wird, muß ich aus einer ganz anderen Perspektive hier ansprechen. Mir war und ist bewußt, daß die Kürze des Quelltextes kein sinnvolles Kriterium ist. Ich hatte jedoch noch den Berg mit den über 250 Primzahl-Lösungen des damaligen Wettbewerbs lebhaft in Erinnerung, der mich 14 Tage lang in Beschlag genommen hatte mit dem „Erfolg", daß das Januar-Februar-Heft zum Doppelheft umgemünzt werden mußte. Bei meiner extremen Arbeitsüberlastung mußte eine *inhaltliche* Prüfung der Pascal-Quelltexte von vornherein ausgeschlossen werden, so daß ich also auch nicht gemerkt hätte, ob in einem beliebig langen Listing irgendwo eine Peek-Poke-Routine versteckt wurde. Meine Hoffnung, Peeks und Pokes durch die Vorgabe eines möglichst kurzen Quelltextes zu verhindern, erfüllte sich nicht. Andererseits wollte ich aber auch nicht den regulären Assembler zulassen, denn sonst hätten nur lupenreine Assemblerprogramme eine Gewinnchance gehabt, was nicht im Sinne eines Pascal-Wettbewerbs sein kann. Zudem gibt es auch hier wieder einen rein „arbeitstechnischen" Grund. Wären beispielsweise 250 Lösungen eingegangen, die dann unter Pascal 1.1 und 1.2 hätten nachvollzogen werden müssen, so wäre ich tagelang mit dem Assemblieren, Compilieren und Linken beschäftigt gewesen. Und Sie hätten sich gewundert, warum das September-Heft vom Peeker so spät erscheint... Soviel aus einer ganz anderen Perspektive. Den grundgescheiten Argumenten in den Leserbriefen selbst habe ich nichts hinzuzufügen.
Ulrich Stiehl

Leserbriefe

In jedem Heft Pascal
Ich glaube, der Erfolg des Peeker beruht darauf, daß er allen etwas zu bieten hat, und zwar in jedem Heft. Egal welche Interessen der jeweilige Leser hat, ob er in Applesoft programmiert, in Assembler oder in Pascal, ob er ProDOS benutzt oder CP/M, in jedem Heft findet er einen Artikel, der ihn anspricht.
Für Ihre weitere Arbeit würde das bedeuten, daß jeder dieser Teilbereiche weiterhin in jedem Heft behandelt werden muß. Für mich wäre es ein Grund, das Abonnement zu kündigen, wenn ich nicht in jedem Heft einen Artikel über Pascal finden würde! Ich kann deshalb auch nicht den Folgerungen von Herrn Stiehl aus der mangelhaften Resonanz auf das Pascal-Preisausschreiben zustimmen. Das geringe Interesse beruht meines Erachtens auf der Tatsache, daß es nur wenige Programmierer gibt, die sich für Pascal und ProDOS gleichzeitig interessieren. Wenn es nach mir ginge, dürfte die Berichterstattung über Pascal ausgedehnt werden.
Joachim Schneider, Siegburg

Wettbewerbsregeln nur für Freaks
In dem Artikel über das Pascal-Preisausschreiben beklagen Sie die geringe Teilnahme und leiten daraus den Schluß ab, „daß es unter den Apple-Programmierern nur ganz wenige Pascal-Programmierer gibt und daß es unter den Pascal-Programmierern nur ganz wenige gibt, die wirklich schwierige Programme schreiben können." Diese Schlußfolgerung ist meiner Meinung nach sehr voreilig: Das Preisausschreiben war so angelegt (möglichst kurzer Quell- und P-Code, möglichst schnelle Kopierzeiten), daß nur Pascal-„Freaks", die die P-Maschine in- und auswendig kennen, unter Anwendung aller möglichen „Tricks" eine Chance zum Gewinnen hatten. Die Bedingungen waren damit völlig pascal-untypisch und mußten zu völlig pascal-untypischen Programmen führen.
Die Bedingungen des Preisausschreibens liefen den Stärken von Pascal (z.B. Blockstruktur, selbstdokumentierendes Programm-Listing, Unabhängigkeit von Kenntnissen über Interna des Betriebssystems...) völlig zuwider. Somit motiviert das Preisausschreiben eben nicht „normale" Pascal-Programmierer, sondern nur die o.g. „Freaks". Das abgedruckte Sieger-Programm ist dann auch ein Musterbeispiel für unstrukturierten, unverständlichen (Anti-) Pascal-Code. Man muß schon zweimal hinschauen, um zu erkennen, daß es sich dabei um ein Pascal-Programm handelt. Solch ein Programmierstil wird an Schulen sicher nicht gelehrt.
Da Herr Allgeier ein wirklich kompaktes mit Programmierkniffen gespiktes Programm geschrieben hat, verstehe ich nicht, daß Sie es so kommentarlos abdrucken. Als „normaler" Pascal-Programmierer verstehe ich (und sicher die meisten Peeker-Leser) so manchen Trick nicht. Ich wäre Ihnen also dankbar, wenn Sie die Funktionsweise des Sieger-Programms in einer der nächsten Ausgaben Ihrer Peeker-Zeitschrift ausführlich erklären würden. Insbesondere interessiert mich, was in den BIOS-Puffer ab $0200 gepokt wird und was die in den Apple-Handbüchern nicht dokumentierte Prozedur UNITSTATUS bewirkt. Das wäre auch ein gutes Thema in der interessanten Serie „Tips und Tricks in Pascal" von D. Geiß.
Friedel Berlage, Borgentreich

Vielseitigkeit der Aufgabe
Sie haben bestimmt nicht recht, wenn Sie sagen, es gäbe unter den Apple-Programmierern nur ganz wenige Pascal-Programmierer, von denen nur ein Bruchteil wirklich schwierige Programme schreiben kann.
Die mangelnde Resonanz beruht wahrscheinlich vielmehr auf der Vielseitigkeit der gestellten Aufgabe: Der Programmierer muß gute Pascal-Kenntnisse besitzen, um ein gerafftes Programm zu schreiben, er muß die Maschinensprache beherrschen, um eine akzeptable Routine zum Zusammenfassen der Leerzeichen schreiben zu können, dann muß er so weit im UCSD-System beheimatet sein, daß er weiß, wie man peekt, pokt und einen Maschinensprache-Aufruf simuliert, der unter Pascal 1.1 und 1.2 läuft, und schließlich hat er auch noch den Aufbau des ProDOS-Inhaltsverzeichnisses zu kennen.
Ferner sind bestimmt viele Programmierer durch das Kriterium der Kürze abgeschreckt worden, das überaus „häßliche" Programme zustande kommen ließ. Zudem hätte man externe Maschinenunterprogramme zulassen sollen, was die Arbeit wesentlich erleichtert und die Programme lesbarer gemacht hätte.
Thorsten Brants, Norden

Wenige Pascal-Aufsätze für Fortgeschrittene
Ich wollte schon seit langem ein paar Bemerkungen zu den Peeker-Pascal-Artikeln mitteilen und nehme u.a. Bezug auf das Pascal-Preisausschreiben und Ihre in der Lösung geäußerten kritischen Gedanken für mehr (bzw. weniger) Pascal-Artikel: Auf keinen Fall sollten (meiner Meinung nach) weniger Pascal-Artikel gebracht werden, eher mehr. Das geringe Echo auf Ihr Preisausschreiben liegt wahrscheinlich daran, daß

1. gute (= brauchbare) Pascal-Artikel in anderen Zeitschriften nicht vorhanden sind; man beschränkt sich oft auf Anfänger-Lehrgänge, die erklären, was eine For-Next-Schleife ist; das gleiche gilt für Bücher: Die meisten Pascal-Programmierer (nicht professionell!) kennen z.B. das Buch: „Algorithmen und Datenstrukturen" von N. Wirth nicht!
2. „Seriöse" Pascal-Programmierer keine Zeit haben, ein Preisausschreiben zu lösen.

Andreas Kempf, CH-Basel

Maschinenunabhängige Programmierung
Ihre Rückschlüsse zu Ihrem (sog. „Pascal"-Preisausschreiben-) Artikel kann ich absolut nicht teilen. Jeder, der sich in irgendeiner Form mit Pascal beschäftigt, stößt in irgendeiner Form auf das Primzahlenproblem. So ist es wahrscheinlich zu erklären, daß sich relativ viele Leser an dem ersten Wettbewerb beteiligt haben, da sie schon „Standard-Lösungen" kannten. Ferner läuft Ihr zweiter „Pascal"-Wettbewerb der Pascal-Philosophie entgegen. In Pascal soll maschinenunabhängig programmiert werden. Ferner soll eine gewisse Form des Quelltextes vorhanden sein (besonders „gut" in dem abgedruckten PROTOPAS-Quelltext zu sehen).
Als weiteres sollte man sich die Frage stellen, warum sich ein Pascal-User unbedingt das ProDOS-System anschaffen sollte. Ich habe es nicht und werde es auch nicht tun, da es den einzigen Vorteil brächte, wieder in irgendeinem unstrukturierten, wildwuchernden BASIC-Dialekt mit Peeks und Pokes absolut unlesbare Programme zu schreiben. Sie werden verstehen, daß viele Pascal-Fans, die über kein Superbetriebssystem wie ProDOS verfügen, nicht an dem Wettbewerb teilnehmen konnten. Meiner Meinung nach könnten Sie natürlich wesentlich mehr Pascal-Aufsätze bringen, aber bitte mit Problemen, zu deren Lösungen man kein anderes Betriebssystem benötigt oder irgendwelche System- oder sonstige Adressen kennen muß. Vielleicht sollte man unter den Lesern mal eine Umfrage starten, welche Themen besonders interessieren. Ihre Serie „Tips und Tricks in Pascal" finde ich übrigens ganz hervorragend.
Axel Berthold, Edertal

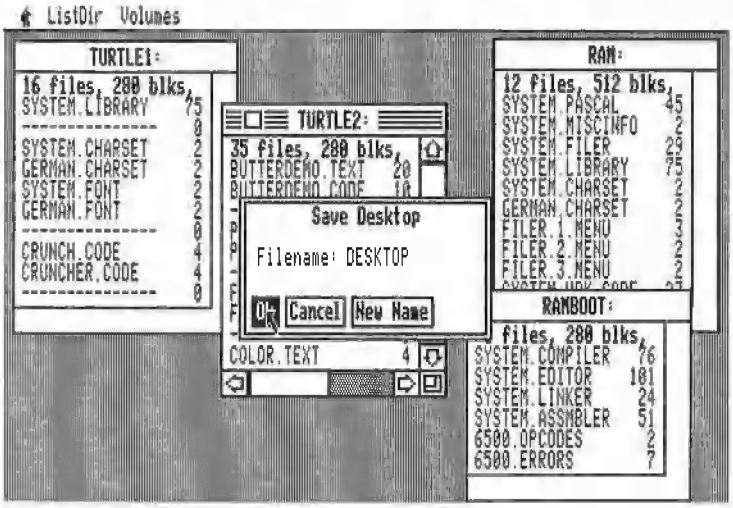
Quelltextlänge absurd
Gegen die Aufgabenstellung an sich wäre nichts einzuwenden gewesen, einmal abgesehen davon, daß ich UCSD längst in die Ecke gestellt habe und nur noch unter Turbo arbeite. Daß ein Bewertungsmaßstab die Geschwindigkeit war, geht auch in Ordnung. Aber die Länge des Objectcodes zu bewerten war überflüssig, da ein Kopierpro-

In Vorbereitung

TurtleGraphics-Library-Paket

von Dieter Geiß

Turtle-Utilities für Fenstertechnik und Apple-Maus in einfacher und doppelter Hires-Grafik für Pascal 1.1/1.2 auf Apple II+/e/c mit Maus oder Joystick.
2 Disketten mit umfangreichem Manual, DM 98,–
Erscheinungstermin Anfang 1986



Das Utility-Paket besteht aus vier Modulen, die von Programmierern benutzt werden können, um professionelle grafische Anwendungsprogramme in Pascal zu schreiben.

Benötigt wird ein Apple Pascal Betriebssystem, entweder die Version 1.1 oder die neue Version 1.2. Bestehende Programme laufen ohne Einschränkung mit der neuen „TurtleGraphics", wenn diese nicht zu viel Speicherplatz verbraucht haben, da die neue „TurtleGraphics" umfangreicher als die alte ist.

Im einzelnen bietet das Paket folgende Möglichkeiten:

- volle Kompatibilität mit der alten „TurtleGraphics"
- lauffähig auf Pascal 1.1 und 1.2
- funktionsfähig mit angeschlossener UltraTerm-Karte
- alle zeitkritischen Funktionen in reinem Assembler programmiert
- Benutzung der zweiten Hires-Seite ($4000–$5FFF) möglich
- für Apple IIc und Apple IIe mit erweiterter 80-Zeichen-Karte Benutzung der doppelten Hires-Grafik mit 560 × 192 Punkten bzw. 16 neuen Farben möglich
- schnelle Prozeduren zum Zeichnen eines Punktes oder einer Linie
- Linearisierung von Teilen des Hires-Schirms
- Benutzung mehrerer Zeichensätze gleichzeitig
- Laden und Speichern von Hires-Bildern mit Ausdruck über Pascal-SUPERDUMP
- Scrolling des Hires-Schirms oder eines Teils in vier Richtungen
- drei verschiedene Schriftarten: Fett-, Breit- und Proportionalschrift, beliebig mischbar (acht Möglichkeiten)
- spezielle schnelle Ausgabe von Text
- Cursor bei Eingabe frei programmierbar
- Ein-/Ausgabe von INTEGER-, CHAR-, STRING- und REAL-Werten im Grafikmodus
- Menüzeile wie beim Macintosh (Die nachfolgenden Module benötigen Maus/Joystick)
- Pull-down-Menüs
- Laden und Speichern von Fenstern (Windows)
- Öffnen von Fenstern
- Aktivieren und Deaktivieren von Fenstern
- Verschieben und Vergrößern/Verkleinern von Fenstern
- Scrolling von Fensterinhalten in allen vier Richtungen
- Umfangreiche Demos als Quelltexte.

Hüthig Software Service · Postfach 10 28 69 · 6900 Heidelberg

gramm ganz allgemein um so schneller ist, je mehr freien Arbeitsspeicher es zur Verfügung hat. Trotzdem hätte ich ja auch das noch toleriert, wenn nicht jemand (Sie!) auf die ungeheuerliche Idee gekommen wäre, die Länge des Quelltextes einer Compilersprache zu bewerten. Ich nehme an, daß 99,9% aller Pascal-Programmierer an dieser Stelle aufhörten, die Ausschreibung weiter durchzulesen. Wie Ihr Sieger-„Programm" zeigt: Das ist nicht Pascal, sondern ein Alptraum, den man nun wirklich nicht mehr als Pascal-Programm bezeichnen kann.
So beweist das magere Ergebnis nicht etwa, daß Ihre Leser sich nicht mit Pascal beschäftigen, eher im Gegenteil. Es beweist nur, daß die Pascal-Gemeinde nicht bereit ist, bei einem derartigen Verbrechen an dieser schönen Sprache zum Mittäter zu werden.
Klaus Mahlmann, Braunschweig

Konvertierungsprogramm ohne Nutzen

Wenn die Resonanz auf Ihr Preisausschreiben extrem gering war, so liegt das nicht am Interesse der Leser an Pascal, auch nicht am hohen Preis für das UCSD-System (Turbo-Pascal läuft ja auch auf dem Apple und ist billiger), sondern an der Art der Aufgabenstellung:
Zur Teilnahme war nicht nur erforderlich, Pascal zu beherrschen (inklusive Systemkenntnisse wie UNITREAD etc.), sondern man mußte darüber hinaus detaillierte Kenntnisse über den internen Aufbau von ProDOS mitbringen. Darüber hinaus hat für viele Leser die geforderte Utility keinen praktischen Nutzen, denn Pascal wird vorwiegend zum Entwickeln eigener Programme verwendet, während ProDOS doch eigentlich nur für die Anwendung kommerzieller Software interessant ist. Wenn ich das UCSD-System benutze, wozu brauche ich dann ProDOS? Daher war die geringe Resonanz auf das Preisausschreiben vorherzusehen – wählen Sie in Zukunft wieder weniger spezielle Aufgabenstellungen!
Das soll keine Kritik an Ihrer ausgezeichneten und vielseitigen Zeitschrift sein. Im Rahmen Ihrer Pascal-Artikel würden mich insbesondere solche über die interne Struktur des UCSD-P-Systems interessieren. Hier gibt es eine Reihe von Eigenschaften, die keinem mir bekannten Buch zu entnehmen sind (Wirkung der U-Compiler-Option, Funktion von SYSTEM.ATTACH etc.).
Dr. Hans-Joachim Scheefer, Frankfurt

UCSD-Pascal zu teuer

Ich möchte das einjährige Jubiläum Ihrer Zeitschrift zum Anlaß nehmen, zu einigen Themen der letzten Zeit Stellung zu nehmen.
Die geringe Resonanz auf das Pascal-Preisausschreiben erscheint mir nicht weiter verwunderlich. Der extrem hohe Preis des Apple-UCSD-Systems wird meines Erachtens viele Leser von einem Kauf abgehalten haben. Wer das UCSD-P-System kennt, wird überdies wenig Lust an Pascal bekommen. Ohne die Vorteile dieses Systems leugnen zu wollen, scheinen mir die Nachteile – Geschwindigkeit, Diskettenorientierung, Pseudocode – für den Hobbyisten zu überwiegen. Die Portabilität der Programme allein ist zumindest für mich kein Kaufanreiz gewesen. Wer das Turbo-Pascal-System kennt, wird sich wohl kaum weiter für Apple-Pascal interessieren. Der Wechsel vom Editor in den Compiler, die Compilierung selbst, all das ist in einer phantastischen Geschwindigkeit möglich. Hinzu kommt die Möglichkeit, CP/M-Command-Files zu erzeugen, die unabhängig vom Turbo-Pascal arbeiten können, ein unbestreitbarer Vorteil. Selbst mit einer Z80-Karte ist das Turbo-Pascal-System noch immer günstiger als das Apple-UCSD-System.
Georg Basse, Oldenburg

Pascal in der Schule

In der September-Ausgabe des Peeker veröffentlichten Sie die Gewinner des Pascal-Preisausschreibens und beklagten gleichzeitig die geringe Beteiligung. Damit verbinden Sie die Frage an die Leser, ob in Zukunft weniger Pascal-Aufsätze im Peeker abgedruckt werden sollen. Aus meiner Sicht kann ich nur sagen: Lieber wäre mir mehr Pascal. Ich bin Lehrer und unterrichte auch Informatik. In den Lehrplänen des Landes Rheinland-Pfalz ist der Informatikunterricht ganz auf Pascal abgestellt. Allerdings muß man auch dazu wissen, daß Informatik nicht gleichbedeutend mit einem Programmierkurs sein kann und darf.
Daraus erklärt sich auch, daß zwar viele Schüler Grundkenntnisse in Pascal haben, mehr aber auch nicht. Im Normalfall hat der Schüler, der Informatik gewählt hat, keinen pascal-fähigen Computer zu Hause, auf dem er sich die notwendige Übung für das UCSD-Betriebssystem holen kann. Bei zwei oder drei Stunden Unterricht pro Woche bleibt dafür in der Schule auch wenig Zeit. Der „normale" Informatikschüler (ich spreche hier nicht von dem „Freak") ist also kaum in der Lage, Aufgaben von der Schwierigkeit Ihres Preisausschreibens zu lösen.
Wenn Sie diesen Schülern – und auch den meisten Lehrern – mit Ihrer Zeitschrift helfen wollen, dann bringen sie weiter Pascal-Aufsätze, und Sie können ruhig vom Niveau etwas weniger hoch sein. Von Zeit zu Zeit könnten auch manche Begriffe noch einmal erklärt werden. Das gilt übrigens nicht nur für Pascal-Aufsätze. Nicht jeder Leser Ihrer Zeitschrift weiß, was z.B. „Patch" oder „Double-Hires" ist. Und Ihre Redakteure haben ja bereits bewiesen, daß sie Ihr Handwerk verstehen.
Ansonsten kann ich mich eigentlich nur dem Lob anschließen, das z.B. aus den Leserbriefen der September-Ausgabe herauszuhören ist. Ich weiß natürlich auch, daß es außer dem von mir genannten Adressatenkreis auch die vielen Bastler, Tüftler usw. gibt, für die Sie anders schreiben müssen. Aber vielleicht kann man es hier doch mal allen recht machen.
Jürgen Kettner, Vallendar

Pascal-Interna wenig bekannt

Der mangelnde Zuspruch auf Ihr Pascal-Preisausschreiben hat meiner Meinung nach folgene Ursachen:

1. Da ProDOS seine Vorteile nur bei sehr großen Datenmengen voll ausspielen kann, die bei den meisten Hobby-Programmierern einfach nicht anfallen, dürfte das Interesse an ProDOS schon verhältnismäßig gering sein.
2. Die Pascal-Diskettenorganisation ist nahezu vollständig unbekannt und schreckt daher viele davon ab, Low-Level-Prozeduren zu benutzen.
3. Es ist beinahe unmöglich, mit dem Pascal-System in Assembler zu programmieren, da

a) keinerlei Systemroutinen bekannt sind;
b) es keinerlei Beispielprogramme gibt, die auch nur den Einsatz der diversen Befehle einigermaßen erklärt;
c) man nie genau weiß, wohin man denn die Routinen schreiben soll, ohne daß sie Gefahr laufen, zerstört zu werden;
d) es keine Beispiele gibt, in denen erklärt wird, wie die Parameter wieder korrekt zurückgegeben werden müssen (bes. Strings, Arrays, Reals);
e) mir keine Routine bekannt ist, die mir z.B. den Typ eines „typlosen" Parameters angibt, somit ist diese Möglichkeit der Parameterübergabe für mich vollkommen wertlos.

Aus diesen o.g. Gründen würde ich es begrüßen, wenn Sie Ihre Pascal-Serie weiter fortsetzen würden und wenn irgend möglich auch dazu beitragen, dieses Informationsdefizit zu füllen.
Konkret würde mich derzeit folgendes interessieren: Simulieren einer Pascal-STR-Funktion unter Benutzung der entsprechenden BASIC-ROM-Routine.
Gerd Grammel, Wiernsheim

Quelltextkürze sinnlos

Der neueste Peeker ist wieder voll interessanter Beiträge, die sich qualitativ wohltuend von denen anderer Fachzeitschriften unterscheiden. Ihre Anmerkungen zum Pascal-Preisausschreiben haben mich allerdings ziemlich getroffen, daher kann ich mir einen Kommentar nicht verkneifen. Ich programmiere seit über zwei Jahren UCSD-Pascal und bin wohl fortgeschritten genug, auch komplizierte Probleme lösen zu können. Zwei Dinge haben mich jedoch an einer Teilnahme gehindert:
1. Ich kenne den Aufbau des ProDOS-Inhaltsverzeichnisses noch nicht.
2. Die Forderung eines möglichst kurzen Quellcodes erscheint mir absolut sinnlos. Mir ist ein mehrseitiges, gut kommentiertes Pascal-Programm tausendmal lieber als dieser unästhetische, undurchsichtige Quellcode.
Norbert Dörner, Langenhahn

Pascal-Assembler statt Peeks und Pokes

Sicherlich ist Pascal bei den Apple-Besitzern nicht so verbreitet wie BASIC, aber es ist sicherlich nicht so, daß die Pascal-Programmierer eine sehr kleine Gruppe sind. Die von Ihnen angeführten Gründe für das geringe Echo auf das Pascal-Preisausschreiben sind meines Erachtens nicht ausschlaggebend, wenn sie auch zutreffend sein mögen. Ich möchte nun Gründe nennen, die aus meiner Sicht zutreffender sind:
1. Für das Preisausschreiben wurde ein sehr ungünstiger Zeitpunkt gewählt, denn in ganz Deutschland war vom Erscheinen des Heftes Nr. 7/85 bis zum Einsendeschluß am 22.07.1985 Ferienzeit, somit waren wenige der potentiellen Teilnehmer zu Hause und hatten Zeit, sich zu beteiligen.
2. Die Voraussetzung 1 erscheint mir nicht konsequent genug zu sein. Es hätte, wenn schon gepokte Maschinenroutinen erlaubt sind, auch die Benutzung des Assemblers und somit auch die des Linkers zum Einbinden von Maschinenroutinen gestattet sein sollen, denn was spricht dagegen, das Pascal-System voll auszunutzen? Die Erweiterung dieser Voraussetzung hätte natürlich zur Folge gehabt, daß die Lösungen hauptsächlich in Assembler geschrieben worden wären, womit das Preisausschreiben kein Pascal-Preisausschreiben mehr gewesen wäre. Deswegen hätte die Bedingung restriktiver sein müssen, sie hätte auch gepokte Maschinenroutinen verbieten müssen.
3. Die Vorschrift zur Berechnung der Punkte hätte nicht die Länge der Source-Codes beinhalten sollen, denn dies führt zu Programmen, die in ihrem Erscheinungsbild dem Konzept von Pascal widersprechen. Dies hat sich ja in dem im Peeker Nr. 9/85 veröffentlichten Programm gezeigt.
4. Ein weiterer gravierender Nachteil des Preisausschreibens war die Aufgabe selbst, da sie voraussetzte, daß weitergehende Kenntnisse von zwei Betriebssystemen vorhanden sein müssen.

Jetzt möchte ich noch auf meine persönliche Erfahrungen mit dem Pascal-Preisausschreiben eingehen. Bei mir stellte sich beim Kauf des Peeker Nr. 7/85 eine starke Euphorie ein, als ich vom Pascal-Preisausschreiben las, da ich ein begeisterter Pascal-Programmierer bin. Bei näherer Betrachtung stellte sich bei mir allerdings eine starke Enttäuschung ein, die sich durch die oben genannten Gründe belegen läßt, zumal ich auch nicht im Besitz von ProDOS bin. Im großen und ganzen scheint mir, daß das Pascal-Preisausschreiben nicht richtig durchdacht war, was ich von Ihrer Zeitschrift nicht erwartet hätte. Dies ändert allerdings nichts an dem positiven Eindruck am Peeker, den ich während der Zeit gewonnen habe, in der ich zu den Lesern des Peeker zähle.
Ich hoffe, daß die Reaktion auf das Pascal-Preisausschreiben beim Peeker nicht zur Vernachlässigung von Pascal führt. Ich persönlich wünsche mir mehr über Pascal, speziell über das Betriebssystem.
Arno Zietz, Hamburg

Pascal 1.2 veraltet

Mehr Beiträge über Pascal, das ist Ihre Frage.
Mein Antwort: ja, aber...
Computersprachen und Betriebssysteme sind eine Glaubensfrage, welche oft zur Weltanschauung erhoben wird. Wo liegen nun die Möglichkeiten von Pascal?
1. Strukturierte Programmierung,
2. Strukturierte Daten,
3. Modulares Konzept.

Das erste wird oft gerühmt und nachgeahmt (Fortran 77, Ratfor, neuere BASIC-Dialekte), aber Punkt 2 und 3 werden oft völlig unterschlagen und werden wohl auch nie so richtig kohärent in andere Sprachen eingebaut werden

können. Man kann auch zusammenfassend sagen, daß Pascal ein Konzept zu Grunde liegt, wogegen besonders die o.g. Sprachen aus der Steinzeit der Computer stammen und eher eine Anhäufung von Regeln darstellen. Wo liegen nun die Vorteile von UCSD-Pascal?
4. Ohne Z80-Karte auf IIe und IIc lauffähig,
5. Flexibel und ausbaufähig etc., aber nicht mehr State of the Art. Es gibt für andere Rechner immerhin schon die Version 5.2 und nicht erst eine popelige 1.2. Es ist viel zu teuer. All das und vieles mehr hängt wohl mit der merkwürdigen Firmenpolitik von Apple zusammen, wenn Apple überhaupt eine hat.
Dipl.-Ing. Peter Klamser, Goslar

Pascal, ProDOS und Schule
Zur Auflösung der Programmieraufgabe ProDOS nach Pascal und der Artikelserie über Pascal von Dieter Geiß habe ich einige Anmerkungen zu machen.
Ich glaube, daß die Aussage aus Ihrem Artikel, daß Pascal bei den Lesern des Peeker keinen so großen Anklang oder keine so große Verbreitung hat, nicht unwidersprochen hingenommen werden kann. Meiner Ansicht nach hat sicher ein großer Anteil der Leserschaft Interesse an der Sprache und der Lösung von Problemstellungen. Allerdings war die Programmieraufgabe von einer etwas seltsamen Gestalt, so daß mich die geringe Resonanz nicht im mindesten verwundert hat. In meinem weiteren Bekanntenkreis, der sich durchaus intensiv mit der Programmierung in Pascal befaßt, hat sich niemand gefunden, der auch nur im geringsten an einer Lösung für ein Transferprogramm von ProDOS nach Pascal Interesse bekundet, weil niemand mit ProDOS und Pascal simultan arbeitet. Das liegt sicherlich nicht unbedingt an ProDOS, sondern daran, daß Applesoft-BASIC eine Programmiersprache darstellt, die nicht als Ideal zur Lösung größerer Programmieraufgaben betrachtet werden kann und somit die Verwendung des Pascal-Editors zur Editierung von BASIC-Programmen ausscheidet, da größere Programme sowieso besser in Pascal geschrieben werden. Allerdings finden die o.g. Anwender das Arbeiten mit ProDOS auch nicht so einfach wie etwa das Arbeiten unter DOS 3.3 und Ablegern, da Disketten- und Dateienansprache nicht vereinfacht, sondern eher verkompliziert wurden. Des weiteren fehlt bestimmt vielen Pascal-Programmierern der tiefere Einblick in das ProDOS-Betriebssystem, da sie sich aus den o.g. Gründen nicht damit zu beschäftigen brauchen und wollen. Die einzige vorstellbare Anwendung eines Transferprogramms von ProDOS nach Pascal wäre die Übertragung von Appleworks-Dateien, wobei hier allerdings die eventuelle Verwendung von Steuercodes im Text die Übertragbarkeit stark einschränken würde und somit die Erstellung von Texten im Pascal-Editor sinnvoller erscheint.
Zu dem von Ihnen angesprochenen Thema Pascal an Schulen wäre einiges zu sagen.
Daß Pascal an den Schulen nicht bis zur Perfektion gelehrt wird, ist durchaus korrekt. Man kann dieses Manko allerdings nicht den Schulen anlasten, da der Lehrauftrag der Schulen nicht darin besteht, perfekte Pascal-Programmierer auszubilden, sondern Pascal nur als Beispiel für eine höhere Programmiersprache in seinen Grundzügen behandelt und gelehrt werden kann. Hier denke ich auch, daß Pascal mehr oder weniger zufällig als Sprache nur ausgewählt wurde, da Pascal auf beinahe allen Mikrocomputern implementiert ist und, noch wichtiger, durch seine Strukturierung zu einem überlegten Programmieren und einem relativ sauberen Programmierstil erzieht. Hieraus resultiert eine vergleichsweise (z.B. zu BASIC) einfache Nachvollziehbarkeit der Programmierschritte und somit die ideale Eignung als Sprache für den Unterricht. Wenn eine andere Sprache diese Voraussetzungen erfüllt hätte, so wäre sicherlich jene Sprache zur Schulprogrammiersprache Nummer eins geworden. In dieser Eigenschaft stellt Pascal nur einen kleinen Teil des Lehrplanes neben der theoretischen Informatik und einer Einführung in die Assemblerprogrammierung dar.

Als letzten Punkt möchte ich noch die sehr interessanten Artikel von Herrn Geiß ansprechen. Sie sind von hohem Nutzen für den Anwender, der sich tiefer mit Pascal befaßt. Nur vermisse ich immer Hinweise auf die Quellen des Wissens, so daß manche Artikel etwas undurchsichtig bleiben. Es kann doch nicht angehen, daß das Wissen nur durch die allgemein bekannte Pascal-Literatur kommt, da hier eine sehr detaillierte Kenntnis des Systems vorliegt, die ich noch nirgends beschrieben gesehen habe. Soll man doch der armen, unwissenden Leserschaft die Chance geben, über die paar Krümel hinaus, die durch die Artikel verteilt werden, etwas mehr an Wissen zu erlangen und nicht eine Art Abhängigkeit zu schaffen, die ich persönlich nicht als sehr angenehm empfinde.
Adrian Meissner, Trier

Jahresinhaltsverzeichnis

Peeker 1/84 bis 12/85

Das nachfolgende Jahresinhaltsverzeichnis ist nach Heften und innerhalb der Hefte nach Rubriken gegliedert. Ein gesonderter Textfile mit einem Suchprogramm für Apple II+, IIe und IIc ist auf der Peeker-Sammeldisk # 12 enthalten. Das Sachprogramm wird mit RUN WORT.SUCHER gestartet.

Rubriken

Applesoft
Assembler
Buch
CP/M
DOS 3.3
Drucker
Grafik
Händler-Profil
Hardware
Hobby
Leserbriefe
Kurzbericht
Macintosh
MBASIC
Nachtrag
Pascal
ProDOS
Produkt
Recht
Schule
Telekommunikation
Test
Turbo
Vermischtes

Heft 1/84

Hardware: Der neue 65C02-Prozessor. Ulrich Stiehl. 1/84/6
Assembler: 65C02-Disassembler für Apple IIe und II Plus. Ulrich Stiehl. 1/84/14
Hardware: Accelerator IIe. Die neue Superkarte. Ulrich Stiehl. 1/84/19
Applesoft: Applesoft simuliert Turtle Graphic. Harald Grumser. (Ergänzung 2/84/10) 1/84/26
Grafik: Wie man die Grafik verdoppelt. Teil 1: Double Lores. Ulrich Stiehl. 1/84/32
Drucker: Hires-Grafik-Dump für Epsondrucker. Jürgen und Dieter Geiß. 1/84/40
Turbo: Turbo-Pascal Schritt für Schritt. Dr. Ekkehard Kaier. 1/84/51
ProDOS: ProDOS-Patch für geänderte F8-ROMs. Ulrich Stiehl. (Ergänzung 2/84/10) 1/84/54
ProDOS: Patch für ProDOS.INIT 80 Spuren. Hans-Georg Hüneke. 1/84/56
Hobby: Wer ist der Schnellste? Primzahlen-Preisausschreiben. 1/84/58
Vermischtes: Hinweise für Autoren. 1/84/62
Produkt: Individuelle Zeichensätze für Matrixdrucker (DMP-Charger). 1/84/63
Produkt: AFC-Tastatur mit Maus für Apple. 1/84/63
Produkt: Copy Killer (Kopierschutzsystem). 1/84/64
Produkt: Sony-Laufwerke von Ueding. 1/84/64
Produkt: 65C02 + Accelerator: Wer liefert was? 1/84/64
Produkt: Bidirektionaler Datenverkehr zwischen Apple und Brother-Typenradschreibmaschine. 1/84/64
Buch: Apple DOS 3.3. 1/84/65
Buch: Mathematische + Wissenschaftliche Programme in Basic. 1/84/65
Buch: Statistik in Basic. 1/84/65
Buch: Apple II leicht gemacht. 1/84/65

Heft 2/84

Leserbriefe: 2/84/8
ProDOS: ProDOS für Anfänger. Teil 1: Was ist ProDOS? Ulrich Stiehl. 2/84/12
ProDOS: Was Geschäftsleute vom ProDOS-FILER wissen sollten. Ulrich Stiehl. 2/84/20
Grafik: Wie man die Grafik verdoppelt. Teil 2: Double Hires. Karl-Walter Bott. 2/84/24. Vgl. 1/84/32
Telekommunikation: Modems kommen in Mode. Praktische Ratschläge für die Telekommunikation mit Modem. Matthias Pohl. 2/84/36
Applesoft: Ampersand macht's möglich. INSTRING-Befehl in Applesoft. Dr. Jürgen B. Kehrel. 2/84/40
Applesoft: IF-THEN-ELSE-Befehl simuliert in Applesoft. Franz-Josef Hüskens. 2/84/48
Applesoft: Wie man Arrays löscht. Christiane Breit und Thomas Schneyer. 2/84/51
Macintosh: Microsoft Basic leicht gemACht. Teil 1: Operanden und Operatoren. Pit Capitain. 2/84/54
Test: Gestochen scharf. Die Videokarte UltraTerm für den Apple II. Dieter und Jürgen Geiß. 2/84/60
Hardware: Prometric. Ein Apple-Kompatibler mit 65C02C. Ulrich Stiehl. 2/84/64
Test: S.A.M. The Software Automatic Mouth oder wie man den Apple zum Sprechen bringt. Ralf Augspurger. 2/84/66
Händler-Profil: r + r electronik. 2/84/69
Hobby: Die Gewinner des Primzahlen-Wettbewerbs. 2/84/70
Vermischtes: Hinweise für Autoren. 2/84/72
Test: Diskettenlaufwerke für ProDOS im Test. Ulrich Stiehl. 2/84/74
Produkt: Atari-Joystick und Apple. 2/84/75
Produkt: MacLabel. 2/84/75
Produkt: CTS-Mineralöl. 2/84/75
Buch: BASIC Übungen für den Apple. 2/84/76
Buch: Bücher für Elektronik + Computer. 2/84/76
Buch: Schulsoftware-Katalog. 2/84/76
Buch: The Apple in your hand. 2/84/76
Buch: Apple II Tips & Tricks. 2/84/76
Buch: Apple ProDOS für Aufsteiger (Band 1). 2/84/78
Buch: Apple Maschinensprache. 2/84/78
Buch: Apple Assembler. 2/84/78
Buch: Pandasoft-Katalog. 2/84/78

Heft 1-2/85

DOS 3.3: Poor Man's RAM-Disk. RAM-Disk-Driver für die Language Card. Ulrich Stiehl. (Ergänzung 6/85/69) 1-2/85/8
Technik: Die RAM-Karten von IBS. Mit Quellcode eines RAM-Disk-Drivers. Ulrich Stiehl. 1-2/85/18
ProDOS: QUICKCOPY. Ein schnelles ProDOS-Kopierprogramm für 35-80 Spuren. Ulrich Stiehl. 1-2/85/22
ProDOS: ProDOS und „Kompatible". Ein allgemeines Patch-Programm. Arne Schäpers. (Berichtigung in Leserbrief 10/85/69) 1-2/85/31
Applesoft: Müllabfuhr wie ein Blitz. Eine ultraschnelle Garbage-Collection-Routine. Harald Grumser. (Berichtigung 8/85/70) 1-2/85/32
Assembler: Speichern Sie den Bildschirm ab! Organisation des 40-Z/Z-Apple- und des 80-Z/Z-Videx-Bildschirms. Dr. Jürgen B. Kehrel. 1-2/85/42
MBASIC: Lohn- und Einkommensteuer 1984. Ein umfassendes CP/M-MBASIC-Programm. Volker Duske. 1-2/85/45
Pascal: Pascal-Diretory unter der Lupe. Dieter Geiß. 1-2/85/64
Pascal: GETPAS: Konvertierung von Pascal- in DOS-Textfiles. Ulrich Stiehl. 1-2/85/68
Pascal: GETDOS: Konvertierung von DOS- in Pascal-Textfiles. Jürgen Geiß. 1-2/85/70
Macintosh: Ikonen und Deixis oder die Philosophie des Macintosh. Ulrich Stiehl. 1-2/85/74
Macintosh: Microsoft Basic leicht gemACht. Teil 2: Funktionen und Programmsteuerbefehle. Pit Capitain. 1-2/85/78
Telekommunikation: Weltweite Datenübertragung mit dem WS 2000. Ein Universalmodem für alle Normen. Matthias Pohl. 1-2/85/84
ProDOS: Applesoft-Editor-Macros. Ein Quickie für den PRODOS.EDITOR. Ulrich Stiehl. 1-2/85/86
Test: Exbasic Level II im Test. Eine Basic-Erweiterung bringt Sie ins Staunen, nicht nur positiv. Dr. Jürgen B. Kehrel. 1-2/85/88
Vermischtes: Hinweise für Autoren. 1-2/85/92
Buch: Das Buch zum Apple II. 1-2/85/93
Buch: Programmierung des 6502. 1-2/85/93
Buch: 6502 Anwendungen. 1-2/85/93
Buch: Apple II ROM Listing. 1-2/85/93
Buch: Fortgeschrittene 6502 Programmierung. 1-2/85/94
Buch: Macintosh Benutzerhandbuch. 1-2/85/94
Produkt: Doppel-Parallelinterface mit RAM oder EPROM. 1-2/85/96
Produkt: Modula-2 für den Macintosh. 1-2/85/96

KURZBERICHTE

Produkt: 64K-Karte für IIe. 1-2/85/96
Produkt: Staub und Schmutz auf Computer und Tastatur? 1-2/85/97
Produkt: TempoHexe „SpeeDemon" 65C02C-Prozessorkarte. 1-2/85/97
Produkt: Lohn- und Gehaltsabrechnung. 1-2/85/97
Produkt: Lotus – Jazz, ein multifunktionales Software-Paket. 1-2/85/97
Produkt: Branchenprogramm Reisebüro. 1-2/85/97
Produkt: UCSD p-System für den Macintosh. 1-2/85/97
Produkt: STATEX für Bildschirme. 1-2/85/97
Produkt: Computerunterstütztes Lernen. 1-2/85/99
Produkt: Mailmerger für AppleWorks. 1-2/85/99
Vermischtes: Apple User Club Austria. 1-2/85/99
Vermischtes: Apple-Diebe am Werk. 1-2/85/99

Heft 3/85

Technik: Der Apple II als persönliches Ingenieurwerkzeug. Dipl.-Ing. Bernd Worms. 3/85/6
Hardware: Aufbau und Funktion von Diskettensystemen. Dipl.-Ing. Gerhard Berg. 3/85/10
Schule: Testgenerator für Legastheniker. Dr.Wolfgang Kersken. 3/85/22
Schule: Höhere Präzision bei den vier Grundrechenarten. Konrad Steinmeier. 3/85/30
CP/M: Wordstar-Transfer-Refiner. Konvertierung von CP/M-Wordstar- in DOS-Applewriter-Textfiles. Dr. Jürgen B. Kehrel. 3/85/35
Vermischtes: Apple-Hotline. Ulrich Stiehl. 3/85/35
CP/M: GETCPM: Konvertierung von CP/M- in DOS-Textfiles. Applesoft-Version. Thomas Fink. 3/85/43
Macintosh: Die Maske fällt und Mecki seh' ich nun, entblößt von allen Reizen. Der Macintosh im Leistungstest. Ulrich Stiehl. 3/85/44
Hobby: Geschwindigkeit ist keine Hexerei. Erastosthenes erneut betrachtet. Michael G. Schneider. 3/85/56
Test: Pascal 1.2 Evolution statt Revolution. Claus Rautenstrauch. (Ergänzung 10/85/60) 3/85/65
Hardware: Accelerator-Karte abstellen. Ulrich Stiehl. 3/85/66
Test: Double-Hires-Grafik-Programme. Doublestuff Plus. Thomas Bühner. 3/85/70
Test: Double-Hires-Grafik-Programme. Superplot. Anmerkung von Ulrich Stiehl. 3/85/70
Test: Hardbreaker und Softbreaker oder wie man Programme „entschützt". Ulrich Stiehl. 3/85/72
Produkt: Beliebte Apple-Spiele. 3/85/73
Test: Memdos Junior. Ein neues Betriebssystem für den Apple II. Dr. Jürgen B. Kehrel. 3/85/75
Test: CP/M-Karte für den Apple IIc. Dr. Jürgen B. Kehrel. 3/85/75
Test: HOCO-Masterclock. Arne Schäpers. 3/85/76
Leserbriefe: 3/85/77
Vermischtes: Große Peeker-Umfrage. 3/85/78

Heft 4/85

Recht: Knock, Knock! Who's there? Software und Kopierrecht. Ulrich Stiehl. 4/85/6
Grafik: Graf-quattro. Künstlerische Grafik für jedermann. Teil 1: Wie ein Supereditor entsteht. Nino G. Barbieri. 4/85/14
Applesoft: Menü-Generator für den Apple II. Dr. H. Kersten. 4/85/20
Assembler: Makros für die neuen Befehle des 65C02. Dipl.-Oec. Edgar Meyzis. 4/85/30
Drucker: SCREEN80. 80-Z/Z-Bildschirm-Dump. Ulrich Stiehl. 4/85/33
Telekommunikation: Terminal-Programm für den Apple II. Andreas Lecreux. (Ergänzung 6/85/72) 4/85/34
ProDOS: ProDOS für Anfänger. Teil 2: Das definitorische Grundgerüst. Ulrich Stiehl. 4/85/41
ProDOS: Trace-Bug. 4/85/46
ProDOS: PRODOS.READER. 4/85/46
Pascal: Apples Maus lernt jetzt auch Pascal. Jürgen Geiß. 4/85/48
CP/M: CP/M für Einsteiger. Jörg Lange. 4/85/59
Macintosh: Microsoft Basic leicht gemACht. Teil 3: Die Ein- und Ausgabe. Pit Capitain. 4/85/63
Händler-Profil: Orgasoft plant für die Zukunft. 4/85/67
Kurzbericht: Auf dem Abstellgleis: Steve Wozniak. 4/85/68
Kurzbericht: AUGE und Apple-München. 4/85/68
Kurzbericht: Ergänzungen zu „ProDOS für Aufsteiger", Bd. 1. Arne Schäpers. 4/85/69
Kurzbericht: ProDOS-Bücher und -Aufsätze 4/85/69
Kurzbericht: Apple-Bücher im Preisvergleich. 4/85/70
Applesoft: INALL. INPUT für DOS/ProDOS. (Berichtigung 5/85/50) 4/85/70.
Leserbriefe: 4/85/72
Test: Mailmerger und Appleworks. Ulrich Stiehl. 4/85/73
Test: Erphi-Controller. Eine kompatible Erweiterung. Harald Grumser. 4/85/75
Produkt: INTERTEXT. Internationale Textautomation mit dem Apple IIe. 4/85/76
Applesoft: SCREEN80.SAVER. Ulrich Stiehl 4/85/76
Produkt: Akustikkoppler AK 300. 4/85/77
Produkt: Akustikkoppler dataphon s-21-d. 4/85/77
Produkt: Parallelportkarte für Apple II. 4/85/77

Heft 5/85

DOS 3.3: Der File-Manager des DOS 3.3. Harald Grumser. 5/85/6
Hardware: Balfer-Interface. RAM-Disk-Driver für Apple IIe. Ulrich Stiehl. 5/85/12
Grafik: PLOT 2.0. Ein komfortabler Funktionsplotter. Hans-Martin Eng. (Berichtigung 9/85/69) 5/85/17.
Grafik: CONVERT560. Konvertierungsroutine für Double-Hires-Drucker-Dump. Marc van Woerkom. 5/85/25
Assembler: Disassembler für „geheime" 6502-Opcodes. Christoph Bregler. 5/85/31
Pascal: Transzendente Funktionen in Pascal. Dieter Geiß. 5/85/35
CP/M: Das CP/M-Directory näher beleuchtet. Der Aufbau von 16-Sektor-CP/M-Disketten. Dr. Jürgen B. Kehrel. 5/85/39
Macintosh: Apple-II-Monitor für den Mac. Wie zu Wozniaks Zeiten. Pit Capitain. 5/85/41
Applesoft: INPUT für alle Fälle. Tastatur – DOS – ProDOS. Harald Grumser. 5/85/50
ProDOS: Block-Tracer für ProDOS. Ulrich Stiehl. 5/85/51
ProDOS: FORMAT-Befehl für ProDOS. Ulrich Stiehl. 5/85/56
Kurzbericht: Neologismen in der Computersprache. Ulrich Stiehl. 5/85/60
Kurzbericht: Kopierschutz für den Apple II. Matthias Greve. 5/85/61
Kurzbericht: Apple-Kompatible. Hans-Kurt Kuhn. 5/85/66
Vermischtes: Hotline. 5/85/68
Leserbriefe: 5/85/70
Applesoft: RANDOM.DEMO. Random-Zahlen-Generator. Ulrich Stiehl. 5/85/69
Applesoft: COLUMN80.DEMO. VTAB-HTAB-Demo. Ulrich Stiehl. 5/85/69
Test: Slim-Line-Laufwerk von Chinon. 5/85/71
Test: Druckerumschaltung per Knopfdruck. 5/85/71
Test: The Last Disk-Utility. Ein umfangreicher Disketteneditor. Harald Grumser. 5/85/72
Assembler: Tabelle der Interpreter- und Monitor-Adressen. 5/85/72

Heft 6/85

Grafik: Graf-quattro. Teil 2: Schneller als der Schall. Nino G. Barbieri. (Ergänzung 10/85/69) 6/85/6
Grafik: Das Farbbit. Pseudo-Double-Hires auf dem Apple II Plus. H.-D. Siewert. 6/85/16
Grafik: Superdump. Das universelle Hires-Grafik-Dump-Programm. Jürgen Geiß. (Ergänzung 8/85/50) 6/85/22
Drucker: Hardcopy auf Olympia ESW 100 RO. Ein Typenraddrucker wird grafikfähig. Ralf Menssen. 6/85/34
Technik: Fourier-Analyse. Peter Walber. (Berichtigung 9/85/69) 6/85/38
Applesoft: Applesoft-Erweiterungen. Eine Sammlung von Ampersand-Routinen. Markus Enzinger. 6/85/43
Pascal: RAM-Disk-Driver für Pascal 1.1. Michael Schröter. (Berichtigung 7/85/52) 6/85/48
ProDOS: ProDOS. Theorie und Praxis. Ulrich Stiehl. 6/85/52
CP/M: Apple-CP/M. Mit einem RAM-Disk-Driver für die 64K-Karte. Karl-Walter Bott. 6/85/55
Hobby: SOLITAIRE und wie man es löst. Stefan Maas. 6/85/64
Leserbriefe: 6/85/69
Kurzbericht: Wie gibt man Maschinen-

programme ein? 6/85/70
Kurzbericht: An Ear to the Ground. Ulrich Stiehl
Test: Extended Graphic System. Dr. Jürgen B. Kehrel. 6/85/71
Nachtrag: Terminal-Programm für die Super-Serial-Card. Andreas Lecreux. 6/85/72
Test: Diversi-DOS 4-C. Rudolf Rötering. 6/85/72
Vermischtes: Diversi-DOS 2-C auf Peeker-Sammeldiskette #6. (Ergänzung 10/85/69) 6/85/74
Test: Lasar II ze. Harald Grumser. 6/85/75
Test: Typenradschreibmaschine Brother CE 50. Rainer Gerdes. 6/85/76
Test: Editierhilfen für den Apple. Franz-Josef Hüskens. 6/85/77
Test: Lernprogramme von INTUS. Dr. Jürgen B. Kehrel. 6/85/78

Kurzbericht: Apple-Kompatibler MK II. 9/85/65
Kurzbericht: 20M-Festplatte für Mac. 9/85/65
Kurzbericht: Appletalk für Macintosh. 9/85/65
Kurzbericht: Apple-Schul-Programm AULA. 9/85/65
Kurzbericht: Laserwriter von Apple. 9/85/65
Kurzbericht: Design-Preise für Apple IIc. 9/85/65
Kurzbericht: Spooler für Epson-Drukker. 9/85/65
Kurzbericht: Transfer II Terminalprogramm. 9/85/65
Kurzbericht: Druckertisch. 9/85/66
Kurzbericht: Notstromaggregat. 9/85/66
Leserbriefe: 9/85/66
Vermischtes: Paul Lutus im ZDF. 9/85/68
Vermischtes: Visicalc eingestellt. 9/85/68
Vermischtes: Pyramid Pitty als Hex-Dump? 9/85/68
Nachtrag: Fourier-Analyse. 9/85/69
Nachtrag: Formatter. 9/85/69
Nachtrag: PLOT 2.0. 9/85/69
Kurzbericht: Pressestimmen zur Wirtschaftslage von Apple. 9/85/69

Heft 7/85

Hobby: Pyramid Pitty. Ein Reaktionsspiel. Michael Matzat. 7/85/6
Technik: Die AP33-Megawarp-RAM-Karte. Mit einem RAM-Disk-Driver für ProDOS. Ulrich Stiehl. 7/85/8
Technik: FORMATTER. Ein universelles Formatierungsprogramm. Arne Schäpers. (Ergänzung 9/85/69) 7/85/20
Technik: Bit-Editor. Zeichensatz-EPROMs für die Videx-Karte. Joachim Klamt. 7/85/29
Assembler: 6502 leicht gemacht. Ulrich Stiehl. 7/85/33
Nachtrag: Nachtrag zum Pascal-RAM-Disk-Driver. 7/85/52
CP/M: Wordstar mit allen FX-80-Schriftarten. Dipl.-Ing. H.A. Rohrbacher. 7/85/57
Hobby: Pascal-Preisausschreiben. 7/85/62
Kurzbericht: No Orchids für Miss Lisa. Lisa und die Folgen. Ulrich Stiehl. 7/85/67
Assembler: Hex-Dez-Konvertierung für 32-Bit-Zahlen. Harald Grumser. 7/85/69
Assembler: Vorlesestunde. Apple und S.A.M., ein hilfreiches Gespann. Dr. Jürgen B. Kehrel. 7/85/71
Leserbriefe: 7/85/73
Test: MEGACORE. Festplatte mit 10 MBytes. Harald Grumser. 7/85/76
Test: Star Delta-10 und Grafstar-Interface. Karl-Walter Bott. 7/85/77

Heft 8/85

Technik: Testprogramm für Apple-II-Diskettensysteme. Analyse von Disketten, Laufwerk und Controller. Dipl.-Ing. Gerhard Berg. 8/85/6
ProDOS: ProDOS für Anfänger. Teil 3: Geheimnisse von BSAVE und BLOAD. Ulrich Stiehl. 8/85/18
Grafik: Graf-quattro. Teil 3: Tricks über Tricks. Nino G. Barbieri. 8/85/24
Grafik: Double-Lores-Applesoft-Erweiterungen. Karl-Walter Bott. 8/85/32
CP/M: Adreßverwaltungsprogramm mit dBASE II. Ing. (grad.) Ernst Fischer. (Ergänzung 10/85/21) 8/85/40
CP/M: Wordstar druckt internationale Zeichensätze. Frei definierte Sonderzeichen auf dem FX-80 nutzen. Dipl.-Ing. H.A. Rohrbacher. 8/85/45
Pascal: Tips und Tricks in Pascal. Teil 1: Die versteckte Prozedur „Idsearch" oder wie man Schlüsselwörter halbfett druckt. Dieter Geiß. 8/85/48
Nachtrag: SUPERDUMP und Apple IIc. 8/85/50
Assembler: Assembler-Pseudo-Opcode-Referenztabelle. Dr. Jürgen B. Kehrel. 8/85/51
Assembler: Fakultäten. Roland und Manfred Fietkau. 8/85/56
Macintosh: Microsoft Basic leicht gemACht. Teil 4: Die Grafik. Pit Captain. 8/85/60
Hobby: Grafik-Demonstrationen. Ralf Knoke. 8/85/67
Kurzbericht: Applepreise = Mondpreise? Ulrich Stiehl. 8/85/67
Nachtrag: Diversi-DOS 2-C. 8/85/68
Hobby: Zeichenjagd. Ein Einzeiler. Hans-Peter Lendle. 8/85/69
Test: Die Polaroid-Foto-Systeme. Thomas Bühner und Prof. Dr. Klaus Hausmann. 8/85/71
Produkt: Das Bildverarbeitungssystem MAGIC. 8/85/74
Kurzbericht: Siemens entdeckt neue Primzahl. 8/85/74
Produkt: TRICARD – Multifunktionskarte für Apple IIe. 8/85/74
Produkt: Copy-Killer jetzt in deutsch. 8/85/76
Produkt: Microfloppy mit 2 x 1 MByte. 8/85/76
Test: ProDOS-Debugger BUGBYTER. Dr. Jürgen B. Kehrel. 8/85/76
Test: Beagle Graphics. Rolf W. Becker. 8/85/77

Heft 9/85

Recht: Ave auctor, plagiarii te salutant. Software und Zitatrecht. Ulrich Stiehl. 9/85/6
Recht: Urheberrechtsnovelle. Ulrich Stiehl. 9/85/11
DOS 3.3: Ein neues FID für DOS 3.3. Mit Auto-FID für RAM-Disk-Besitzer. Harald Grumser. 9/85/12
ProDOS: ProDOS-Fastboot oder wie man ProDOS in 6 Sekunden bootet. Arne Schäpers. Nach einer Idee von Ulrich Stiehl. 9/85/20
Hardware: Super-HGR für NEC- und ITOH-Drucker. Rainer Hammerschmidt. 9/85/30
Applesoft: Umwandlung in Großbuchstaben. Harald Grumser. 9/85/31
Assembler: Flag Monitor. Eine Taktzähler-Utility. Michael G. Schneider. 9/85/32
Pascal: Tips und Tricks in Pascal. Teil 2: Kann man den P-Code optimieren? Dieter Geiß. 9/85/40
MBASIC: MBASIC für den Applesoft-Profi. Jörg Lange. 9/85/48
Hobby: Pascal-Preisausschreiben. Die Gewinner. 9/85/58
Händler-Profil: HIB. Der Computerladen in Nürnberg. 9/85/59
Test: Erfahrungsbericht OPERATOR I. Rainer Hammerschmidt. 9/85/60
Test: Die OPERATOR IIe. Harald Grumser. 9/85/60
Test: Dazzle Draw und Mousepaint. Dieter Charchot. 9/85/61
Test: DMP-Charger – eigene Zeichensätze auf dem Matrixdrucker. Harald Grumser. 9/85/62
Test: Apple II/IIe Assembler-Kurs. Dr. Jürgen B. Kehrel. 9/85/62
Test: Deutsche Sprachausgabe für den Apple. Harald Grumser. 9/85/63
Kurzbericht: 512K für Apple III. 9/85/64
Kurzbericht: Apple IIe-Emulation für Apple III. 9/85/64
Kurzbericht: Lisa/Mac-Screenswitcher. 9/85/64
Kurzbericht: 640K-Drive für IIc. 9/85/64
Kurzbericht: Apple-Paketpreise. 9/85/64
Kurzbericht: Apple-Geschichte. 9/85/64
Kurzbericht: Mactablet. 9/85/64
Kurzbericht: Macpack. 9/85/64
Kurzbericht: Incircuit-Emulator. 9/85/64
Kurzbericht: Erno Unibox. 9/85/65

Heft 10/85

Grafik: Graf-quattro. Teil 4: Der Grafik-Editor. Nino G. Barbieri. 10/85/6
Grafik: Applesoft-Interpreter-Erweiterungen für Double-Hires. Dr. Wolfgang Braun. 10/85/14
Nachtrag: Tips zur Konvertierung der Adreßverwaltungs-Dateien von DOS 3.3 nach CP/M. 10/85/21
Hardware: Die neuen ROMs für den Apple IIe. Teil 1: Grundlagen. Teil 2: Standardein- und -ausgabe. 10/85/22 Ulrich Stiehl
Hardware: Die neuen ROMs für den Apple IIe. Teil 3: Applesoft-Interpreter. Harald Grumser. 10/85/34
Hardware: EPROM-Programmiergerät für den Apple II. Dr. Roland Schule. 10/85/40
Pascal: Pascal-Kompaktkurs für Applesoft-Programmierer. Hinweis für Sammeldisk. Ulrich Stiehl. 10/85/46
Recht: Rechtsprechung zum Software-Urheberrecht. RA Dieter Naber. 10/85/47

Pascal: Tips und Tricks in Pascal. Teil 3: P-Code-Cruncher für 128K-Pascal. Dieter Geiß. 10/85/50
Pascal: Pascal 1.2. Eine Richtigstellung. Bernard Condrau. 10/85/60
Nachtrag: Schwierigkeiten mit den Graf-quattro-Cursoren. Nino G. Barbieri. 10/85/63
Hobby: Was ist Logo? Eine Kurzeinführung. Kurt Rudl. 10/85/64
Hobby: Puzzle mit der Maus. Joachim Mette. 10/85/65
Buch: Lerne BASIC auf dem Apple. 10/85/67
Buch: Mathematik auf dem Apple II, IIe, IIc. 10/85/67
Buch: Trainingsbuch zu APPLESOFT-BASIC. 10/85/67
Buch: Apple II für Technik und Wissenschaft. 10/85/67
Buch: Macintosh (Anwenderhandbuch). 10/85/67
Buch: Macintosh. 10/85/67
Buch: Das Apple Macintosh Buch. 10/85/67
Leserbriefe: 10/85/68
Test: Speedemon-Karte. Ulrich Stiehl. 10/85/71
Test: Die 16K-Akku-Karte LC-85. Harald Grumser. 10/85/72
Test: Zwei neue Apple-IIe-Kompatible. Apple-IIe-Kompatibler SCSe. Jürgen Geiß. 10/85/73
Test: Zwei neue Aplle-IIe-Kompatible. Apple-IIe-Kompatibler SCSes. Harald Grumser. 10/85/74
Kurzbericht: 4.000 Frei-Programme. 10/85/75
Kurzbericht: Buchhaltungsprogramm BUCH. 10/85/75
Kurzbericht: Präsentationsgrafiken mit (G)ROGRAF. 10/85/75
Kurzbericht: Melco-Druckpuffer von Watanabe. 10/85/75
Kurzbericht: Digitalisiertabletts höchster Präzision. 10/85/75
Kurzbericht: Neuer Apple-Kompatibler. 10/85/75
Kurzbericht: Flugsimulator mit deutscher Anleitung. 10/85/75
Kurzbericht: OKI-Drucker für Apple II. 10/85/76
Kurzbericht: Macprint-Satzsystem. 10/85/76
Kurzbericht: Macprolog2-Interpreter. 10/85/76
Kurzbericht: Turbo-Lader Graph. 10/85/76
Kurzbericht: Sound Sampling System. 10/85/76
Kurzbericht: Apple dementiert „Heute Journal". 10/85/76
Kurzbericht: Apple GmbH erhöht Stammkapital. 10/85/76
Kurzbericht: Apple-Händler-Netz umstrukturiert. 10/85/76
Kurzbericht: MAUS: Mac an Hochschulen. 10/85/76
Kurzbericht: Mac-Leasing für Studenten: 10/85/76
Kurzbericht: Leasing von Apple-Computern. 10/85/76
Kurzbericht: BTX für Apple IIc. 10/85/77
Kurzbericht: Teletex für den Apple II. 10/85/77
Kurzbericht: Mac-Pressekonferenz bei Apple in München. Harald Grumser. 10/85/77

Heft 11/85

Grafik: Trickfilmgrafik für Spielprogramme. Mit einem Sprite-Editor. Bernd Klawonn. 11/85/6
Drucker: Großformatiger HGR-Ausdruck. Mark Liebrand. 11/85/20
DOS 3.3: Disk-Chirurg. Ein Programm zur Überprüfung schadhafter Diskettensektoren. Wolf-J. Faust. 11/85/24
ProDOS: ProDOS-Anpassung für Laufwerke mit 40 bis 160 Spuren. Horst Hanke und Hans-Georg Hüneke. 11/85/29
Assembler: Z80-Assembler-Programmierung unter CP/M. Dr. H. Kersten. 11/85/33
Applesoft: Print-Using. Rechtsbündige Zahlenformatierung. Ulrich Stiehl. 11/85/49
Applesoft: Referenztest. Zeilenverweise in Applesoft-Programmen. Ludger T. Engbert. 11/85/55
Pascal: Tips und Tricks in Pascal. Teil 4: Die Compiler-Optionen. Dieter Geiß. 11/85/57
Test: Utilities von Beagle Brothers. Franz-Josef Hüskens. 11/85/65
Test: Triple-Dump. Ein Bildschirm-Druckprogramm. Rolf W. Becker. 11/85/66
Test: Diversi-DOS 4.1-C mit Garbage-Collection. Rudolf Rötering. 11/85/67
Test: BASF-Flexydisk 1X und 1D. Ulrich Stiehl. 11/85/68
Test: Die Microsoft-Premium-Softcard für den Apple IIe. Ulrich Stiehl. 11/85/70
Vermischtes: Peeker-Sammeldisketten #1-#11. 11/85/71

Heft 12/85

Hobby: Cosmo-Crumble. Ein Weltraum-Action-Spiel. Oliver Steinmeier und Alexander Niemeyer. 12/85/6
Drucker: Imagewriter kurz und bündig. Die wichtigsten Steuerbefehle. Thorsten Schunk. 12/85/8
Grafik: ASCII-Text im HGR-Bildschirm. Mit einem Zeichengenerator-Programm. Markus Geltenpoth. 12/85/16
Applesoft: Komfortabler INPUT von Strings. Ulrich Kußmann. 12/85/20
Applesoft: Tabelle der wichtigen Applesoft-Interpreter-Routinen. Harald Grumser. 12/85/30
Pascal: Pascal-Kompaktkurs. UCSD- und Turbo-Pascal. Teil1: Grundlagen für Applesoft-Programmierer. Ulrich Stiehl. 12/85/33
Vermischtes: Richtlinien für Autoren. 12/85/61
Turbo: CP/M-56K-RAM-Disk unter Turbo-Pascal. Bernd Eichinger-Wieschmann. 12/85/62
Kurzbericht: Kleine Mikrocomputer-Chronik. Ulrich Stiehl. 12/85/64
Leserbriefe: Pascal-Leserbriefe. 12/85/68
Vermischtes: Jahresinhaltsverzeichnis. Peeker 1/84 bis 12/85. 12/85/72
Test: MS-Basic 2.0 für den Mac. Pit Capitain. 12/85/76
Produkt: Kyan-Pascal. 12/85/77

Jahresinserenten-verzeichnis 1985

aaa-electronic, Freiburg
ABACOMP GmbH, Frankfurt
AFC Computer GmbH, Köln
APPLELAND WALLISER & Co. Heilbronn
Beam-Verlag, Marburg
Brainware GmbH, Wiesbaden
Gerhard Brecht, Stuttgart
Bühler Elektronik, Baden-Baden
CCP Datentechnik, Hamburg
COMPTRON Hoffmann, Computertechnik, Kiel
Computerladen, Memmingen
Computerware, Frankfurt
Copy-Team, Erlangen
CP/D, Düsseldorf
CPS Datenservice, Offenbach
CVB Computervertrieb, Hamburg
Jürgen Dahlhoff Computertechnik, Soest
DATA BECKER GmbH, Düsseldorf
Deutsche Rechner Gesellschaft, Dreieich
Digitalanalog, Krombach
Ulrich Dobbertin, Brühl
D.O.S. Computersysteme, Schwäbisch Hall
Electronic Spezial Service GmbH, Gießen
Erbrecht Computer Related Products, Hamburg
ERPHI Electronic, Baldham
Franzis Verlag, München
German Empire Software, Bremen
Hamburger Computerversand, Hamburg
HIB Computerladen, Nürnberg
Norbert Hunstig, Münster
IBS Computertechnik, Bielefeld 14
Ingenieurbüro Fricke, Berlin
Inter-Data GmbH, Singen
Interkom Electronic GmbH, Isernhagen 4
Intus Lern-Systeme AG, Astano
Klaus Jeschke, Kelkheim
KFC, Königstein
Lucius Computer Programme, Dinslaken 1
W.-D. Luther Verlag, Sprendlingen
McGraw-Hill Book Co. GmbH, Hamburg
MCI GmbH, Bergisch Gladbach
Memsoft GmbH, Frankfurt
Erich-Willi Meyer, Frohnhausen
Micromint Streil, Erkrath
Ulf Mohwinkel Electronic, Leverkusen
Arnd Nolte, Langen
Novocomp Datensysteme GmbH, Trier
Pandabooks, Berlin
Pandasoft, Berlin
Röckrath, Aachen
R + R Electronic, Heidelberg
SCS Sander Computer Systeme, Remscheid
Softline, Oberkirch
Dipl.-Ing. R. Springmann, Hannover
Summagraphics GmbH, München
Sybex-Verlag GmbH, Düsseldorf
Schappach Computer, Mannheim
Tewi Verlag, München
TLK Hard- und Software, Münster
Tombstone Micro, Berlin
UCD Computersysteme, Freiburg
Ueding Electronics, Menden
VOBIS, Aachen
Weidemann Electronic, Rengsdorf
Karl-Heinz Weiss, Wilhelmshaven

MS-Basic 2.0 für den Mac

getestet von Pit Capitain

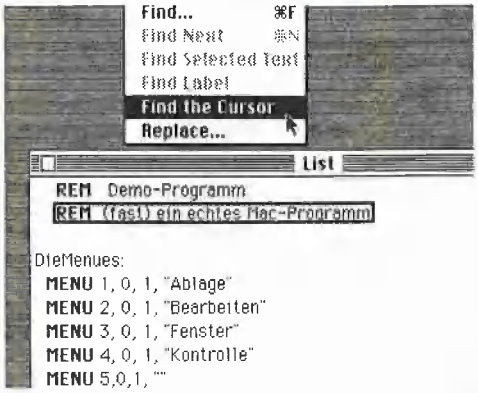


Abb. 1

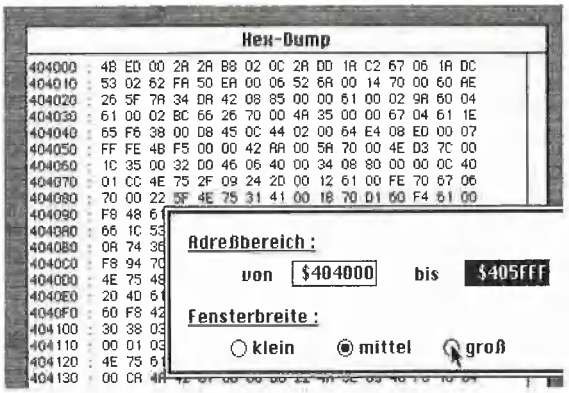


Abb. 2

Seit einiger Zeit ist eine neue Version des Microsoft-Basic für den Macintosh erhältlich, nämlich die Version 2.0. Da die bisherige Version 1.0 bereits im Peeker ausführlich beschrieben worden ist (s. 2/84, 1/85, 4/85, 5/85, 8/85), werden hier nur die Besonderheiten der Neuversion vorgestellt.

Die alte Version 1.0 des Basic war von Microsoft recht schnell (etwas voreilig?) auf den Markt gebracht worden, und das merkte man ihr auch an. Sie war sehr langsam und unterstützte nur wenige Möglichkeiten des Macintosh.

Nun gibt es die neue Version 2.0, die zwar schon besser ist, aber im Vergleich zu anderen Programmen für den Mac immer noch Wünsche offen läßt. Microsoft schreibt denn auch im neuen Handbuch : „MS-Basic is always growing, changing, and improving." Dies erscheint mir persönlich auch notwendig.

Der äußere Eindruck

Das Handbuch zu der neuen Version ist besser geworden. Vor allem die ROM-Routinen, die man von Basic aus aufrufen kann, sind nun recht ausführlich dokumentiert.

MS-Basic 2.0 gibt es in zwei unterschiedlichen Formen: eine *dezimale* Version für den kaufmännischen Bereich (wie Version 1.0) und eine *binäre* Version für den wissenschaftlichen Bereich (schnellere Arithmetik).

Der Unterschied zwischen beiden Versionen liegt einzig in der Art, wie Zahlen intern dargestellt werden. So gering dieser Unterschied auch erscheinen mag, die Basic-Programme laufen nur unter *einer* der beiden Versionen, nicht mit beiden!

Nachdem man eine der beiden Basic-Varianten gestartet hat, sieht man gleich an den Menüs, daß sich etwas verändert hat.

Die Befehle des „Edit"-Menüs kann man nun endlich auch über die Tastatur aufrufen, was sehr angenehm ist. Daneben gibt es ein „Search"-Menü, mit dem man bestimmte Stellen im Programmtext suchen bzw. ersetzen kann. Im „Run"-Menü gibt es den Befehl „Step", mit dem man Programme Schritt für Schritt durchtesten kann. Wie das dann auf dem Bildschirm aussieht, zeigt **Abb. 1**. Der Befehl, der gerade ausgeführt wird, ist umrahmt. Das neue „Windows"-Menü dient zum Öffnen der unterschiedlichen Fenster.

Die beste Änderung für das Arbeiten mit dem Basic besteht darin, daß der Editor nun endlich *bildschirmorientiert* arbeitet. Das bedeutet, daß man nicht mehr nur einzelne Zeilen auf einmal bearbeiten kann, sondern mehrere Zeilen und beliebige Abschnitte gleichzeitig.

Das Editieren geschieht direkt im „List"-Fenster. Man erspart sich also das mühsame Kopieren der Zeilen in das Kommandofenster.

Das „List"-Fenster hat sich übrigens noch weiter verbessert: Zum einen werden die Basic-Befehle fett gedruckt, was die Lesbarkeit der Programme deutlich erhöht (siehe Abb. 1), zum anderen läßt sich das Fenster ganz einfach vergrößern. Mit einem Doppelklick in der Titelzeile wird das Fenster so groß wie der ganze Bildschirm. Nach einem erneuten Doppelklick schrumpft es wieder automatisch auf die alte Größe zusammen.

Labels und Subprograms

Der bedeutsamste Unterschied zum Basic 1.0 ist: Man braucht keine Zeilennummern mehr! Das heißt, man kann Zeilen entweder gar nicht oder mit einem Label im Klartext markieren (siehe Abb. 1). Dadurch ist das Programmieren sehr viel komfortabler geworden. Man kann zum Beispiel schnell einmal einen Abschnitt des Programms an eine andere Stelle verschieben, ohne daß man gleich alle Zeilen neu durchnumerieren muß. Wenn man unbedingt will (oder wenn einem keine neuen Labels mehr einfallen), kann man natürlich auch Zeilennummern verwenden.

Eine weitere Änderung, die das Programmieren erleichtert, ist die Möglichkeit, sogenannte „Subprograms" zu schreiben. Es handelt sich hierbei um spezielle Unterprogramme. Sie haben eigene (lokale) Variablen, die nichts mit den Variablen außerhalb des Unterprogramms zu tun haben (ähnlich wie bei den Prozeduren in Pascal). An diese Unterprogramme können, ebenfalls wie in Pascal, Parameter übergeben werden, wobei sogar Arrays mit variabler Länge zugelassen sind (wie in Modula). Die lokalen Variablen behalten ihren Wert von Aufruf zu Aufruf bei, was ebenfalls vorteilhaft sein kann.

Leider können solche Unterprogramme nicht verschachtelt und auch nicht rekursiv aufgerufen werden.

Töne, Bilder und Fenster

Das MS-Basic 2.0 wurde in einigen neuen Punkten an die Fähigkeiten des Macintosh angepaßt.

Mit den Befehlen „SOUND" und „WAVE" kann man Töne erzeugen. Dabei können bis zu vier Stimmen gleichzeitig mit unterschiedlichen Hüllkurven erzeugt werden. Die Töne werden – wie beim Mac üblich – „asynchron" ausgegeben, d.h. das Programm läuft weiter, während die Musik spielt. Dadurch verlangsamt sich dann natürlich die Ausführung des restlichen Programms.

Bisher konnte man Texte von anderen Mac-Programmen über das „Clipboard" in Basic-Programme übernehmen. Das gleiche ist nun auch mit *Bildern* möglich.

Diese Bilder kann man auch durch ein Basic-Programm selbst erzeugen und auf dem Bildschirm ausgeben lassen. Dafür gibt es die „PICTURE"-Befehle. Die Bilder können natürlich auch auf dem Imagewriter ausgedruckt werden.

Neben diesen nützlichen Eigenschaften kann man beim Basic 2.0 vor allem auch solche Programme schreiben, wie man sie vom Macintosh her kennt, also mit eigenen Menüs, eigenen Fenstern, Dialogboxen usw. Dazu dienen die Befehle „MENU", „WINDOW", „DIALOG" u.a.

Ein solches Programm kann zunächst seine Menüs und Fenster erzeugen und dann so lange warten, bis der Benutzer etwas mit der Maus anwählt. Danach werden die entsprechenden Unterprogramme aufgerufen, und das Ganze wiederholt sich so lange, bis das Programm beendet ist.

Ein Beispiel hierfür zeigt **Abb. 2**. Diese Programme sehen zwar gut aus, aber die Erzeugung der Menüs und Fenster ist recht aufwendig. Auch sind solche Programme langsamer als die „normalen" Basic-Programme.

Fazit

Die neue Version 2.0 des MS-Basic hat viele Schwachstellen der alten Version 1.0 beseitigt. Das Programmieren ist deutlich komfortabler geworden. Da die Fähigkeiten des Macintosh nun ebenfalls in höherem Maße unterstützt werden, kann man den Basic-Programmierern dieses Basic empfehlen, wenngleich die Programmausführung immer noch nichts mit der Geschwindigkeit eines 16-Bit-Computers zu tun hat.

Kyan-Pascal

Kyan-Pascal ist ein völlig neuartiges Pascal-System für den Apple II+/e/c, das von der Firma Kyan Software in San Francisco entwickelt wurde und als Version 1.0 im April dieses Jahres erstmals ausgeliefert wurde. Inzwischen gibt es die neueste Version 1.2 vom 14. August 1985.

Vor- und Nachteile von Kyan-Pascal

1. UCSD-Pascal-Implementierungen für den Apple II laufen zwar unter dem 6502-Prozessor und sind deshalb auf den Apple bestmöglich angepaßt, doch wird nur der relativ langsame Pseudo-Code erzeugt. Z80-Pascal-Implementierungen sind demgegenüber zwar in der Regel als Vollcompiler ausgelegt, doch wird erstens eine Z80-Karte benötigt und zweitens läßt die Anpassung an den Apple II, z. B. bezüglich der HGR-Grafik, zu wünschen übrig.

   Kyan-Pascal erzeugt einen echten 6502-Objektcode. Es vereinigt damit die Geschwindigkeitsvorteile eines Vollcompilers mit der bestmöglichen Anpassung an den Apple II.

2. Kyan-Pascal läuft unter allen denkbaren Apple-Konfigurationen (II+/e/c). Man kann es bereits mit Nutzen einsetzen, wenn man nur über eine Minimalkonfiguration mit Apple II+/e ohne 80-Zeichenkarte und ohne zweites Laufwerk verfügt, weil auf der (ungeschützten) Programmdiskette noch über 40.000 Bytes für die Arbeitsdatei frei sind. Wer besser ausgestattet ist, kann eine 80-Zeichenkarte, ein zweites Laufwerk sowie eine RAM-Disk verwenden.

3. Kyan-Pascal läuft unter dem (mitgelieferten) Betriebssystem ProDOS und kann deshalb auch in Verbindung mit größeren Massenspeichern (Festplatten sowie 80-Spur-Disketten usw.) problemlos eingesetzt werden.

4. Im wesentlichen besteht Kyan-Pascal aus einem sehr brauchbaren Fullscreen-Editor (in zwei Versionen für 40- und 80-Zeichendarstellung), der auch als Textverarbeitungsprogramm benutzt werden kann, einem Compiler und einer Runtime-Library (Bibliotheksdatei).

   Fertig compilierte Programme können in Verbindung mit der Bibliotheksdatei unabhängig von Kyan-Pascal als ProDOS-Systemdateien gestartet werden. Die Weitergabe der Runtime-Library wird von der Firma Kyan Software ohne Zahlung von Lizenzgebühr gestattet.

5. Der Kyan-Pascal-Befehlssatz lehnt sich eng an Standard-Pascal an, so daß beispielsweise Befehle wie GOTOXY usw. fehlen. Für String-Verarbeitung und HGR-Grafik werden jedoch Include-Dateien mitgeliefert. Dieser zunächst gravierend erscheinende Nachteil wird jedoch – wie wir meinen – durch zwei entscheidende Vorteile aufgehoben:

   a) Ein laufendes Kyan-Pascal-Programm befindet sich in der von Applesoft her gewohnten Monitor-Umgebung, d. h. der Pascal-READLN-Befehl wird z. B. durch die Monitor-Routine GETLN (entspricht dem Applesoft-INPUT-Befehl) realisiert. Der Aufruf von Monitor- oder gar Applesoft-ROM-Routinen ist deshalb, wenn man auf die Zero-Page Rücksicht nimmt, grundsätzlich möglich.

   b) In einen Kyan-Pascal-Quelltext können beliebig viele echte 6502-Quelltexte eingebaut werden. Man kann deshalb Kyan-Pascal auch als 6502-Assembler benutzen. Ein Beispiel-Programm macht dies deutlicher:

```
PROGRAM DEMO;
BEGIN
  WRITELN ('Jetzt 6502');
#a {Beginn Assembler}
HOME EQU $FC58
BELL EQU $FBDD
  JSR HOME ;Bildschirm löschen
  JSR BELL ;Piepston ausgeben
  {usw.}
#{Ende Assembler}
  WRITELN ('Das war 6502');
END.
```

   Auf diese Weise ist es prinzipiell möglich, beliebige Assemblerroutinen in Kyan-Pascal-Programme einzubauen und darüber hinaus die in Standard-Pascal nicht vorhandenen Befehle zu kompensieren.

6. Kyan-Pascal scheint uns sowohl für Anfänger, insbesondere auch im Schulunterricht, als auch für Fortgeschrittene bestens geeignet zu sein. Anfänger können sich auf den Standard-Befehlssatz beschränken, während Fortgeschrittene die Assembler-Features ausnutzen können. Übrigens erzeugt Kyan-Pascal beim Compilieren ein echtes 6502-Quellcode-Listing, und zwar auch für die Pascal-Befehle, so daß ein nachträgliches Optimieren des Assembler-Quellcodes möglich ist.

Einladung zum Sammelbezug

Kyan-Pascal kostet in den USA 69,95 Dollar. Unter Berücksichtigung der Auslandseinfuhr- und Überweisungskosten muß man umgerechnet über DM 200,– bezahlen. Damit Sie diese interessante Pascal-Implementierung so günstig wie möglich beziehen können, bieten wir Ihnen als Peeker-Serviceleistung die Teilnahme am Sammelbezug an. Als Stückpreis haben wir DM 170,– (inkl. Porto, Verpackung und MwSt) festgelegt. Voraussetzung hierfür ist allerdings, daß eine ausreichende Anzahl von Vorbestellungen eingeht. Werten Sie bitte diese Aktion als einen Versuch, Software im Sammelbezug preiswert zu erwerben. Sollten sich nicht genügend Interessenten melden, so müssen wir dieses Service-Angebot wieder vergessen. Kyan-Pascal ist freilich auch bei einschlägigen Importeuren (für z. Zt. ca. DM 270,–) erhältlich.

Wie wird's gemacht? Senden Sie uns bis spätestens 31.1.1986 Ihre Bestellung über Kyan-Pascal zum Preis von DM 170,– inkl. Versandspesen. Teilnehmen können alle Peeker-Leser. *Ihre Peeker-Redaktion*

Vorschau auf Peeker 1/86

Wie funktioniert Quicksort?
Mit einem Applesoft-Demo

Superschnelles Quicksort in Assembler

Vokabeltrainer in Applesoft
Mit 1000 Wörtern Deutsch-Englisch

Applesoft-Grafik-Erweiterung

PROTODOS
Konvertierung von ProDOS- in DOS-Textfiles

Designer
Zeichensatz-Generator in UCSD-Pascal

Read-Pascal
Konvertierung von UCSD- in Turbo-Textfiles

Der Apple IIe als Werkzeug für Betriebswirte
Mit Simplex-Anwendungsbeispielen

Serielle Schnittstellen beim Apple IIc

Testberichte:
Video-1000-Interface
Star-Drucker SG-15
Prometric
Microbuffer-Karte
Mockingboard
Write Choice
Ampersoft
FSS-280-Disk-Subsystem

Inserentenverzeichnis Peeker 12/85

Peeker-Sammeldisk #12 (DOS 3.3; Heft 12/85)

COSMO CRUMBLE
T.CC2
T.CC3
T.CC4
T.CC5
T.CC6
T.CC8
CC2
CC3
CC4
CC5
CC6
CC7
CC8
CC.LEVELS
(1) Reaktionsspiel für Tastatur oder Joystick; (2) Heft 12/85, S. 6; (3) II+, IIe oder IIc; wahlweise Joystick; (4) DOS 3.3; (5) RUN COSMO CRUMBLE

IW.DEMO
T.IW.IN
IW.IN
T.IW.OUT
IW.OUT
ECHO
(1) Initialisierung des Imagewriters durch mit IW.DEMO generierbare, BRUN-fähige Steuer-Sequenz-Programme; (2) Heft 12/85, S. 12; (3) IIe oder IIc (II+ mit Einschränkungen, wegen der Tastatur); Imagewriter (beim IIe mit Super-Serial-Card); (4) DOS 3.3 oder ProDOS; (5) RUN IW.DEMO

ASCII.EDITOR
COPY.TEXT.DEMO
T.COPY.TEXT
COPY.TEXT
ASCII.CODES
(1) Erstellung eines eigenen Zeichensatzes; Kopieren des 40-Z/Z-Text-Bildschirms in den HGR-Bereich; (2) Heft 12/85, S. 16; (3) speziell II+, aber auch IIe oder IIc mit Einschränkungen; (4) DOS 3.3; (5) RUN ASCII.EDITOR; RUN COPY.TEXT.DEMO; (6) Nach FLASH werden Großbuchstaben auf dem HGR-Bildschirm als Kleinbuchstaben ausgegeben

GETTEXT.DEMO
T.GETTEXT
GETTEXT
GETTEXT.PRODOS
(1) Hilfsprogramm zur Erweiterung des Tastatur-INPUT-Befehls durch Editierfunktionen; (2) Heft 12/85, S. 20; (3) II+, IIe oder IIc (40 und 80 Z/Z, auch Videx); (4) DOS 3.3 (GETTEXT) oder ProDOS mit BASIC.SYSTEM 1.0 (GETTEXT.PRODOS); (5) RUN GETTEXT.DEMO; (6) für Videx und ProDOS leichte Anpassung von GETTEXT.DEMO erforderlich, s. Heft

RAMDISK.PAS
(1) Installierung einer CP/M-56K-RAM-Disk unter Turbo-Pascal; (2) Heft 12/85, S. 62; (3) IIe mit 64K-Karte; (4) CP/M 2.20 56K; Turbo-Pascal 2.0 oder 3.0; (5) RAMDISK (unter CP/M, nach Konvertierung mit APDOS und Compilieren auf Diskette); (6) entspricht der RAM-Disk aus Heft 6/1985, S. 55

JAHRESINHALT
STICHWORT.SUCHER
STICHWORT.SUCHER.OBJ
(1) Jahresinhaltsverzeichnis als binäre Textdatei mit Programm zur Suche von Stichwörtern; (2) Heft 12/85, S. 72; (3) II+, IIe oder IIc; 40 und 80 Z/Z (auch Videx); (4) DOS 3.3; oder ProDOS (5) RUN STICHWORT.SUCHER

Leser werben Leser

»peeker« bietet Ihnen was!

Wer jetzt schenkt hat mehr von seinem Apple. Dafür schenkt »peeker« Ihnen etwas: Den praktischen Disk-Locher, der die Speicherkapazität Ihrer Disketten verdoppelt!

Sie wissen ja, wie gut der »peeker« Ihnen im täglichen Umgang mit Ihrem Apple behilflich ist. Und Sie brauchen Ihren »peeker« nicht mehr zu teilen oder auszuleihen.

Der blaue Disketten-Locher ist unser Geschenk an Sie für einen neuen »peeker«-Abonnenten. Denn wer einen Apple hat, der soll auch seinen »peeker« haben.

peeker 12/85

Bestellcoupon

Ich habe den neuen Abonnenten geworben und erhalte kostenlos den Disk-Locher.

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 und einsenden an:

»peeker«
Abonnementservice
Im Weiher 10
6900 Heidelberg 1

Hüthig
PUBLIKATION

MEGA-CORE

läßt keine Wünsche offen:

- 10 Megabytes in Ihrem Apple
- 4 Betriebssysteme im Zugriff
- Superstarkes Netzteil mit Kühlung

MEGA-CORE macht durch Einbau eines 10 MB Festplattenlaufwerks Ihren Apple //e zu einem XT.

MEGA-CORE besteht aus:

- 3 1/2" Festplatte mit 10 MB
- Harddiskcontroller auf einer Slotkarte
- 70 W Netzteil (12 W f. Platte / 58 W f. Apple)
- Lüfter zur Kühlung des Rechners
- Software zur Anpassung von vier Betriebssystemen, DOS, CP/M, PASCAL u. ProDOS
- Ausführliches deutsches Handbuch

MEGA-CORE macht Ihren Apple zum Profisystem:

- Jedes Betriebssystem bootet von der Platte
- Alle Betriebssysteme gleichzeitig auf der Platte bei extrem schnellem Zugriff
- Die ganzen 10 MB sind frei konfigurierbar

MEGA-CORE ist ein Produkt von:

FRANK & BRITTING
Elektronik Entwicklungs GmbH
Langestr. 4 Postfach 1129, 7529 Forst
Telefon: 07251 / 103068-69
Telex: 7822452 fub d

Die Harddiskcontroller-Spezialisten

MEGA-CORE kostet 4.560,— DM inkl. MwSt. beim örtlichen Fachhandel