DAS ELEKTRONISCHE MAGAZIN 6/86

INPUT 64

Infos · News · Programme · Unterhaltung · Tips

MACRO-ASSEMBLER

— extrem schnell
— eigener Editor
— bibliotheksfähig

Die versunkene Stadt

Dramatische Suchaktion mit dem Raumschiff

Zeichensatz-Hardcopy

Umlaute und Grafik schwarz auf weiß

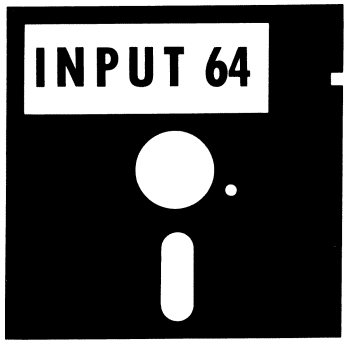
LISP 64 Teil 3

— Dialogprogramm Eliza
— Ein kleines Expertensystem

Mathe mit Nico, 64er-Tips, Spiel, Ratsel ...

Dokumentation und Bedienungshinweise

INPUT-ein Magazin aus dem Verlag Heinz Heise GmbH, Postfach 61 04 07, 3000 Hannover 61



Ab 4/85 auch auf Diskette-

direkt vom Heise-Verlag, INPUT-Vertrieb,
Postfach 610407, 3000 Hannover 61

für 19.80 DM inkl. Versandkosten + MwSt.-

Nur gegen V-Scheck!

Das Geschenk:

INPUT 64 V . Disk im Sixpack

Die Ausgaben 4/85 bis 9/85 der Disketten-Version bekommen Sie ab sofort zum Paketpreis von 90 DM.

Jetzt bestellen, **24.80 DM** sparen!

Direkt beim Verlag:
(Inclusive Porto und Verpackung) **Nur gegen V-Scheck!**

Verlag Heinz Heise GmbH · Postfach 610407 · 3000 Hannover 61

Kurs komplett

Jetzt als Sampler:
Die Serie BITS & BYTES IM VIDEO-CHIP

Alle Folgen des Kurses auf Kassette und Diskette. Eine grundlegende Einführung in die Programmierung des Video-Chip, mit Exkursen in die Binärarithmetik, Programmiertips und so weiter.

Überarbeitet und um einen Teil zur Multicolor-Grafik erweitert.

Kassette 17.80 DM
(mit SuperTape-Lader und Sicherheitskopie auf der Rückseite)

Diskette 24.80 DM

Direkt beim Versand: (inclusive Porto und Verpackung)

Nur gegen V-Scheck!

Verlag Heinz Heise GmbH · Postfach 610407 · 3000 Hannover 61

INHALTSVERZEICHNIS

INPUT 64

6/86

Liebe 64er-Besitzer(in)!

Der C64 bleibt auf der Erde!

In der Gerüchteküche brodelt es schon lange. Die verschiedensten Vermutungen und Behauptungen kursieren über die Zukunftschancen des C64. Wir wissen es jetzt genau: der C64 ist tot, es lebe der C64!

Wie von der westdeutschen Vertretung der Firma Commodore zu erfahren war, sind die Lagerbestände des C64 geräumt. Das Gerät wird nicht mehr produziert.

Doch, keine Panik, der C64 feiert eine Wiedergeburt. Er erscheint uns im neuen Gewand, ohne an inneren Werten verloren zu haben. Das Konzept des 128er-Gehäuses steht Pate für das neue Erscheinungbild. Der innere Aufbau soll zum "alten" 64er vollständig kompatibel bleiben. Das bedeutet: die umfangreiche Palette der existierenden C64-Software kann auf dem neuen Modell weiter verwendet werden.

Darüber hinaus wird jedoch einiges mehr geboten: Commodore plant ein Steckmodul, das dem 64er (auch dem bisherigen!) einen Zugriff auf circa 500 KByte ermöglichen soll! Außerdem ist ein neues Betriebssytem namens GEOS für den Rechner entwickelt worden. Die Computer-Welt wird also auch hier mit Fenstern, Pulldown-Menues und einer Maus gestaltet sein. Dieses Betriebssytem kann, wie schon vom CP/M-Betriebssytem beim C128 bekannt, von Diskette zugeladen werden. Es steht zu erwarten, daß der neue 64er mit dieser GEOS-System-Diskette ausgeliefert wird.

Damit sind selbstverständlich wesentlich höhere Grafik-Anforderungen zu erfüllen. Man darf gespannt sein, ob der lästige 40-Zeichen-Modus endlich vom Bildschirm verschwindet. Da zusammen mit diesem Rechner auch ein neuer Monitor (Typ 1801) auf den Markt kommen soll, sind dezente Hoffungen wohl berechtigt.

Leser von INPUT 64 werden also noch lange unser Magazin auf dem 64er genießen können und so über eine günstige Software-Quelle und aktuelle Informationen verfügen – für den "alten" und den "neuen" 64er.

Ihre INPUT 64 Redaktion

PS: Wem der Spaß durch den April-Scherz in INPUT 64 4/86 vergangen war, sollte die Auflösung der "rätselhaften Erscheinungen" in der folgenden Rubrik 'Leser fragen. . .' nachlesen.

Auf einen Blick: INPUT 64 - Betriebssystembefehle

Titel abkürzen	CTRL und Q
Hilfsseite aufrufen	CTRL und H
Inhaltsverzeichnis aufrufen	CTRL und I
Farbe für Bildschirm-Hintergrund ändern	CTRL und F
Rahmenfarbe ändern	CTRL und R
Bildschirmausdruck	CTRL und B
Programm sichern	CTRL und S

Laden von Kassette mit LOAD oder SHIFT + RUN/STOP

Laden von Diskette mit LOAD "INPUT*",8,1

Ausführliche Bedienungshinweise finden Sie auf Seite 29

Leser fragen...

Terminkalender historisch

Für meinen Verein möchte ich gerne mit Terminkalender aus INPUT 9/85 eine Übersicht der geschichtlichen Daten zusammenstellen. Leider läßt das Programm keine Termin-Eingaben vor 1985 zu. Kann man das Programm entsprechend ändern? (K.H Reiff)

Terminkalender wurde in PASCAL erstellt. Im Magazin wurde das Kompilat veröffentlicht. Für ein Source-Listing stand nicht genug Platz im Magazin zur Verfügung. Doch wie ein Kompilat ändern?. Der Weg, das Quell-Programm zu ändern und neu zu kompilieren, würde zu einer Zweit-Veröffentlichung führen. Aber, es geht auch anders. Terminkalender laden und nicht mit RUN starten. Dann im Direkt-Modus POKE 14221,108 und POKE 6298,108 eingeben und Terminkalender wieder abspeichern. Diese Version verarbeitet jetzt Termine ab 01.01.1900.

(d. Red.)

ChipList fragmentarisch

Bei dem abgespeicherten Programm ChipList läuft die Drucker-Steuerung nicht. Es erscheint eine Fehlermeldung (Krause)

ChipList wurde bei der Übertragung auf die Master-Diskette der letzten Zeilen im Programm beraubt. Zur Korrektur gehen Sie so vor:

- ChipList vom eigenen Datenträger in den Rechner laden,

- mit RUN starten und auf der Menue-Seite mit RUN/STOP + RESTORE abbrechen,

- POKE 12967,0 im Direktmodus eingeben (setzt Pointer auf 0, so daß die restlichen zerstörten Zeilen verschwinden),

- folgende Zeilen eintippen:

```
2297 GOTO 341
3000 REM DRUCKER
3010 OPEN1,DV
3020 PRINT#1:IFST=-128THEN3050
3030 POKE53280,0:CLOSE1:OPEN4,DV
3040 RETURN
3050 POKE53280,2:SYSPA,0,4,"DRUCKER EINSCHALTEN UND (RVS)TASTE(OFF)"
3060 WAIT203,63
3070 PRINT#1:IFST=-128THENCLOSE1:DR=0:RETURN
3080 SYSPA,0,0:LEFT$(BL$,40):GOTO3030
3090 GOTO 400

READY.
```

- im Direktmodus eingeben:

- CLR (RETURN)

- POKE 44,8 (RETURN)

- SAVE CHIPLIST.KORR ,dv

Chiplist wieder abspeichern, wobei dv = 1, 7 oder 8 sein kann für Kassette, SuperTape und Diskette. SuperTape muß natürlich vorher initialisiert werden, wenn dv = 7 verwendet wird.

(d. Red.)

Grafik-Bilder adressiert

Wie können Grafiken, die unter INPUT-BASIC erstellt und mit GSAVE abgespeichert wurden, so in eigene Programme eingebaut werden, daß man sie ohne Hires-Befehle auf den Bildschirm bringen kann? T.Günther, Düsseldorf

Die entsprechenden Grafik-Befehler müssen dazu in BASIC nachgebildet werden. Nachstehendes Listing verdeutlicht die einzelnen Schritte, die erledigen die Zeilen 60000 bis 60050 des beiliegenden Listings. Das sonst übliche Löschen des Grafikbildschirms erübrigt sich, da ja im nächsten Zug ein kompletes Bild nachgeladen wird.

2. den im C64-BASIC nicht vorhandenen Befehl BLOAD nachbilden; damit ist das Laden von Daten an eine benutzerbestimmte Startadresse gemeint. Diese Aufgabe erledigen die Zeilen 61000-61060.

In den restlichen Zeilen wird auf Tastendruck gewartet, und es wird in den Textmodus umgeschaltet. (d.Red)

Listing BLOAD

```
60000 poke53272,peek(53272)or2^3:rem einschalten
60010 poke53265,peek(53265)or2^5:rem der graphik
60020 zf=0:hf=1:rem schwarz auf weiss
60030 for i=1024to2023
60040 poke i,16*zf+hf:rem farbe setzen
60050 next i
61000 ga=8:rem oder 1/7 fuer kassette/supertape
61010 fi$="kreis":rem name des bildes
61020 ad=8192 : rem anfangsadresse des grafik-speichers
61030 sys57812 fi$,ga : poke185,0:rem name,geraete- und sek-adresse
61040 ha=int(ad/256):la=ad-ha*256:rem low- high byte aufteilung
61050 poke780,0:poke781,la:poke782,ha:rem load-parameter uebergeben
61060 sys 65493 : rem laden
61070 wait198,1:rem warten auf taste
61080 poke53272,peek(53272)and(255-2^3) :rem grafik wieder
61090 poke53265,peek(53265)and(255-2^5)

ready.
```

STAR SG-10C abgehängt

Damit das grausame Probierspiel ein Ende hat, bitte ich Sie, mir die Druckeranpassung Ihres INPUT-BASIC an den STAR SG-10C zu verraten (der Grafik-Ausdruck funktioniert nicht).

Bruckner,Hofheim

Ein Grafik-Ausdruck ist in der Konstellation INPUT-BASIC/Star SG-10C nicht möglich. Der Star erwartet nämlich im Einzel-Nadel-

Modus ein gesetztes 7.Bit, also Daten plus 128. (Laut Handbuch, englische Fassung, Seite 64)
(d.Red.)

INPUT 64 beschleunigt

. . . konnte ich nun genau und hinreichend prüfen, ob und in welchem Maße INPUT 64 mit PrologicDos (einem neu auf dem Markt erschienenen Floppy-Beschleuniger, d.Red.) kompatibel ist. Zu Ihrer Kenntnis und zu meiner Freude kann ich Ihnen berichten, das ich mit der Diskette keinerlei Probleme hatte. Alle Optionen funktionieren einwandfrei. . . . Ladezeiten: Titelbild 9 Sekunden, Kiki 5 Sek., TabCalc 2 Sek., Citron 5 Sek..
Bolloff, Brühl

Rätsel beschleunigt

Zum Rätsel aus Ausgabe 1/86 zunächst meine Anerkennung: Ihr Lösungsweg in INPUT 4/86 ist ja schon prima. Aber: Ich habe den Eindruck – wenn ich richtig zwischen den Zeilen gelesen habe – daß es für Sie NUR die Lösung auf Permutations-Basis gibt. Um Ihnen die (vermuteten) Scheuklappen zu nehmen, hier ein "Gleichnis": Ein Kinderzimmer, Vater, Sohn, jede Menge Spielsachen. Vater: "Heinz-Hugo, kannst Du mir alle grünen Bälle in diesem Zimmer geben?" Freudig beginnt der Sohn herumzuwuseln, schmeißt alles, was grün ist, in eine Ecke, und greift aus den 72 Teilen den Ball heraus. Gespannte Erwartung . . . Vater hebt den Zeigefinger: "Richtig wäre es gewesen, wenn du ZUERST alle Bälle herausgenommen hättest. Paß auf!" Er stapelt alle 576 Bälle in einer Ecke und nimmt den grünen heraus. Der Sohn beginnt zum ersten Mal, leise Zweifel an Vaters Kompetenz zu hegen.

*Ende des Gleichnisses . . . zu meiner Lösung: Man braucht eine Grundstruktur, die nur teilbare Zahlen erzeugt. Wenn man sich die einzelnen Stellen anschaut, sieht man, daß für die erste und zweite Ziffer nur vier Möglichkeiten, für die dritte, vierte und sechste nur zwei Möglichkeiten bestehen. Da Ziffer fünf festgelegt ist und die Ziffern sieben, acht und neun jeweils nur eine Möglichkeit bieten (Natürlich nur, wenn man die Teilbarkeit berücksichtigt!), gibt es also höchstens 4*4*2*2*2=128 verschiedene "teilbare" Zahlen. . . . Das entsprechende Permutations-Programm muß aber 576 Zahlen auf Teilbarkeit untersuchen. (I.v.d.Berg, Goch)*

Die Permutations-Lösung war als die "klassische" Problem-Lösung vorgestellt worden, nicht als einzige. Trotzdem ein Dankeschön für die Anregung. Wir haben, um unseren Lesern Tipparbeit zu ersparen, Hernn Bergs Programm in die Rätselecke dieser Ausgabe übernommen, sozusagen als "Nachschlag". Eine Kritik daran wollen wir uns aber nicht verkneifen: Man sollten aus FOR-NEXT-Schleifen nicht mit GOTO oder THEN heraussprungen. (d.Red.)

INPUT geklaut

. . .von einem Freund bekam ich die INPUT 2/85. Das Programm SCRIPTOR gefiel mir so gut, daß ich es unbedingt ausprobieren wollte. Nur weiß ich leider nicht, wie es funktioniert. Könnt Ihr mir bitte eine Anleitung für dieses Programm schicken? Ich wäre Euch sehr dankbar – und zwar für unsern CB-User Club Hailer.
(Benatzky, Gelnhausen)

Das, lieber Herr Benatzky, ist eben das Problem bei Raubkopien: die Anleitung fehlt. Hätten Sie diese Anleitung, könnten Sie auf der letzten Seite im Impressum (fettgedruckt!) lesen: "Die gewerbliche Nutzung ist ebenso wie die private Weitergabe von Kopien aus INPUT64 nur mit schriftlicher Genehmigung des Herausgebers zulässig." Von den – ebenfalls im Impressum für diesen Fall angekündigten – straf- und zivilrechtlichen Schritten sehen wir noch einmal ab, statt dessen die Empfehlung: Für 12.80 DM kann INPUT 2/85 beim Verlag nachbestellt werden.
(d.Red.)

Aprilscherz abgestürzt

. . . wird in Ausgabe 4/86 nach dem Titelbild der Bildschirm blau, unten erscheinen vier READY-Meldungen. Sytemabsturz?k (zahlreiche tel. Anfragen)

Anscheinend ist uns der April-Scherz zu gut gelungen. . . Ihr Rechner ist in Ordnung, auch das Programm ist nicht abgestürzt. Wir haben uns lediglich erlaubt, eine Verballhornung der bekannten Spiele der Kategorie "Space Invaders" als kleine Zugabe hinter das Titelbild zu hängen – man muß die April-Scherze ja nicht immer am 1.April machen. Wenn Ihnen das Spiel nicht gefällt, können Sie wie gewohnt mit CTRL und i oder CTRL und q das Inhaltsverzeichnis aufrufen. d.Red.)

Der Macro-Assembler:

INPUT-ASS

INPUT-ASS ist ein 2-Pass-Macro-Assembler mit integriertem Editor und optionaler Speicher- oder Peripherie-Orientierung, der sowohl symbolische als auch berechnete Ausdrücke verarbeitet.

Klingt gut, nicht? Wenn Sie gerade Ihr hundertunderstes Assembler-Programm schreiben, sowieso wenig Zeit haben und Begriffe wie Vorwärtsreferenz und Phase-Error im Schlaf erklären können, lesen Sie bitte nur die Kurz-Übersicht. (Wahrscheinlich lesen Sie dann nicht 'mal die, weil Sie zu den Leuten gehören, die Anleitungen erst nach dem fünften System-Crash zwischen den Comics hervorholen, oder?) Für alle anderen also:

Das Prinzip

INPUT-ASS verwandelt den für den Menschen verständlichen Text eines Assembler-Programms in für den Prozessor verständlichen 6502-Maschinen-Code. Aus dem Befehl JMP $FCE2 (Sprung zur Adresse 64738) wird die Byte-Folge $4C,$E2,$FC. Klartext-Befehle wie JMP, LDA und so weiter werden übrigens Mnemonics genannt. Nun sind – um gleich auf das Stichwort Symbolverarbeitung zu kommen – Adressen, Bytes und Bits genau das, was ein Computer gern verarbeitet. Da der Mensch aber den Dingen lieber einen Namen gibt, verarbeitet jeder vernünftige Assembler Symbole. Ein Befehl wie JMP RESET klingt doch wesentlich deutlicher als das tumbe JMP $FCE2.

Von berechneten Ausdrücken war oben die Rede. Das heißt einfach, daß der Assembler für Sie das tut, was er sowieso besser kann als Sie, nämlich mehr oder weniger komplizierte Rechenoperation durchführen. Wenn beispielsweise in die zehnte Reihe und fünfte Spalte des Bildschirms ein Zeichen geschrieben werden soll, kann man als Adresse angeben: 10*40+5+VIDEO. Natürlich muß dem Assembler vorher mitgeteilt werden, welchen Wert VIDEO hat, etwa durch die Zuweisung VIDEO=$400.

Wie erledigt ein Assembler nun seine eigentliche Arbeit, die Umwandlung von Programm-Text in Maschinen-Code? Stellen Sie sich einmal vor, Sie seien der Assembler. Ihnen wird Zeile für Zeile Source-Text vorgehalten, den Sie in den entsprechenden Maschinen-Code verwandeln sollen. Das ist in solchen Fällen nicht besonders schwierig, wo es nur um die Zuordnung von Mnemonics wie LDA zum Code $A9 geht. Das kann

man in einer Kartei (sprich Tabelle) zuordnen, und Zahlen werden ohnehin nur in ihre entsprechende Bit-Kombination umgewandelt. Labels (Sprungmarken, Einsprungspunkte für Unterprogramme und so weiter) sind dann kein Problem, wenn Sie zum Zeitpunkt des Aufrufs bekannt sind. Beispiel:

```
org $C000
:wait dex
bne wait
```

Die Anfangsadresse ist durch die ORG-Anweisung bekannt. Deswegen können Sie das Label WAIT eindeutig der Adresse $C000 zuordnen. Interessant wird es bei dem indirekten Sprung BNE WAIT. Sie – immer noch in der Rolle des Assemblers – müssen jetzt feststellen, an welcher Adresse WAIT liegt. Dies ist, werden Sie sagen, kein Problem, denn dem Symbol WAIT wurde bereits vor (!) dem Befehl BNE WAIT der Adresse $C000 zugeordnet. Richtig, Sie mußten sich diese Zuordnung nur merken. Darum jetzt der schwierige Fall:

```
org $C000
:begin    ldx #100
          jsr wait
          . . .
          . . .
:wait     dex
          bne wait
          rts
```

Beim Interpretieren des Programm-Textes stoßen Sie auf den Befehl JSR WAIT – und haben nicht den Schimmer einer Ahnung, an welcher Adresse sich WAIT befinden könnte, denn es taucht ja wesentlich später im Text auf. (Und heißt deswegen Vorwärtsreferenz !) Was tun? Die gängige Möglichkeit ist die: Sich erstens die Stelle merken, an der das noch unbekannte Label auftauchte. Zweitens, so zu tun, als ob nichts gewesen wäre, und den Programmzähler um die richtige Länge erhöhen. Denn es ist bekannt, daß ein JSR-Befehl immer drei Bytes lang ist. Beim weiteren Durchsuchen des Assembler-Textes stoßen Sie dann irgendwann auf das Label WAIT und merken sich natürlich dessen Adresse. (Dieses Merken, das heißt die Zuordnung von Labels bzw. Symbolen heißt übrigens fachgerecht Generieren einer Symboltabelle).

Aber, werden Sie einwenden, wann trage ich denn die echte Adresse von WAIT nach dem JSR-Befehl ein? Sie ahnen es sicherlich schon: Sie müssen ein zweites Mal durch. Wenn Sie dann auf den Befehl JSR WAIT stoßen, sehen Sie in der Symboltabelle nach, ob es schon eine Zuordnung zu WAIT gibt. Wenn ja, wird die entsprechende Adresse eingesetzt. Wenn nicht, bekommen Sie – jetzt wieder in der Rolle des Programmierers – eine Fehlermeldung.

Das erklärt wohl auch den Begriff Zwei-Pass-Assembler : damit ist nichts anderes als das eben beschriebene zweimalige Bearbeiten des Programm-Textes gemeint, so daß erst im zweiten Durchgang (Pass) Maschinen-Code erzeugt wird.

Die oben erwähnte optionale Speicher- oder Peripherie-Orientierung hat zu tun mit dem eingebauten Editor. Der Editor ist eigentlich ein eigenes Programm, nämlich eine zeilenorientierte Textverarbeitung. Ihr aus den oben erwähnten Mnemonics, Zuweisungen und (hoffentlich!) Kommentaren bestehender Source-Code (wörtlich übersetzt: Quell-Code, also der vom Assembler zu verarbeitende Text) wird mit diesem Editor erstellt. Näheres zum Editor siehe unten, hier soll es nur um die Beziehung zwischen Editor und Assembler gehen.

Macros und Include-Files

Im Prinzip erwartet der Assembler den Text im Editor. Aber eben nur im Prinzip. Es gibt nämlich zwei Spezialfälle:

Erstens: Macros. Macros sind, theoretisch ausgedrückt, Text-Ersatz-Anweisungen. Ihr praktischer Nutzen besteht darin, ständig wiederkehrende Befehls-Folgen nicht immer neu eintippen zu müssen. Dazu wird einmal ein Macro definiert; diese Definition besteht aus

- dem Namen des Macros
- einem Befehl, der angibt Jetzt wird ein Macro definiert (bei ASM/ED 64 m oder mf)
- gegebebenfalls der Anzahl der Parameter, die diesem Macro übergeben werden sollen
- der diesem Macro zuzuordnenden Befehlsfolge
- und, natürlich, einem Ende-Kennzeichen.

Findet der Assembler nach dieser Macro-Definition im Text den Namen dieses Macros, so tut er so, als ob der vollständige zu diesem Macro gehörende Text dort stände (und setzt gegebenenfalls die Parameter ein). Wie das im Einzelnen aussieht, ist weiter unten beschrieben, zum Prinzip hier noch eins: Macros machen zwar den Source-Code kürzer und lesbarer, es wird aber jedesmal der vollständige Maschinen-Code erzeugt. Macros sind also keinesfalls ein Ersatz für Unterprogramme!

Zweitens: Include-Files. Include-Files sind Source-Texte, die während des Assemblierens von einem Datenträger Zeile für Zeile eingelesen werden. Der entsprechende Maschinen-Code wird dort im erzeugten Programm eingefügt, wo das Include-File aufgerufen wurde. So können häufig benutzte Unterprogramme oder System-Definitionen bequem eingebaut werden. Außerdem ist dadurch die Länge von Source-Texten prinzipiell unbegrenzt.

Damit hätten wir den einen Fall von optionaler Speicher- oder Peripherie-Orientierung , nämlich die Frage: woher kriegt der Assembler seinen Text? Da drängt sich natürlich eine andere Frage geradezu auf: Wohin soll der Assembler mit dem erzeugten Code? Beim ASM/ED 64 gibt es drei Möglichkeiten. Erstens: der Code wird direkt in den Speicher geschrieben. Das ist praktisch zum Austesten, geht aber nur, wenn der betreffende Speicherbereich nicht belegt ist. In den Assembler oder in den Textspeicher zu assemblieren, führt natürlich zu bösen Fehlern . . . Zweitens: Der Code wird auf Diskette geschrieben. Geht im Prinzip immer. Drittens: es wird kein Code erzeugt. Dies kann zum Test auf syntaktische Fehler im Source-Code sinnvoll sein, oder wenn sich Drucker und Floppy nicht vertragen.

Das soll an Theorie erst einmal reichen, jetzt wirds konkret:

Die Bedienung

INPUT-ASS wird vom eigenen Datenträger (Sie müssen ihn natürlich aus dem Magazin heraus auf Ihre eigene Kassette/Diskette abspeichern) mit LOAD INPUT-ASS ,8,0 oder LOAD INPUT-ASS ,1,0 (SuperTape: LOAD INPUT-ASS ,7,0) geladen und mit RUN gestartet.

Editor

Der Editor dürfte denjenigen, die schon mit WordStar oder TURBO-Pascal gearbeitet haben, irgendwie bekannt vorkommen. Die Befehle sind tatsächlich weitgehend WordStar-kompatibel. Am einfachsten und schnellsten können Sie die Befehle kennenlernen, indem Sie alle ausprobieren. Bildschirmaufteilung: Die oberste Zeile des Bildschirms ist die Statuszeile. Ein inverses Zeichen am linken Rand zeigt an, daß ein aus zwei Zeichen bestehender Befehl eingegeben wird. Die Hexzahl daneben gibt die Grösse des noch freien Speichers an. Der Rest der Statuszeile wird für Eingaben, Statusmeldungen der Floppy und Fehlermeldungen des Assemblers verwendet. Der Text wird in den restlichen 24 Zeilen des Bildschirms dargestellt. Die Zeilen sind 80 Zeichen lang, durch horizontales Scrolling werden jeweils 40 von 80 Zeichen pro Zeile dargestellt. Der Cursor wird invers angezeigt.

Texteingabe: Die maximale Zeilenlänge beträgt 79 Zeichen, danach werden keine Eingaben mehr angenommen. Die Zeilen werden durch RETURN abgeschlossen. Bei der Eingabe gibt es zwei Möglichkeiten (durch ein Flag in der linken oberen Ecke angezeigt): Im Insert-Modus (i) werden die eingegebenen Zeichen automatisch eingefügt, im Overwrite-Modus (o) werden die unter dem Cursor liegenden Zeichen überschrieben.

Alle Befehle, die Peripherie-Operationen betreffen (Laden, Speichern, Drucken), sind blockorientiert. Das heißt, das immer der aktuell markierte, invers dargestellte Block gedruckt oder abgespeichert wird. Dies kann natürlich auch der gesamte Text sein. Diese Block-Orientierung erlaubt ein beliebiges Zusammenbauen und Auseinandernehmen des Programm-Textes. Haben Sie zum Beispiel eine Eingabe-Routine geschrieben, die als Unterprogramm für weitere Anwendungen geeignet ist – den entsprechenden Teil als Block markieren und abspeichern. Dieser Programmteil kann dann in andere Source-Texte wieder eingefügt werden. Oder Sie wollen nur bestimmte Teile ausdrucken – als Block markieren und über CTRL kw (siehe unten) auf Geräteadresse 4 schreiben. ASM/ED unterstützt also die Möglichkeit, sich eine Programm-Bibliothek zusammenzustellen – Sie müssen dann nur noch entsprechend überlegt programmieren.

Sämtliche Befehle werden über die Eingabe von CONTROL und gleichzeitig irgendeines anderen Zeichens erreicht. CONTROL meint dabei die CTRL-Taste, CTRL v bedeutet beispielsweise, die CTRL-Taste gedrückt zu halten und gleichzeitig (!) ein kleines v einzugeben. Mit Cursorzeile bzw. Cursorposition ist immer die Zeile/Position gemeint, in der sich der Cursor gerade befindet.

Bei allen Befehlen, die aus zwei Zeichen bestehen, kann das zweite Zeichen mit oder ohne Control eingegeben werden.

CTRL v (Control-v; Insert-Taste geht auch) schaltet zwischen Insert- und Overwrite-Modus um.

CTRL p erlaubt die Eingabe von Kontrollzeichen in den Text, die sonst als Befehle interpretiert würden. Kontrollzeichen werden invers dargestellt. Beispiel:
CTRL p CTRL c gibt CTRL c in den Text ein.

Cursor-Bewegungen

Die Tasten zur Bewegung des Cursors sind als Kreuz angeordnet (Natürlich können auch weiterhin die Cursortasten benutzt werden!):

```
Scroll W E R  Seite
       A S D F
       Z X C
```

CTRL s Cursor links
CTRL d Cursor rechts
CTRL a Wort links
CTRL f Wort rechts
CTRL e Cursor rauf
CTRL x Cursor runter
CTRL w Scroll up
CTRL z Scroll down
CTRL r Seite rauf
CTRL c Seite runter
CTRL i Tabulator; bewegt den Cursor in die Spalte des nächsten Wortanfangs der vorigen Zeile. Dies funktioniert natürlich nicht, wenn die Zeile über dem Cursor eine Leerzeile ist. (Klingt komplizierter als es ist, ausprobieren!)
CTRL qs Zeilenanfang
CTRL qd Zeilenende
CTRL qr Textanfang
CTRL qc Textende
CTRL qb Blockbeginn
CTRL qk Blockende
(Zum Begriff Block siehe unten)

Einfügen und Löschen:

DEL Zeichen links vom Cursor löschen. Falls der Cursor am linken Rand steht, wird die Cursorzeile mit der vorigen Zeile verknüpft

CTRL g Zeichen unter dem Cursor löschen.

CTRL qy Von der Cursorposition bis zum Zeilenende löschen.

CTRL y Cursorzeile löschen. Vorsicht: Das Auto-Repeat kann hier verheerende Auswirkungen haben!

CTRL ql macht alle Änderungen in der Cursorzeile rückgängig, sofern die Zeile noch nicht verlassen wurde. Vorsicht: CTRL y wird dadurch nicht rückgängig gemacht!

Block-Befehle:

Ein Block wird invers angezeigt, die Cursorzeile aber immer normal. Es gibt immer nur einen aktuellen Block.

CTRL ka gesamten Text als Block markieren. Empfehlenswert vor dem Abspeichern!

CTRL kb Cursorzeile wird als Blockanfang markiert.

CTRL kk Cursorzeile wird als Blockende markiert, das heißt, als erste Zeile, die nicht zum Block gehört.

Falls die Markierungen zu einem Widerspruch führen (Blockende vor Blockanfang), wird der Blockanfang auf den Textanfang bzw. das Blockende auf das Textende gelegt.

CTRL kc kopiert den markierten Block an die Cursorposition, als Block ist danach die Kopie markiert; der zuvor als Block markierte Text bleibt unverändert.

CTRL kv verschiebt den Block an die Cursorposition; der zuvor als Block markierte Text wird gelöscht. Sehr grosse Blöcke müssen in Portionen verschoben werden, da CTRL kv zuerst den Block kopiert und dann das Original löscht.

CTRL ky löscht den Block ENDGUELTIG.

Eingaben über die Statuszeile:

Wenn die Eingabe eines Filenamens oder eines Suchwortes erwartet wird, erscheint eine Frage (zB. load:) und der Cursor. In der Statuszeile kann nur mit Delete editiert werden. Die Eingabe wird mit Return abgeschlossen und mit STOP abgebrochen.

Filenamen werden in einem speziellen Format eingegeben:

load:82name

dabei ist 8 die Gerätenummer und 2 die Sekundäradresse (diese Zahlenangaben sind hexadezimal einzugeben, Gerätenummer 8 und Sekundäradresse 15 ist also 8f!) Der Name wird ohne Anführungsstriche eingegeben. Beispiele:

40:Printer (kein Filename !)
10name: File name von Kassette lesen
11name: File name auf Kassette schreiben
70name: File name mit Supertape lesen
71name: File name mit Supertape schreiben

Mit CTRL d kann nach der Eingabe diese als Default (festgelegte Vorgabe) gespeichert wer-

den. Dieser Default erscheint dann jedesmal, wenn diese Eingabe gemacht werden soll.

File-Befehle:

CTRL ke Diskbefehl senden. disk:8fs:text löscht das File text auf der Diskette (8f für 8,15 !).

CTRL kf Directory einlesen. Die Ausgabe kann mit Space angehalten und mit Commodore wieder fortgesetzt werden. Am Schluss muß eine Taste gedrückt werden, um wieder in den Editor zu kommen.

CTRL kr File einlesen. load:82asm liest das File asm von der Diskette ein. Der Text wird an der Cursorposition eingefügt und als Block markiert.

CTRL kw Block als File schreiben. save:11asm speichert den markierten Block unter dem Namen asm auf Kassette. (Wenn es verdächtig schnell ging, haben Sie wahrscheinlich vergessen, den Block zu markieren und die Warnung NO BLOCK übersehen. . .

CTRL k* schreibt den Editor mitsamt Text und Defaults auf die Diskette (Geht nur mit Diskette, da ein Datenfile geschrieben wird!). Mit diesem Befehl kann ohne Kopierprogramm eine angepasste Version von ASM/ED auf die Arbeitsdiskette geschrieben werden.
Kassette/Supertape: Programm verlassen und mit SAVE speichern!

Nach den Disketten-Operationen wird in der Statuszeile der Diskstatus angezeigt.

Fileformat:

Der Editor speichert die Texte als SEQ-Files ohne jegliche Zusätze, das heißt, die Zeilen werden durch CR (Carriage Return, entspricht CHR$(13)) begrenzt. CHR$(0) ist verboten, da der Editor das als Endmarke verwendet. Zeilen sollten – CR inbegriffen – nicht länger als 80 Zeichen sein. Texte von Assemblern, die den BASIC-Editor verwenden (herzliches Beileid. . .), können so lesbar gemacht werden:

```
load "source",8
open 1,8,2,"dest,s,w"
cmd 1:list
print #1:close 1
```

Ein vorheriges POKE22,35 erzeugt übrigens ein Listing ohne Zeilennummern.

Suchen und Ersetzen

Die Suchwörter bzw. Ersatzwörter werden in der Statuszeile eingegeben.

CTRL j Labeldefinition suchen, zum Beispiel setzt jump::label den Cursor auf die Labeldefinition :label. Dieser Befehl durchsucht den ganzen Text, also nicht nur vorwärts wie CTRL qf/CTRL qa.

CTRL qf-beliebige Zeichenfolge finden. Dabei wird ab Cursorposition der Text bis zum Ende durchsucht. Der Cursor bleibt jeweils am Ende des gefundenen Wortes stehen. Sollen auch Zeichenfolge vor der aktuellen Cursor-Position gefunden werden, empfielt sich ein vorheriges CTRL qr!

CTRL qa Suchen und Ersetzen.
find:.asc
chng:b
ersetzt .asc durch b. Bei jeder Fundstelle fragt der Computer:
replace (y/n/*)?
y = ersetzen, n = nicht ersetzen, * = automatisch ersetzen STOP = Abbruch. Durch * werden also sämtliche weiteren Suchworte durch das Ersatzwort ersetzt, ohne nochmalige Abfrage.

CTRL l wiederholt die letzte Operation, sucht also die nächste Fundstelle oder fährt mit dem Suchen und Ersetzen fort.

Verlassen des Editors

CTRL kq Editor verlassen. Mit RUN kann man wieder in den Editor zurückkehren, ohne den Text zu verlieren.

Assembler

Sämtliche Assembler-Befehle werden wie die Editor-Kommandos über CTRL und eine weitere Taste eingegeben. Eventuell erforderliche weitere Eingaben werden in der Statuszeile erwartet.

Aufruf

CTRL kx Text assemblieren. Dabei wird abgefragt:

code

Dadurch wird festgelegt, wohin der erzeugte Maschinen-Code geschrieben werden soll; ob in den Speicher, auf Diskette oder nirgendwohin.

Beispiele:

code:82dest: Objektcode unter dem Namen "dest" auf Diskette schreiben.
code:RETURN: keine Angabe erzeugt keinen Code.
code:*: Objektcode wird in den Speicher an die Adresse des ersten org-Befehls geschrieben. Vorsicht, nicht in den Text-Speicher oder den Assembler schreiben! (siehe unten: Speicherbelegung)

list

Ausgabegerät des Listings. Beispiele:
list:44Formatiertes Listing auf den Drucker
list:00dito auf Bildschirm (dann bei symb:03)
list:RETURNkein Listing ausgeben

symb

Ausgabegerät der Symboltabelle. Beispiele:
symb:83sym: Symboltabelle unter dem Namen "sym" auf Diskette schreiben.
symb:03: Symboltabelle auf dem Bildschirm ausgeben. Anhalten / Ende wie CTRL kf!
symb:RETURN: nicht ausgeben

CTRL kd Probe-Assemblierung ohne Ausgaben; sinnvoll für Syntax-Checks.

Label

:label1

Label dürfen beliebig lang sein und aus Buchstaben und Zahlen bestehen. Das erste Zeichen muß ein Buchstabe sein. Vor jeder Labeldefinition muss ein Doppelpunkt stehen. Labeldefinitionen dürfen auch allein in einer Zeile stehen. Mnemonics und Pseudo-OP's können nicht als Label definiert werden. Zwischen Label und in derselben Zeile folgendem Text muß ein Leerzeichen stehen.

Mnemonics

Entsprechend der normalen 6502-Notation. Ausnahmen:

Adressierungsart Accumulator: ASL statt ASL A.
Adressierungsart indirekt X-indiziert: (AD),X statt (AD,X).

Der Quelltext muß kleingeschrieben sein.

Assemblerbefehle (Pseudo-OP's)

Diese Befehle werden in den Programm-Text geschrieben und dienen als Anweisungen für den Assembler.

b (Byte-Befehl)

Die Zahlen bzw. der Text werden als Tabelle assembliert. ASCII-Zeichen werden durch ein vorangestelltes Apostroph gekennzeichnet, fortlaufende Texte können durch Anführungszeichen eingeschlossen werden. Die einzelnen Byte-Anweisungen werden durch Kommata getrennt. Entspricht .BYT, .DC.B,.ASC bei anderen Assemblern. Beispiele:

```
b 'a,'b,'c,12,10,13,0
b "input 64",13,0
:schwarz = 0
b schwarz
```

w (Word-Befehl)

Assembliert 16-Bit-Worte. Die Worte werden in der 6502-üblichen Reihenfolge gespeichert (erst Low-, dann High-Byte).
Beispiel 1:

```
w $ffea
```

erzeugt die Byte-Kombination

$EA, $FF

Beispiel 2:

```
org $c000
  :start lda #00
  :end rts
  . . .
  :tabel w start, end
```

erzeugt an der durch Label tabel gekennzeichneten Adresse die Byte-Kombination

$00, $C0, $02, $C0

s num

assembliert num 0-Bytes in den Text. Zum Freihalten von Datenbereichen geeignet. Nachteil: Diese 0-Bytes müssen dann jeweils auch von Diskette/Kassette geladen werden. Beispiel:

```
org $c000
  :init jmp begin
  :buff s $100
  :begin lda #00
```

erzeugt einen Buffer ab Adresse $C003 ($C000 plus Befehlslänge des JMP-Befehls), der $100 Bytes lang ist. Label begin entspricht damit der Adresse $C103.

org adres

setzt den Programmzähler des Assemblers auf adres. Die org-Anweisung muß zu Anfang des Programms stehen, wird dies vergessen, kommt es zu "unerklärlichen" branch-errors. Der Programmzähler kann mehrmals neu gesetzt werden, der Code wird auf jeden Fall an die Adresse des ersten Programmzählers geschrieben! Alle weiteren org-Definitionen beeinflussen nur den logischen, nicht den physikalischen Stand des Programmzählers. Um auch den realen Stand des Programmzählers zu beeinflussen (natürlich nur vorwärts), wird der s-Befehl benutzt:

s $3000-* setzt den Programmzähler logisch und physikalisch auf $3000)

Nächstes Beispiel:

```
org $8000
:trans      ldx #255
:trlop      lda code,x
            sta start,x
            dex
            bne trlop
            jmp start
:code = *
org $C000
:start      jmp start+$100
...
```

Die Routine trans (ab Adresse $8000) verschiebt den Programmteil, der real ab Label code liegt, zur Adresse $C000. Label start entspricht somit der Adresse $C000, der nach start folgende Programmteil wird so assembliert, als befände er sich ab Adresse $C000, dementsprechend ist JMP START ein Sprung zur Adresse $C000. Die Adresszuweisungen im erzeugten Code sehen dann so aus:

```
$8000 LDX #$FF
$8002 LDA $800B,X
$8005 STA $C000,X
$8008 DEX
$8009 BNE $8002
$800B JMP $C000
$800E JMP $C100
...
```

=

:printc = $ffd2

setzt das Label printc = $ffd2. Entspricht EQU. Das Label wird nur im 1. Durchlauf gesetzt. Eine erneute Zuweisung erzeugt einen dd-Error.

:=

:pass: = 1

erlaubt die (eventuell mehrfache) Neudefinition eines Labels, das Beispiel setzt das Label pass = 1. Falls es dieses Label schon gibt, wird es geändert.

pag

bewirkt einen Seitenvorschub im Listing.

lis n

setzt den Listing-Mode.
lis 0: nicht ausgeben
lis 4: ausgeben
lis 5: ausgeben, Makros voll ausgeben
lis 6: ausgeben, nicht erfüllten If-Teil ausgeben
lis 7: alles ausgeben

Der Listing-Mode kann innerhalb eines Programm-Textes beliebig oft gewechselt werden.

tit text

text wird als Titel am Anfang jeder Seite ausgedruckt. Der Titel kann mehrmals im Programm umdefiniert werden. In den Titel können auch Steuerzeichen für den Drucker eingestreut werden.

Ausdrücke

Der Assembler verarbeitet folgende Zahlentypen:

dezimal	ist Vorgabe (Default)
hexadezimal	Vorzeichen $.
binär	Vorzeichen %.
ASCII	Vorzeichen '.
Programmzähler-Stand	Zeichen *.

Beispiel: $ffff — Beispiel: 'a
Beispiel: %010101111 — Beispiel: :lb = *

Die Zahlensysteme können beliebig vermischt werden. Beispiel:

:tabel = $1000+12

weist tabel den Wert $100B zu.

Rechenoperationen

Der Assembler arbeitet mit vorzeichenlosen 16-Bit-Zahlen.

+ − * /	: Grundrechenarten
&	: logisches AND
!	: logisches OR
€	: logisches EOR
= ‹ ›	: Vergleiche. $ffff = wahr, 0 = falsch.

Die Ausdrücke werden von links nach rechts ausgewertet. Klammern sind nicht erlaubt.

Außerdem gibt es noch die Operatoren HIGH und LOW
(Beeinflussen den ganzen Ausdruck); Beispiel:

```
:video = 1024
ldy #>name+10 ;High-Byte
ldx #<name+10 ;Low-Byte
```

entspricht

```
ldy #04
ldx #10
```

Kommentare

;Kommentare gehen bis zum Zeilenende
; gibt im Listing einen Strich !

Makros

Mit Makros kann man eigene Befehle konstruieren. Die Syntax ist 1.) Name des Macros, 2.) Macro-Befehl m 3.) eventuelle Parameter 4.) Befehle dieses Macros 5.) Endekennzeichen. Das Ende der Makrodefinition wird durch / markiert (muß alleine in der Zeile stehen !). Beispiel:

```
:message m 1 ;makro mit 1 parameter
ldy #>@0     ; @0 = 1.parameter
lda #<@0
jsr $ab1e
/
```

Das Makro wird so aufgerufen:

message usertxt

und entspricht dann folgendem Text:

```
ldy #>usertxt
lda #<usertxt
jsr $ab1e
```

usertxt muß natürlich vorher definiert sein.

Ein Makro kann bis zu 10 Parameter haben, beim Aufruf muß diese Anzahl eingehalten werden. Beim Aufruf können beliebige Ausdrücke übergeben werden. Makros können mehrmals aufgerufen und verschachtelt werden. Ein Makro kann erst aufgerufen werden, nachdem es definiert wurde. Probleme kann es mit Labels geben:

- Alle Label sind global, Bezeichner wie loop provozieren double-definition-errors. Bewährt haben sich Bezeichnungen, bei denen die ersten 2 bis 3 Zeichen angeben, zu welcher Prozedur sie gehören.
- Innerhalb von Makros führen doppelte Definitionen nicht zu einer Fehlermeldung.
- Makros mit Vorwärtsreferenzen müssen statt durch m durch mf definiert werden:

```
:com mf 2
ldx @0+1
cpx @1+1
bne com2
lda @0
cmp @1
:com2
/
```

Bedingte Assemblierung

```
:debug = 1
if debug
jsr test
el
jsr normal
ei
```

Wenn die Bedingung erfüllt ist (ungleich 0), wird der folgende Code bis el (else) assembliert, wenn nicht, wird der Code hinter el bzw. ei (endif) assembliert. Im Beispiel wird also JSR TEST assembliert, aber nicht JSR NORMAL. if, el und ei müssen am Anfang der Zeile stehen ! Die Bedingungen können auch verschachtelt werden.

Include

Mit Include können Source-Module von der Diskette assembliert werden.

in 84infile

Der Filename wird im selben Format wie im Editor angegeben und geht bis zum Ende der Zeile. Include kann nicht verschachtelt werden; in einem Include-File können keine Makros definiert werden.

phc (Phase Error Control)

Der Pseudo-Befehl phc schaltet die Kontrolle auf Phase-Errors ein. Beispiel:

```
org $c000
phc
lda test
:test = 0
:end
```

Im ersten Durchlauf wird lda als 3-Byte-Befehl assembliert, da die Variable test noch nicht bekannt ist. Im 2. Durchlauf macht der Assembler daraus einen 2-Byte-Befehl. Zero-Page-Variablen müssen also vor der Benutzung definiert

werden! Mit Hilfe der bedingten Assemblierung kann man übrigens auch sehr schöne Phase-Errors zusammenbasteln. . .

Fehlermeldungen

Fehlermeldungen werden in der Statuszeile angezeigt, der Cursor wird im Text hinter den Fehler positioniert. Falls ein Fehler in einem Makro oder in einem Include-File auftritt, wird zuerst der Makroaufruf bzw. die fehlerhafte Zeile im Include-File angezeigt, nach Drücken einer beliebigen Taste erscheint die Fehlermeldung, der Cursor ist jetzt auf die fehlerhafte Zeile im Makro bzw. auf den Aufruf des Include-Files gesetzt.

am:illegale Adressierungs-Art (adress-mode error); Beispiel: STA #8000
Die Adressierungsart immediate kann nicht Ziel eines Befehls sein.

br:zu weiter Branch (branch error);
Beispiel: bedingter Sprung weiter als 127 Bytes vorwärts oder -128 Bytes rückwärts; oder erste org-Anweisung fehlt.

dd:Label doppelt definiert (double definition error);
Beispiel:
```
:print = $able
org $c000
:print ldx #00
```
Einem Symbol wurde erst (durch =) ein Wert zugewiesen, und das gleiche Symbol wird später als Label im Programm verwandt.

md:falsche Makrodefinition (macro definition error); Beispiel:
Es wurde nicht als erstes der Name des Macros, sondern die Macro-Anweisung selbst (m) gegeben.

no:kein Fehler !

op:illegaler Opcode (opcode error); Beispiel: Leerzeichen zwischen Befehl und Operand fehlt

ov:Zahl zu groß (overflow error); Beispiel:
STA 80000
Der Operand ist größer als 65535.

ph:Phase error; im zweiten Pass ergibt sich für eine Adress-Berechnung ein anderer Wert als im ersten Pass – selten auftretender Fehler.

sn:Syntax error; Beispiel:
.w oder .b statt w beziehungsweise b

sv:Symboltabelle zu groß (symbol table overflow); die Symbol-Tabelle paßt nicht mehr in den Speicher. Lösung: Text in zwei Teilen assemblieren.

tm:falsches Symbol; Beispiel:
Das erste Parameter in einem Macro wird über 1 statt über 0 angesprochen.

ud:Label nicht definiert (undefined label); es wurde ein Label angesprochen, das im Programm nicht existiert.

Speicherbelegung

$0801 – $2541 Assembler/Editor
$2541 – ende Textspeicher, Endadresse in Statuszeile, ende kann maximal auf $B6FF stehen
$B700 – $BFFF div. Puffer, Anfang Symbol-Tabelle
$C000 – $CFFF nicht benutzt
$D000 – $DFFF I/O-Bereich
$E000 – $FFFF Fortsetzung Symboltabelle

Nach einem Reset kann INPUT-ASS durch SYS2061 wieder aufgerufen werden, in der Regel ohne Textverlust.

Source-Diskette

Als besonderen Service haben wir Ihnen eine Source-Diskette/Kassette zusammengestellt. Diese enthält folgende Source-Files
Assembler
Editor
Monitor MLM/64
Library-Module
Konvertierungen

Um welchen Assembler und Editor es dabei geht, dürfte klar sein. Der Monitor MLM/64 wurde in Ausgabe 3/85 unseres Magazins veröffentlicht. Er beinhaltet – neben den gängigen Funktionen wie HUNT, FILL, DUMP und so weiter – einen Diss-Assembler und einen Direct-Line-Assembler. Die Library-Module bestehen aus Standard-Routinen (HEX ASCII-DEZIMAL-Wandlung und ähnlichem), spezifischen C64-I/O-Routinen (LOAD/SAVE) und anderen nützlichen Unterprogrammen. Die Konvertierungen wandeln Source-Texte im Format PROFI-ASS und MAE in das File-Format des INPUT-ASS. Dies ist der Stand bei Redaktionsschluß dieser Ausgabe, der Inhalt der Source-Diskette kann sich eventuell noch erweitern.

Die Kassette ist im SuperTape-Format bespielt, um die entsprechenden Programme auch für andere 6502-Rechner nutzbar zu machen.

Preis: 49.– DM; Bestelladresse: siehe Impressum

ASM-Befehle Kurzübersicht

Alle Befehle sind zu verstehen als CTRL und gleichzeitig die aggeführten Tastenkombinationen. Bei Befehlen, die aus zwei Buchstaben bestehen, kann der zweite Buchstabe auch ohne CTRL eingegeben werden.

v Insert-Modus ein/aus
p Control-Zeichen Eingabe

Cursor-Bewegung

w Scroll up
e Zeile hoch
r Seite rauf
a Wort links
s Zeichen links
d Zeichen rechts
f Wort rechts
z Scroll down
x Zeile runter
c Seite runter
i Tabulator-Sprung
qs Zeilenanfang
qd Zeilenende
qr Textanfang
qc Textende
qb Blockbeginn
qk Blockende

Einfügen und Löschen

g Zeichen unter Cursor löschen
qy Zeichen rechts ab Cursor löschen
y Zeile löschen
ql UNDO

Block-Handling

ka ganzen Text als Block markieren
kb Blockanfang markieren
kk Blockende markieren
kc Block kopieren
kv Block verschieben
ky Block löschen

Statuszeile

d Aktuelle Eingabe wird Default

File-/Peripherie-Befehle

ke Disk-Befehl senden
kf Directory
kr Block laden
kw Block schreiben
kq Editor verlassen

Suchen und Ersetzen

j Sprung zum Label
qf Zeichenfolge suchen
qa Zeichenfolge austauschen
l letzten Befehl wiederholen

Assembler

kx Assembler-Aufruf
kd Probe-Assemblierung

Schlaumeier

Das Programm Schlaumeier ist aus der Beschäftigung mit 'künstlicher Intelligenz' entstanden und soll Sie spielerisch in das Teilgebiet Expertensysteme entführen.

Was sind denn Expertensysteme?

Expertensysteme oder auch 'wissensbasierte Systeme' sind Programme, die dem Anwender bei der Lösung eines Problems helfen. Nun tun das ja eigentlich alle Programme, oder sie sollten es wenigstens. . .
Der entscheidende Unterschied zu herkömmlicher Software ist, daß ein Expertensystem quasi intelligentes Verhalten an den Tag legt. Ein solches Programm ist sozusagen ein Experte auf seinem Spezialgebiet, das heißt, es verfügt über Spezialwissen und ist in der Lage, anhand von diesem die Lösung einzukreisen und anzugeben.

Sehr vielfältig ist das Angebot solcher Programme auf dem Markt noch nicht, aber der Trend geht eindeutig in die Richtung. In den USA gibt es bereits eine Handvoll Expertensysteme, die vielfältig kommerziell im Einsatz sind. Darunter sind so schillernde Namen wie DENDRAL, ein Programm, das aus einer chemischen Summenformel auf die Stukturformel organischer Moleküle schließen kann. (Wer schon einmal Chemie gebüffelt hat, wird das zu schätzen wissen. . .) Dieses Programm ist bereits Ende der sechziger Jahre entwickelt worden und gilt mithin als das erste Expertensystem. Sie sehen, ganz neu ist die Idee nicht mehr. Ein anderer Vertreter ist MYCIN, ein Programm, das bei der Diagnose von bakteriellen Blutinfektionen behilflich ist. Etwas neuer ist da ein Objekt der Nixdorf-KI-Abteilung: Ein Computer Reparatursystem namens Faultfinder. Dieses System erarbeitet im Dialog mit dem Benutzer eine Vermutung über die Ursache der Störung eines Computers. Der Dialog ist überhaupt ein zentraler Punkt der Benutzeroberfläche eines Expertensystems.

Wie sieht denn die Arbeit mit so einem System nun aus?

Ein Expertensystem kennt eine Menge von Fakten (diese nennt man Objekte) und eine Reihe von Regeln (auch Kriterien genannt), mit denen sich die Fakten verknüpfen lassen. Treten Sie nun mit einem Problem an das System heran, so stellt es aufgrund dieser Regeln gezielte Fragen und versucht so, Objekte auszuschließen. Nehmen wir an, wir haben einen Pflanzenbestimmungsexperten vor uns. Das System kennt dann viele Pflanzen(die Objekte) und einige Regeln, die diese Pflanzen in Gruppen aufteilen. Sie haben nun beim Spazierengehen eine Ihnen unbekannte Pflanze gefunden und wüßten gern, was für ein Exemplar Sie vor sich haben. Dann beginnt jetzt ein Frage-und-Antwort Spiel:

System:Ist es eine kleine Pflanze?
Sie: Ja.
(damit sind bereits alle Bäume und ähnliche Pflanzen ausgeschieden)
System:Hat die Pflanze eine Blüte?
Sie: Nein.
(Also ist es wohl keine Orchidee. . .)
System: Sticht es ?
Sie: Ja.
System: Mit großer Wahrscheinlichkeit ist es eine Distel!

Naja, eine Distel hätten Sie wohl auch so erkannt. Dieses einfache Beispiel zeigt aber schon den großen Vorteil eines Expertensystems gegenüber einer normalen Datenbank: Das gezielte Stellen von Fragen macht die Lösung oft erst möglich. Einer Datenbank geben Sie die Ihnen bekannten Kriterien an und es werden alle Datensätze ausgegeben, die zu diesen Informationen passen. Das Verknüpfen von einzelnen Informationen nach Regeln unterbleibt aber, und damit gehen wichtige Informationen 'verloren'.

Dieser Art von intelligenten Informationssystemen steht sicherlich eine große Zukunft ins Haus; nicht zuletzt wegen ihrer Benutzerfreundlichkeit. Die dialoggeführte 'Erarbeitung' einer Lösung ist einfach angenehmer zu handhaben, insbesondere auch für Nicht-EDV-Spezialisten.

Was können Sie nun mit Schlaumeier anfangen?

Schlaumeier ist sicher kein Expertensystem wie die vorhin beschriebenen. Es ist ein Programm zum experimentieren; deswegen ist es einfach gehalten und in Basic geschrieben. Schlaumeier hat noch einen weiteren Vorteil: Es kann (oder muß) neue Objekte hinzulernen. Wenn Sie das Programm zum ersten mal starten, ist es noch völlig dumm. Hier haben Sie nun die Möglichkeit, das System selbst mit Wissen zu füttern, zu einem Thema, das Sie selbst auswählen können. Das gesammelte Wissen wird abgespeichert und

beim nächsten Mal wieder geladen. So wird Ihr System von Mal zu Mal kompetenter.

Beim Aufbau Ihrer Wissensbasis werden Sie merken, daß das eine Kunst für sich ist. Die Fragen müssen die Objekte treffend beschreiben, sonst hat man schnell den Eindruck, daß das System nur dumme Sachen fragt. Außerdem sollten Sie am Beginn mit sehr verschiedenen Objekten anfangen, die sich mit allgemeinen Fragen unterscheiden lassen. So erreichten Sie, daß das System auch geschickt fragt: von der groben Unterteilung am Anfang bis zur Einkreisung der Lösung.

Für alle, denen die Sache bisher zu trocken war, kommt zum Schluß noch eine Anregung: Schlaumeier eignet sich hervorragend für Ratespiele wie z.B. 'Ich sehe was, was Du nicht siehst oder Tiere raten. Letzteres ist ein Spiel, das gern von Studenten auf dem Universitätsrechner gespielt wird (sehr zum Ärger der ernsthaften Rechnerbenutzer. . .). Die Objekte bei diesem Spiel sind Tiere aller Art. Der Spieler muß sich ein Tier denken und dann die Fragen des Programms beantworten. Entweder kennt das Programm das Tier schon und errät es, dann ärgert der Spieler sich, oder das Tier ist neu, und das Programm muß sich geschlagen geben und die Lösung erfragen. Beim nächsten Mal kennt es das Tier natürlich. Der Reiz an der Sache liegt nun darin, daß viele Leute mit dem selben Programm spielen, so sammelt sich das Wissen von allen, und der einzelne ist bald nicht mehr in der Lage, das Programm zu schlagen. Wenn man das zum ersten Mal erlebt, steht man doch recht fassungslos vor dieser scheinbaren Intelligenzkiste.

In ähnlicher Weise kann man mit der ganzen Familie 'ich sehe was,was Du nicht siehst' spielen, wobei man sich bald wundert, wie genau das Programm bald alle Einzelheiten des Raumes kennt. (Dabei fehlt dann natürlich die umgedrehte Version: Der Computer denkt sich was und der Spieler muß raten. . .)

Wie wird Schlaumeier bedient?

Schlaumeier wird mit RUN aufgerufen. Über die Tasten F1-F8 können folgende Funktionen erreicht werden:

F1 – altes Wissen laden
F3 – neues Wissen speichern
F5 – Dialog führen
F8 – Programm-Ende

Zuerst müssen Sie das Thema angeben, mit dem Sie arbeiten wollen. Haben Sie bereits eine Datei auf Ihrem Datenträger, die mit '.wissen' endet, können Sie diese mit F1 laden. Ist die Datei unbekannt, ist Schlaumeier zu Anfang dumm. Sie können im Dialog mit ihm (F5) seinen Kenntnisstand nach und nach erhöhen. Natürlich geht das ebenso bei bereits vorhandenen Dateien:

Antworten Sie nach der Meldung 'Nach meinem Wissen tippe ich auf.. mit 'Nein', geht Schlaumeier in die Lernphase und erwartet von Ihnen eine neue Frage, auf die man mit Ja oder Nein antworten kann.

Mit F8 können Sie, nachdem Schlaumeier seinen Tip ausgegeben hat, das Programm beenden. Mit LIST könnnen Sie sich den BASIC-Teil des Programms ansehen und dort auch Änderungen vornehmen. Vergessen Sie aber nie, vor dem Abspeichern POKE 44,8:POKE 43,1 anzugeben, damit der Maschinenprogramm-Teil mit abgespeichert wird.

Rätselecke

Flott permutiert

Auflösung des Rätsels aus INPUT 64 3/86

So, das hätten wir! Ganz schön geschwitzt, bei der Auswertung der Rätsel-Auflösungen. Aber das wird Sie wenig interessieren, also zur Sache:

Die Lösung

Zur Erinnerung: es ging um die Permutation (das heißt, die Ausgabe sämtlicher Kombinations-Möglichkeiten der Reihenfolge) der sechstelligen Zahlenreihe mit den Ziffern 1 bis 6. Die einfachste Möglichkeit – für den etwas mathematisch bewanderten – war natürlich, nur die Anzahl der Kombinationen über die Fakultät der Anzahl der zu permutierenden Ziffern (zum Beispiel für 6 Ziffern: 6 * 5 * 4 * 3 * 2 * 1) zu errechnen. Dadurch wurden natürlich sehr kurze Laufzeiten erreicht. Wer sich aber die Aufgabenstellung und vor allem das von uns vorgegebene Pro-

Listing Rätselgewinner

```
10 rem *    permutation     *
20 rem *--------------------*
30 rem * ab zeile 100 steht *
40 rem *    ihr programm    *
50 rem * zur errechnung und *
60 rem * anzeige der moegl. *
65 rem * kombinationen      *
66 :
70 rem fori=1to6:av(i)=i:next
80 rem n=0: rem zaehler
90 ti$="000000"
100 :
110 i=0:r=0:s=0:t=0:x$="":dimf(54):n=0:e=54:q=0:p=0:o=0:a=49:y=309:z=0
130 print chr$(147);
140 fori=atoe:f(i)=0:next
190 :
200 foro=atoe
205 f(o)=1:printchr$(19);
210 forp=atoe:iff(p)then340
215 f(p)=1:z=y-o-p:z$="  "+chr$(o)+chr$(p)
220 forq=atoe:iff(q)then330
230 f(q)=1:x$=z$+chr$(q):fori=ato52:iff(i)thennext
250 r=i:i=e:next:fori=r+1to53:iff(i)thennext
270 s=i:i=e:next:t=z-q-r-s:n=n+6:f(q)=0
300 printx$chr$(r)chr$(s)chr$(t)x$chr$(r)chr$(t)chr$(s)x$chr$(s)chr$(r)c
hr$(t);
302 printx$chr$(s)chr$(t)chr$(r)x$chr$(t)chr$(r)chr$(s)x$chr$(t)chr$(s)c
hr$(r);
330 next:f(p)=0
340 next:f(o)=0:next:print
990 :
1000 print"9900 rem anzahl der moeglichk.: ";n
1010 print"9910 rem zeit: ";ti/60
9000 end
9100 :
9110 rem juergen rampmaier
9120 rem turnerstrasse 120
9130 rem 6900 heidelberg
9140 rem raetsel 3/86 von input 64
9150 rem schnelle, sehr unuebersichtliche version
9190 :
9900 rem anzahl der moeglichk.:  720
9910 rem zeit:  15.4833333
ready.
```

gramm-Gerüst genau angesehen hat, wird bemerkt haben, das auch die Ausgabe jeder einzelnen Kombination gefordert war.

Das schnellste Programm, das diese Bedingungen erfüllt, benötigt 15.48 Sekunden für die 720 möglichen Kombinationen. (Diese Lösung können Sie aus dem Modul "Rätselecke" mit CTRL + S auf Ihren eigenen Datenträger abspeichern.)

Der Autor, Jürgen Rampmaier, hat sich einige Überlegungen zur Geschwindigkeitssteigerung gemacht, nämlich
– daß die Ausgabe von Fließkomma-Variablen nicht gerade die schnellste ist. Zahlenwerte werden statt dessen als Zeichenketten ausgegeben.

– daß logische Verknüpfungen innerhalb von IF-Abfragen sehr langsam arbeiten (IF A = B OR A = C THEN. . ..). Schneller sind getrennte Abfragen:
IF A = B THEN. . .. und IF A = C THEN. . ..

Bleiben wir noch etwas beim IF. Eine IF-Abfrage ist ja immer ein Vergleich. Wenn dieses zutrifft, dann mache jenes und so weiter. Meist wird verglichen, ob A = B ist oder ob A kleiner als B ist. Fragt man aber nur, ob eine Variable ungleich Null ist, kann ein Teil der Frage entfallen: IF A THEN. . . Dabei wird nur festgestellt, ob eine Aussage wahr oder falsch ist. Sollte A gleich Eins sein, wird der Rest der Zeile ausgeführt, sonst die nächste Zeile. Das erspart dem BASIC-Interpreter erheblich Arbeit, und das Programm wird dementsprechend schneller.

– daß mathematischen Operationen zeitraubend sind. Denn auch Integer-Variablen (die mit dem % hinter dem Namen) werden vom C64 erst in Fließkomma-Zahlen gewandelt, damit wird gerechnet, und anschließend wird zurückgewandelt. Das dauert natürlich alles seine Zeit. Deswegen hat der Autor darauf fast völlig verzichtet, nur in Zeile 260 und 280 werden Variablen subtrahiert bzw. addiert.

– daß Tauschaktionen der Art A = B : B = C : C = A am besten ganz unterbleiben.

Bei den eingesandten Programmen wurden diese Überlegungen häufig nur teilweise berücksichtigt. So ergaben sich Laufzeiten von 15.48 bis 380.13 Sekunden, der "Trend" ging in Richtung der schnelleren Geschwindigkeiten.

Die Gewinner

Der 1. Preis – ein INPUT 64 Jahresabo – geht an Jürgen Rampmaier in Heidelberg

2. bis 11. Preis: Buchpreise (Modelle der Wirklichkeit) gehen an:

Arthur Wallner, Bottrop; Stefan Friedt, Niederwoerresbach; Jonny Schneider, Biebesheim; Jochen Rätsch, Geseke; Manfred Wendt, Kiel; Marco Sommerau, Waiblingen; Peter Schubert, Goslar; Jost Koller, Beckedorf; Ernst Windgassen, Wuppertal; Tilmann Fertig, Stutensee.

Allen Gewinnern herzlichen Glückwunsch und allen allen anderen Einsendern vielen Dank für Ihre rege Teilnahme.

und noch mehr. . .

Als Beigabe gibt es noch eine Alternativ-Lösungs zum Rätsel aus INPUT 64 Ausgabe 1/86. Auch dieses Programm können Sie mit CTRL + S auf Ihren eigenen Datenträger abspeichern.

ZS-Copy

Als Besitzer eines C-64 werden sie sicher wissen, daß man nicht nur auf den Standardzeichensatz des Rechners angewiesen ist. Der Videochip erlaubt die Definition eigener Zeichen. Sehr nützlich ist dieses z.B. für Deutsche Sonderzeichen (ß und Umlaute). Leider verfügt der C-64 nur über einen amerikanischen Zeichensatz, der diese Umlaute nicht enthält. So muß erst einmal ein neuer Zeichensatz definiert werden, der die Sonderzeichen enthält. Besonders interessant ist diese Möglichkeit für Videospiele, bei denen z.B. irgendwelche Felsen im Hintergrund aus einem geänderten Zeichensatz aufgebaut sind.

Einen großen Nachteil haben diese Änderungen dennoch, sie können nicht zu Papier gebracht werden, da normale Hardcopy-Programme nur den ASCII-Wert des Zeichens an den Drucker senden und nicht das Aussehen des Zeichens auf dem Bildschirm. Dieses können Sie jetzt mit ZS-Hardcopy. Dies Programm holt sich den Wert der Zeichen aus dem Bildschirmspeicher, sucht das entsprechende geänderte Aussehen aus dem Zeichensatz-Speicher, speichert sie in einem Puffer als HiRes-Bild und druckt dieses aus.
Da hier der Bildschirm als Grafik ausgedruckt wird, können auch nur Grafik-Drucker mit der

ZS-Hardcopy zusammenarbeiten, die dem MPS-801 entsprechen, da dieses Ausgabeformat fast zu einem C64-Standard geworden ist und auch Interfaces für den C64 zu Fremddruckern diesen Code meist verstehen.

Maschinen-Programmierer können die Routine an andere Drucker anpassen. Wir hatten bei verschiedenen Tests keine Anpassungsschwierigkeiten. Im Maschinen-Code, den man mit einem Monitor disassemblieren kann (MLM64 aus INPUT 64 3/85), finden Sie einen Aufruf für die Drucker-Initialisierung. Hier braucht nur eine Routine für einen anderen Drucker implementiert werden.

Nun zum Aufruf der ZS-Hardcopy. Sie steht ab $C800 und wird durch

sys 51200,adr,mode

aufgerufen. Dabei ist 'adr' die Adresse des Zeichensatzes (Dezimal). Wenn Sie adr mit 0 angeben, sucht sich das Programm die Adresse des aktuell eingeschalteten (!) Zeichensatzes selbst.

'mode' gibt die Ausgabeart an, es sind Zahlen von 0 bis 3 erlaubt. Die folgende Tabelle gibt ihre Bedeutungen an. Zeichensatz 1 bedeutet "normaler" Zeichensatz, Zeichensatz 2 "Extended Colour Mode", also verschiedene Hintergrundfarben.

0 = Zeichensatz 1 mit Zeilenvorschub
1 = Zeichensatz 1 ohne Zeilenvorschub
2 = Zeichensatz 2 mit Zeilenvorschub
3 = Zeichensatz 2 ohne Zeilenvorschub

Mathe mit Nico

Nico beschäftigt Sie und sich mit fundamentalen Grundlagen; der Basis der Zahlensysteme. Sie werden nach dieser Lektion wissen, daß die Differenz zwischen hexadezimal ABCD und oktal 125715 gleich Null ist.
Beachten Sie bitte, daß Sie auf der Ihnen bekannten Rechenseite nur im Dezimalsystem rechnen können. Buchstaben oder andere nicht-numerische Zeichen können Sie nur bei der Übergabe der Lösung eingeben. Dazu muß ein großes L unmittelbar vor der Lösung eingegeben werden. Zum Beispiel: L12AB. Hiermit übergeben Sie den Wert 12AB im erwarteten Zahlensystem.

(Bei diesem Beispiel dürfte es sich mindestens um das 12er-System handel n. . .) Die Rechenseite können Sie immer (mit R) aufrufen, wenn Sie zu einer Lösungseingabe aufgefordert werden.

Die versunkene Stadt

Sie befinden sich in einem Raumschiff mit begrenztem Energie- und Sauerstoffvorrat und sollen, nachdem Sie mindestens sieben Diamanten aufgelesen haben, die versunkene Stadt finden. Soweit so gut. Selbstverständlich hat der Autor einige Schwierigkeiten eingebaut, aber auch die Möglichkeit, unterwegs sowohl Energie als auch Sauerstoff zu tanken, oder sich einfach von hier nach dort verschaukeln zu lassen.
Die Bedienung ist denkbar einfach. Wenn Sie einen Joystick besitzen, stecken Sie diesen bitte in Port 2. Ansonsten ist die entsprechende Tastenbelegung:

aufwärts	=	A
abwärts	=	Z
links	=	.
rechts	=	/
Feuer	=	Leertaste

Energie können Sie an den mit 'F' bezeichneten Stellen tanken (von oben anfahren), und Ihren Sauerstoffvorrat frischen Sie auf, wenn Sie die blinkenden Sauerstoff-Flaschen berühren. Einige Sperren lösen sich in bestimmten Zeitabständen von selbst auf, bei den anderen befindet sich ein Türöffner vor und hinter der Sperre. Wenn Sie sich streßfrei in einen anderen Spielabschnitt befördern lassen wollen, sollten Sie (wie immer von oben) auf den großen Kasten fahren. Wenn Sie Pech haben, kommen Sie aber an der gleichen Stelle wieder heraus.
Haben Sie genügend Diamanten eingesammelt und Ihnen ist unterwegs nichts auf den Kopf gefallen, können Sie beim Erreichen der Stadt (irgendwo ist die bestimmt) die letzte Barriere mit der Feuertaste (Leertaste) beseitigen. Das Spiel ist beendet, wenn Sie auf ein Haus zufahren. Wir wünschen Ihnen eine ruhige Hand (einige Durchfahrten sind wirklich gemein eng) und viel Spaß auf der Entdeckungsreise.

64er Tips

Die Garbage-Collection im C-64 ist etwas, was die wenigsten kennen. Vielleicht haben Sie den Begriff schon einmal gehört, aber können Sie so richtig was damit anfangen? Gut, dem soll hier abgeholfen werden. Auch wenn Sie nicht selbst programmieren, sollten Sie nicht sofort weiterblaettern, denn hier finden Sie vielleicht die Erklaerung, warum einige Ihrer Programme scheinbar nicht richtig funktionieren. Damit jeder in die Sache einsteigen kann, fangen wir also mit den Grundlagen an:

Variablen und Strings

Wenn man ein BASIC-Programm schreibt, braucht man öfter mal Speicherplätze, in denen man Zwischenergebnisse ablegen kann, also gewissermassen eine Schublade .
Diese Schubladen nennt man nun Variablen ('variabel' bedeutet veränderlich). Damit wird ausgedrückt, daß das, was man darin ablegt, sich ändern kann. Legt man nun etwas in einer Variablen ab, so heißt das Wertzuweisung . Ein Beispiel für eine Wertzuweisung :

A = 5

Hier wird die Zahl 5 in der Variablen A abgelegt. Im Computerchinesisch heißt das:

A wird der Wert 5 zugewiesen

Hier haben wir auch gleich gesehen, wie Variablen bezeichnet werden: Mit Buchstaben nämlich! Jede Variable darf einen Namen aus bis zu zwei Buchstaben haben. Geben Sie mehr Buchstaben an, so schadet das in der Regel nicht; die überzähligen Zeichen werden einfach nicht beachtet. Der Wert der Variablen ist jederzeit abrufbar, z. B. mit

PRINT A

Diese Anweisung bringt die Zahl 5 auf den Bildschirm.

Da es im Programmlauf Zwischenergebnisse unterschiedlicher Art gibt, bietet uns der C-64 auch verschiedene Schubladentypen an (In eine Besteckschublade passen nunmal keine Teller. . .).

Es gibt 3 verschiedene Typen von 'Zwischenergebnissen':

1) Ganze Zahlen (Integerzahlen)

Das sind alle Zahlen von -32768 bis 32767, aber nur ganze Zahlen, also solche ohne Komma bzw. Dezimalpunkt.
Integerzahlen sind: -5 0 1000 799 . . .

2) Reele Zahlen (Realzahlen, Fließkommazahlen)

Das sind alle Zahlen mit oder ohne Dezimalpunkt.
Reele Zahlen sind: -3.22 988.2345 3.67e-12 3.141592654 . . .

Warum nun dieser Unterschied? Nun, Integerzahlen werden vom C-64 wesentlich schneller bearbeitet als Reele-Zahlen und brauchen, als Felder, auch viel weniger Speicherplatz!

3) Zeichenketten (Strings)

Strings sind nun eine ganz besondere Gruppe. Ein String ist alles das, was sich zwischen zwei Anführungszeichen aufhält,z.B.:
Hallo, lieber Leser Bitte eine Taste drücken
!#$%&'()kjhlöjkhfs
Strings können eine Länge von 0 bis 255 Zeichen haben.

Deshalb gibt es auch 3 verschiedene Schreibweisen für Variablen:

Realvariablen werden ganz normal bezeichnet, wie oben beschrieben,
Integervariablen werden durch ein zusätzlich nachgestelltes % gekennzeichnet, z.B.: A% NN% X%
Stringvariablen schließlich werden mit $ gekennzeichnet, z.B.: A$ BB$ GH$

Wie werden die Variablen im C-64 abgelegt?

Wenn man einer Variablen einen Wert zuweist und diesen später wieder abrufen kann, dann muß er irgendwo zwischengespeichert sein. Im Magazin in den Tips präsentieren wir Ihnen auf den ersten beiden Seiten die Speicheraufteilung des C-64. Dort sehen Sie zwei Bereiche, die mit Variablen und Felder bezeichnet sind. Die Felder wollen wir nun nicht weiter betrachten.

Für jede Variable ein Bereich von 7 Byte reserviert. Die ersten beiden Bytes werden für den Variablennamen gebraucht, die restlichen fünf Byte werden je nach Variablentyp verwendet.

Integerzahlen können mit zwei Bytes dargestellt werden, die restlichen 3 Byte bleiben leer.

Realzahlen werden dort im Fließkommaformat abgelegt und brauchen dazu alle 5 Bytes.

Wie ist das aber nun mit Strings? Sie können ja schließlich eine Länge von bis zu 255 Zeichen

haben, das heißt, für sie werden bis zu 255 Bytes Speicherplatz gebraucht. Strings können also nicht im Variablenbereich liegen, sondern werden auf den sogenannten Stringstapel gebracht.

Es wird aber trotzdem im Variablenbereich so ein 7-Byte-Eintrag gemacht: Im ersten und zweiten Byte steht, wie immer, der Name, in Byte 3 steht die Länge des Strings und in Byte 4 und 5 schließlich seine Adresse im low/high-Format. Byte 6 und 7 bleiben leer.

Diese Bytes 3-5 heißen auch Stringdescriptor ('Descriptor' bedeutet Beschreiber). Wird nun einem String ein Wert zugewiesen,z.B.:

A$= Hallo Heinz ,

so wird Hallo Heinz auf dem nächsten freien Platz im Stringstapel eingetragen (man sagt, der String wird auf den Stringstapel geworfen).
Die Adresse und seine Länge werden im Stringdescriptor eingetragen.
Wenn wir jetzt derselben Variablen einen anderen String zuweisen, etwa

A$= Hallo Ralph

dann wird dieser neue String wieder auf den Stringstapel geworfen, und die neue Adresse und Länge im alten Stringdescriptor eingetragen. Ja, und was ist mit dem alten String? könnte man jetzt fragen. Und hier fängt das Problem an: Der alte String wird nämlich nicht gelöscht, er steht weiterhin einsam im Speicher herum. Es gibt keinen Descriptor mehr, der seine Adresse enthält; man darf ihn wirklich als Stringmüll bezeichnen. Warum macht man denn sowas? Wenn wir in unseren Beispiel fortfahren und A$ immer neue Strings zuweisen, dann haben wir bald den ganzen Speicher voll, voll Müll nämlich.

Der Grund für diese Technik ist, daß es nur so möglich ist, die Stringverwaltung einigermaßen schnell zu gestalten. Würde man jedesmal erst prüfen, ob ein String zu löschen ist, wäre der BASIC-Interpreter noch langsamer, als er jetzt schon ist. Heißt das jetzt, daß wir nicht zu warten brauchen? Nein, heißt es leider nicht. Der dicke Hund kommt jetzt:

Die Garbage Collection

Wenn der ganze Speicher voller Stringmüll ist, dann müßte beim nächsten Versuch, einen String anzulegen, ein OUT OF MEMORY ERROR kommen. Das passiert nun zum Glück nicht, denn jetzt kommt die eingebaute Müllabfuhr des Computers zum Zuge; die Computerleute sagen dazu Garbage Collection ('Garbage' bedeutet Müll, 'Collection' bedeutet Sammlung). Und diese Sache ist beim C-64 eine ziemlich langweilige Angelegenheit.

Stellt der BASIC-Interpreter fest, daß kein Platz mehr im Speicher ist, dann geht er die gesammte Variablenliste durch, um den String zu finden, der noch gültig ist und im Stringstapel am weitesten unten liegt. Dieser String wird nun nach ganz unten in den Stringstapel gelegt. Eventuell dort liegende Müllstrings werden dabei natürlich einfach überschrieben. Der Stringdescriptor wird entsprechend geändert, d. h. die neue Adresse wird eingetragen. Anschließend wird die Variablenliste wieder durchgegangen und der nächstuntere String gesucht, der noch gültig ist. Der eben bearbeitete String wird natürlich dabei übergangen, ebenso Strings, deren Länge null ist, da diese keinen Inhalt haben. Ist dieser String nun gefunden, wird er über dem ersten plaziert und sein Descriptor entsprechend geändert.

Ahnen Sie was? Natürlich, der Vorgang geht immer so weiter, bis alle Strings bearbeitet sind. Es ist dann ein neuer, aufgeräumter Stringstapel entstanden.

Warum ist die Garbage Collection so gefürchtet?

Na klar: Weil Sie so unheimlich lange dauern kann, und Sie verstehen jetzt auch, warum: Für jeden im Programm benutzten String muß ein Durchlauf durch die Variablenliste gemacht werden, wobei alle Strings überprüft werden müßen. Ein Beispiel:

Sie haben ein Programm, das 100 Stringvariablen benutzt. Kommt es nun zur Garbage Collection, werden 100 Durchläufe durch die Variablenliste benötigt, wobei jeweils 100 Strings überprüft werden müssen. Das ergibt 100*100 = 10000 Überprüfungen! Bei 1000 Strings (die ein Datenverwaltungsprogramm leicht zustande bringt) sind das schon 1 Million Überprüfungen.

Dabei kommt es fast überhaupt nicht darauf an, ob in einem String 1 oder 255 Zeichen stehen. Auch die Anzahl der produzierten Müllstrings spielt keine Rolle, sie werden ja gar nicht beachtet. Für die benötigte Zeit spielt wirklich nur die Anzahl der gültigen Strings, die mindestens 1 Zeichen enthalten, eine Rolle!

Wenn Sie z.B. mit einer Dateiverwaltung arbeiten, die nach dem 20.ten Datensatz scheinbar den Geist aufgibt, dann handelt es sich wahr-

scheinlich um die Garbage-Collection. Wenn Sie läuft, benimmt sich der C-64 wie tot, das heißt, er nimmt keine Eingaben mehr an und zeigt unbeweglich sein Bild. Die Interrupts laufen allerdings weiter, und alles, was damit zusammenhängt, z.B. Musik im Hintergrund bei irgendwelchen Spielen. Wenn Sie also davorsitzen und sich zum X.ten Mal ärgern, das der Computer abgestürzt ist und alle Ihre Daten mit in den Abgrund gerissen hat, probieren Sie es mal mit Geduld: Mit großer Wahrscheinlichkeit meldet er sich nämlich zurück und Sie können frohen Mutes weiterarbeiten. Sie brauchen dazu allerdings viel Geduld, erfahrene Anwender und Programmierer berichten von Zeiten bis zu einer Stunde!!!

Sind Sie nur Anwender, so bleibt Ihnen nichts, als dieses geduldig zu ertragen, oder sich ein besseres Programm zu kaufen. Programmieren Sie aber selber, so ist das ein Grund, in den eigenen Programmen Maßnahmen gegen diese unmöglichen Wartezeiten einzubauen.

Was kann man denn gegen diese langen Wartezeiten tun?

In einigen Fachzeitschriften liest man ab und zu den Tip, die Garbage Collection beizeiten selbst auszulösen, in der Annahme, die Zeit sei kürzer, wenn sich noch nicht soviel Müll angesammelt hat. Wie wir eben gesehen haben, ist das grundverkehrt. Sie haben also nur die Möglichkeit, darauf zu achten, daß Sie nicht zuviele Stringvariablen benutzen. Wenn Sie unter 100 bleiben, dann bleibt die Garbage Collection unter einer Sekunde. (Sie können das in dem Beispielprogramm, das Sie aus dem Magazin heraus mit CTRL&S abspeichern können, ausführlich ausprobieren).

Merke:Lieber mit wenig Variablen viel Müll produzieren, als mit vielen wenig!

Umgekehrt gilt natürlich: Wenn Sie schon viele Stringvariablen benutzen müssen, dann sollten Sie mit dem produzieren von Stringmüll sparsam sein, damit es erst gar nicht zur Garbage Collection kommt Eine andere Möglichkeit ist, Variablen, deren Inhalte nicht mehr gebraucht werden, zu löschen:

A$=

Sie wissen ja: Strings mit der Länge null werden nicht beachtet! Solten Sie einen Punkt im Programm haben, der regelmäßig durchlaufen wird, und an dem Sie die meißten Stringvariablen löschen können, dann können Sie dort gezielt die Garbage Collection selbst auslösen. Danach haben Sie dann wieder einen aufgeräumten Speicher zur Verfügung. Außerdem müssen Sie die Garbage Collection auslösen, wenn im Programm vor einem zeitkritischen Abschnitt stehen, bei dem Sie sich keine Unterbrechung leisten können.

Wie löse ich die Garbage Collection aus?

Zum einen gibt es den Befehl
SYS 46374
zum anderen die BASIC-Funktion
FRE(0)
welcher ja den noch freien Speicherbereich berechnet und daher vorher den Speicher aufräumt, damit das Ergebnis auch stimmt. Vor- oder Nachteile der Methoden sind uns nicht bekannt, Sie können also beide Möglichkeiten beliebig verwenden.

Etwas haben wir Ihnen noch verschwiegen: Es gibt Strings, die nicht auf dem Stringstapel liegen, und das sind solche, die im Programm definiert werden, z.B.:
10 A$= Na so was
20 B$= Du Stoffeltier!
Es ist ja auch gar nicht nötig, diese Strings auf den Stapel zu werfen, als Adresse im Stringdescriptor wird einfach die Adresse im BASIC-Programm eingetragen. Diese Strings werden von der Garbage Collection natürlich auch belassen, wo Sie sind. Normalerweise ist das schön und gut so, es gibt aber einen Fall, wo der String doch lieber auf dem Stapel liegen sollte: Wenn nämlich aus dem BASIC-Programm ein zweites BASIC-Programm nachgeladen wird.(Overlaytechnik). Ist das zweite Programm kleiner als das erste, dann bleiben die alten Variablen erhalten, und das soll ja auch so sein. Das Programm soll ja mit den alten Variablen weiterarbeiten. Was ist aber jetzt mit den im Programm stehenden Strings? Klar: die sind natürlich im Eimer, da steht jetzt nur noch Mist drin! Abhilfe schafft hier folgender Trick: Addieren Sie zu dem String einfach noch einen Nullstring hinzu. Das verändert den String nicht, aber er steht sogleich im Stringstapel:
10 A$= Love me Tender :A$=A$+
Wenn Sie sich bis hierhin durchgearbeitet haben, sind Sie ein voll ausgebildeter C-64 Müllfachmann (-fachfrau), und eigentlich kann ja nun nicht mehr viel schiefgehen!

Des LISP Teil 3

Lästermäuler bezeichnen die beiden LISP-Programme, die wir Ihnen diesmal als dritten (und letzten) Teil des LISP64-Pakets vorstellen, als "LISP-Oldies". Ganz falsch ist das nicht – das Dialogprogramm ELIZA und "selbstlernende" Expertensysteme sind wirklich die klassichen Beispiele der Anwendung sogenannter KI-Prinzipien, der Grundregeln der "Künstlichen Intelligenz" also. Daß diese beiden Programme nur unter dem in Ausgabe 4/86 vorgestelltem LISP-Interpreter lauffähig sind, versteht sich wohl von selbst. Damit dürfte auch klar sein, daß ELIZA.LSP und EXPERTE.LSP nicht innerhalb des Magazins ihre Funktion demonstrieren können, sondern erst auf Ihre eigene Diskette/Kassette abgespeichert werden müssen.

EXPERTE.LSP

Sogenannte Expertensysteme dienen der Darstellung und Verarbeitung von "Wissen" über betimmte Sachgebiete. So gibt es große Systeme, die chemische Analsen auswerten können, Krankheiten diagnostizieren oder geologische Karten auf Erdölvorkommen hin untersuchen. (Wie so etwas prinzipiell auch in BASIC gelöst werden kann, demonstriert das Programm SCHLAUMEIER an anderer Stelle in dieser Ausgabe.)

Unser LISP-Beispielprogramm ist natürlich bescheidener als seine beschriebenen "großen Schwestern". Es soll nur die allgemeinen Grundlagen einer bestimmten Form von Wissensaufbereitung demonstrieren, nämlich die Darstellung von Wissen als **Fakten, Regeln** und **Hypothesen**

Fakten sind einfache Tatsachen oder wahre Aussagen, zum Beispiel:
- Es regnet
- Ein Vogel kann fliegen
- Alle geraden Zahlen sind durch zwei teilbar.

Regeln bestehen aus einem "Wenn"-Teil (den Prämissen) und einem "Dann"-Teil (den Konklusionen), zum Beispiel:
- Wenn es regnet, dann muß ich einen Schirm mitnehmen.
- Wenn ein Tier ein Vogel ist, dann kann es fliegen, hat Federn und legt Eier
- Wenn die Spannung an der Basis eines NPN-Transistors 5 Volt beträgt und der Transistor leitet, dann beträgt die Spannung am Emitter 0.5 – 0.6 Volt.

Die Funktion der Regeln bei einer Deduktion (Ableitung vom Allgemeinen zum Besonderen) oder Diagnose (siehe unten) ist folgende: sind alle Prämissen einer Regel als Fakten bekannt, also wahr, dann sollen auch die Konklusionen dieser Regel als wahr gelten und in die Liste der Fakten aufgenommen werden.

Hypothesen schließlich sind Aussagen, die erst noch aus dem Zusammenspiel der Fakten und Regeln als wahr oder falsch erwiesen werden müssen.

Zum Programm

Nach dem Starten mit (DO) befindet man sich im Dialog-Modus: hier kann man – in (nahezu) natürlicher Sprache – dem System Aufträge erteilen ("Mache eine Diagnose"), Fragen stellen ("Hast Du Regel 3 benutzt ?"), Daten ausgeben lassen ("Drucke alle Regeln") und nicht zuletzt die Regeln definieren, Fakten eingeben und Hypothesen aufstellen.

Die Eingaben brauchen nicht geklammert zu werden (LISP-untypisch). Achtung: ein Punkt als Satzabschluß ist nicht erlaubt (da der Punkt ein spezielles Syntaxzeichen ist); schließt eine Eingabe mit einem Doppelpunkt ab, so wird eine weitere Eingabezeile zur Verfügung gestellt, um längere Texte zu ermöglichen.

Die Kommandos

Für die folgende Beschreibung der Kommandos von EXPERTE.LSP gelten folgende Vereinbarungen:

- die Reihenfolge ist Kommando, Erklärung, Beispiel
- alternative Eingabemöglichkeiten zum Befehl sind (in Klammern eingeschlossen) hinter dem Kommando angegeben; diese Klammern müssen natürlich nicht mit eingegeben werden
- Leerzeichen und Doppelpunkte müssen, wenn sie angegeben sind, unbedingt mit eingegeben werden
- ein Sternchen (*) steht für einen beliebigen Text, statt WIE * kann zum Beispiel eingegegeben werden WIE HAST DU DAS GE-

MACHT. (Für Tautologen: natürlich kann auch ein * statt des * eingegeben werden . . .)

WENN : erste Prämisse
leitet die Definition einer Regel ein, indem die erste Prämisse angegeben wird.
WENN : DAS TIER KANN FLIEGEN

UND (UND WENN) : nächste Prämisse
Hat eine Regel mehr als eine Prämisse, werden die weiteren durch UND verbunden (ein ODER gibt es nicht, es ergibt sich aus dem Aufstellen mehrerer Regeln mit gleichen Konklusionen/Prämissen)
UND : DAS TIER LEGT EIER

DANN : erste Konklusion
Angabe der (ersten) Konklusion einer Regel
DANN : DAS TIER IST EIN VOGEL

UND DANN : nächste Konklusion
Angabe der weiteren Konklusion einer Regel
UND DANN : DAS TIER HAT FEDERN

ALS HYPOTHESE (ALS *)
Die zuletzt eingegebene Konklusion soll als Hypothese für eine spätere Diagnose dienen (im vorigen Beispiel: DAS TIER HAT FEDERN)

VERGISS (LOESCHE) * REGEL n *
Entfernen der Regel Nummer n (die Regeln sind durchnumeriert) aus der Regelmenge.
VERGISS REGEL 3

VERGISS (LOESCHE) * REGELN (R)
Löschen aller Regeln, zum Beispiel bevor eine neue Aufgabenstellung eingegeben werden soll.
VERGISS ALLE REGELN

DRUCKE (ZEIGE) * REGELN
Ausgabe der Regel Nummer n
DRUCKE REGEL 5

DRUCKE (ZEIGE) * REGELN (R)
Ausdrucken aller Regeln
DRUCKE ALLE REGELN

* FAKTUM (LERNE, MERKE) * : Faktum
Füge "Faktum" in die Liste der Fakten ein
FAKTUM IST: DAS TIER KANN FLIEGEN
MERKE DIR: EIN VOGEL HAT FEDERN
VERGISS (LOESCHE) * FAKTUM : Faktum
Entferne "Faktum" aus der Liste der Fakten
VERGISS DAS FAKTUM : DAS TIER KANN FLIEGEN

VERGISS (LOESCHE) * FAKTEN (F)
alle bisherigen Fakten werden gelöscht
VERGISS ALLE FAKTEN

DRUCKE (ZEIGE) * FAKTEN
Ausdrucken aller Fakten
DRUCKE DIE FAKTEN

* HYPOTHESE (HYP) * : Hypothese
Füge "Hypothese" in die Liste der Hypothesen ein
EINE HYPOTHESE SEI : DAS TIER IST EIN ALBATROSS

VERGISS (LOESCHE) * HYPOTHESE (HYP) : Hypothese
Entferne "Hypothese" aus der Liste der Hypothesen.
VERGISS DIE HYP : DAS TIER IST EINE GIRAFFE

VERGISS (LOESCHE) * HYPOTHESEN (H)
Alle bisherigen Hypothesen werden gelöscht
VERGISS DIE HYPOTHESEN

DRUCKE (ZEIGE) * HYPOTHESEN (H)
Ausdrucken aller Hypothesen
DRUCKE ALLE HYPOTHESEN

* PRAEMISSE * : Prämisse
Ausdrucken derjenigen Regeln, die "Prämisse" im Wenn-Teil enthalten
WELCHE REGELN BENUTZEN DIE PRAEMISSE :
DAS TIER IST EIN VOGEL

* KONKLUSION * : Konklusion
Ausdrucken derjenigen Regeln, die "Konklusion" im Dann-Teil enthalten
WELCHE REGELN HABEN DIE KONKLUSION :
DAS TIER KANN FLIEGEN

* DEDUKTION (DEDIZIERE; DEDUZIEREN) *
Bei einer Deduktion wird versucht, mit Hilfe der bisher bekannten Fakten durch die Prüfung aller Regeln weitere Fakten zu erhalten
MACHE EINE DEDUKTION
VERSUCHE ETWAS ZU DEDUZIEREN

* DIAGNOSE *

Bei einer Diagnose wird versucht, eine der verschiedenen Hypothesen mit Hilfe der Regeln und Fakten als wahr zu erweisen; hierbei wird die Wahr- oder Falschheit von Regel-Prämissen, die dem Programm noch nicht bekannt sind, vom Benutzer erfragt
MACHE EINE DIAGNOSE

* ANGEWENDET (BENUTZT) * n
Frage, ob bei einer Deduktion oder Diagnose die Regel Nummer n angewandt wurde.
HAST DU REGEL 3 BENUTZT

WIE * : Konklusion
Frage, mit Hilfe welcher Fakten die "Konklusion" erschlossen wurde.
WIE HAST DU DEDUZIERT : DAS TIER KANN FLIEGEN

WARUM * : Prämisse
Frage, welche Fakten (Konklusionen) aus der "Prämisse" folgten
WARUM HAST DU BENUTZT : DAS TIER IST EIN VOGEL

WELCHE *
Frage nach den erfolgreich benutzten Regeln:
WELCHE REGELN HAST DU BENUTZT

ZOOTIERE.DAT

Dieses File enthält einige Regeln und Hypothesen über Tiere in einem Zoo und dient zur Demonstration des LISP-"Experten" in Form eines kleinen Ratespiels. Es muß nach EXPERTE.LSP geladen werden und dient diesem als "Datenbank".

Nachdem man sich mit DRUCKE HYPOTHESEN die Hypothesen angesehen hat, wähle man eine Möglichkeit für sich aus (zum Beispiel "Das Tier ist eine Giraffe"). In Form einer DIAGNOSE läßt man den Rechner jetzt untersuchen, welches Tier gemeint ist: MACHE EINE DIAGNOSE und beantwortet dementsprechend die Fragen des "Experten".

Vor einer erneuten Diagnose sollte mit VERGISS ALLE FAKTEN zunächst die Faktenliste, die bei einer Diagnose aufgestellt wurde, gelöscht werden.

Beispielssitzung

Laden Sie (unter LISP64 natürlich) erst EXPERTE.LSP und dann ZOOTIERE.DAT und starten das Programm mit (DO). Dann bringen Sie dem Experten eine neue Regel bei:

WENN : DAS TIER KANN FLIEGEN
UND WENN : DAS TIER IST KEIN VOGEL
DANN : DAS TIER IST EINE FLEDERMAUS

Jede Zeile muß mit RETURN abgeschlossen werden. Dann:

MACHE EINE DIAGNOSE

Beantworten Sie die nun folgenden Fragen des "Experten" so, daß er die Fledermaus diagnostizieren soll. Also "DAS TIER FRISST FLEISCH" mit Nein und so weiter.

ELIZA.LSP

ELIZA ist ein einfaches, aber doch nicht zu triviales Standard-Beispiel für Simulationen mit "Künstlicher Intelligenz": es imitiert den "Seelendoktor" in einem psychotherapeutischen Gespräch, indem es den Patienten nach seinen Problemen befragt, auf dessen Intentionen eingeht, selber Fragen beantwortet und so weiter.

Zur Bedienung

Nach dem Laden wird das Programm durch Eingabe von (ELIZA) gestartet. Die Aufforderung (BITTE ERZAEHLE MIR VON DEINEN PROBLEMEN :) leitet dann den Dialog zwischen dem Patienten (Mensch) und dem Psychotherapeuten (Computer!) ein. Die "Stimmigkeit" oder "Echtheit" des Dialogs hängt dabei in hohem Maße davon ab, ob man sich auf ein ernsthaftes Gespräch einläßt. Wichtiger Hinweis: Bei der Eingabe eines Satzes sind alle syntaktischen Zeichen (.,!? und so weiter) fortzulassen.

Das Programm

Die Hauptfunktion ELIZA dient lediglich dazu, eine Begrüßungsformel zu drucken, den Dialog zu starten und nach dessen Beendigung eine Abschiedsmeldung auszugeben.

Die Funktion DIALOG ist für den Ablauf des Mensch-Maschine-Dialogs zuständig: Nach dem Einlesen (SETQ SATZ (READL)) wird versucht, auf diese Eingabe eine passende Antwort zu finden (FINDE-ANTWORT, siehe unten). Wird Sie gefunden, so wird Sie ausgedruckt (PRINC RESULTAT), die LISP-Funktion PRINC unterdrückt beim Ausgeben die Klammern eines LISP-Ausdrucks). Wird keine direkte Antwort gefunden, dann wird eine Ersatz-Antwort gegeben. Ist auch die Liste der Ersatz-Antworten erschöpft, so gilt der Dialog als beendet: (RETURN NIL). Ansonsten bewirkt (GO LOOP1), daß die nächste Eingabe eingelesen und so der Dialog fortgeführt wird.

Um eine möglichst "intelligente", das heißt passende Antwort zu finden, wird die Eingabe mit Mustersätzen (englisch: pattern) verglichen, von denen die jeweiligen Antwortsätze abhängig sind. Die Variable DIALOG-REGELN enthält eine Liste mit solchen Muster-Antwort-Paaren:

DIALOG-REGELN =
((Muster1 Antwortsatz1)

```
(DE ELIZA ()
  (MSG (CHAR 147) T
       "HALLO, ICH BIN ELIZA !" T T
       "BITTE ERZAEHLE MIR VON" T T
       "DEINEN PROBLEMEN :" T T)
       (DIALOG ERSATZ-ANTWORTEN)
       (MSG T "WIR MUESSEN UNSERE SITZUNG" T
            "LEIDER BEENDEN." T T
           "TSCHUESS !" T T))

(DE DIALOG (ERSATZ-ANTWORTEN)
  (PROG (RESULTAT SATZ)
        LOOP1
        (SETQ SATZ (READL))
        (SETQ RESULTAT (FINDE-ANTWORT SATZ DIALOG-REGELN))
        (COND (RESULTAT (PRINC RESULTAT))
              (ERSATZ-ANTWORTEN
                        (PRINC (CAR ERSATZ-ANTWORTEN))
                        (SETQ ERSATZ-ANTWORTEN
                           (CDR ERSATZ-ANTWORTEN)))
              (T (RETURN NIL)))
        (TERPRI)
        (GO LOOP1)))

(DE FINDE-ANTWORT (S R)
  (PROG (RESULTAT)
        LOOP
        (COND ((NULL R) (RETURN NIL))
              ((MATCH (CAAR R) S)
                        (SETQ RESULTAT (CAR(CDAR R)))
                        (NCONC1 R (CAR R))
                        (RPLACA R (CADR R))
                        (RPLACD R (CDDR R))
                        (RETURN RESULTAT)))
        (SETQ R (CDR R))
        (GO LOOP)))

(DE MATCH (P S)
  (COND ((NULL P) (NULL S))
        ((EQ (CAR P) '*)
                  (COND ((NULL S) (NULL (CDR P)))
                        ((MATCH (CDR P) S))
                        ((MATCH P (CDR S)))))
        ((NULL S) NIL)
        ((EQ (CAR P) (CAR S))
                  (MATCH (CDR P) (CDR S)))
        ((AND (CONSP (CAR P))
              (MEMBER (CAR S) (CAR P)))
                  (MATCH (CDR P)(CDR S)))))
```

(Muster2 ANtwortsatz2)
. . .)

Ein Muster ist hierbei eine Liste von Wörtern mit folgenden Besonderheiten:
-***das Zeichen * steht für eine Folge von beliebig vielen Wörtern
-***statt eines einzelnen Wortes kann eine Liste mit (alternativen) Wörtern stehen.

Muster sind zum Beispiel:

```
(DIES IST EIN MUSTER)
(* ICH *)
(GUTEN (TAG MORGEN ABEND))
(* ICH (MAG LIEBE HASSE) * MEN
SCHEN)
```

Das sogenannte "pattern-matching", das heißt, der Vergleich eines einzelnen Satzes mit einem Muster, führt die Funktion MATCH (=Vergleich) durch. Deren Wert ist T (true), wenn alle Wörter des Satzes in das Muster eingeordnet werden können, sonst NIL. Zum Beispiel:

```
(MATCH*********'(GUTEN (TAG MOR-
        GEN ABEND))
'(GUTEN MORGEN))
= T

(MATCH*********'(* IST *) '(DIE
        SONNE IST HEUTE ABEND ROT)
= T
```

Die Funktion FINDE-ANTWORT wendet Match nacheinander auf die in der Liste DIALOG-REGELN enthaltenen Muster an, bis ein Vergleich zutrifft oder die Liste erschöpft ist. Im ersten Falle wird das Muster/Antwort-Paar an das Ende der DIALOG-REGELN befördert (durch die Funktionen NCONC1 bis RPLACD), damit bei einer späteren gleichen oder ähnlichen Eingabe zuerst mit anderen Mustern verglichen und so (eventuell) eine andere Antwort gefunden wird.

Angemerkt sei, daß es, nachdem der Aufbau der globalen Liste DIALOG-REGELN beschrieben ist, sehr leicht ist, das Verhalten von ELIZA zu ändern, indem man die Liste ergänzt oder Muster/Antwort-Paare löscht. Soll ELIZA zum Beispiel auf die Eingabe ICH HABE ANGST mit der Frage WOVOR FUERCHTEST DU DICH? reagieren, wäre die folgende Ergänzung denkbar:

```
(NCONC1 DIALOG-REGELN
'((ICH HABE (ANGST FURCHT) *)
(WOVOR FUERCHTEST DU DICH?)))
```

Die c't Uhr

Die Systemprogramme zur Echtzeituhr

Vorgeschichte

In unserem Schwester-Magazin c't Ausgabe 4/86 ist gerade eine akkugepufferte Echtzeituhr vorgestellt worden. Im ersten Moment wird man sich fragen, wozu eine Echtzeituhr am C64? Im C64 ist doch bereits eine Echtzeituhr enthalten? (siehe auch unsere Ausgabe 2/86). Vom Prinzip her stimmt das zwar, jedoch bietet die c't Echtzeituhr einiges mehr.

Das Zauberwort heißt 'akkugepuffert'!. Die Wirkung: selbst nach Ausschalten des Rechners läuft die Uhr ohne Unterbrechung weiter und zählt nicht nur Sekunden, Minuten und Stunden, sondern auch Wochentag, Tages-, Monats- und Jahresdatum. Sogar Schaltjahre werden mit berücksichtigt. Die Uhr verkraftet mit vollem Akku eine 'Rechnerpause' von mindestens vier Monaten, auch im ausgebauten Zustand. Sie können bei Bedarf den Rechner Ihres Bekannten damit schmücken, ohne daß Sie die Uhr neu stellen müssen.

Man kann, mit entsprechender Software, die Uhr beispielsweise dazu benutzen, seinen Dateinamen Datum und Uhrzeit in Kurzform anzuhängen, was das Wiederauffinden erheblich vereinfacht und Ordnung im Dateienwirrwar schaffen hilft.
Wer genaueres über diese Uhr wissen möchte, kann in c't 4/86 (S. 44 ff) Details über Aufbau und Funktionsweise nachlesen. Die Basisplatine inclusive PAL-Baustein (ein programmierbarer Logik-Chip) kann beim Heise-Verlag - Stichwort c't Platinen-Service zum Preis von 53,- DM bestellt werden (siehe Impressum).

Vorarbeiten

Die Redaktionen beschlossen, ihre Beziehungen zu nutzen und die Software für den C64 in INPUT 64 zu veröffentlichen, um Ihnen die mühselige Abtipperei zu ersparen. Bei der Entwicklung der Software zeigten sich jedoch ernst-

zunehmende Hardware-Probleme: Die Uhr ist in der ursprünglichen Version im C64 nur bedingt verwendbar. Wird sie über das BASIC-ROM gesetzt, ist ein normaler BASIC-Betrieb nicht mehr möglich. Da dies nicht im Sinne des Erfinders ist, wurde die Uhr geringfügig modifiziert. Sie kann jetzt im Huckepack-Betrieb auf das Kernal-ROM gesetzt werden. Sie müssen vorher zwei zusätzliche Kondensatoren von 10 nF zwischen PIN1 und Masse, und PIN12 und Masse des PAL-Bausteins einlöten. Dadurch werden Störimpulse auf diesen Leitungen ausgefiltert, sodaß die Uhr nicht per Zufall eingeblendet wird und damit das Kernal lahmlegt.

Sie sehen: INPUT 64 Leser sind mal wieder im Vorteil. Im Modul "c't Uhr" finden Sie eine kurze Erläuterung für die Programmierung und können dort mit CTRL+S zwei Systemprogramme, eins zum Lesen und eins zum Neusetzen der c't Echtzeituhr, und ein BASIC-Beispiel abspeichern.

Befehlssyntax:

Schreibprogramm zum Setzen der Uhr:
SYS 2144,JJMMTTWhhmmss

JJ = Jahr MM = Monat TT = Tag
W = Wochentag

hh = Stunde mm = Minute ss = Sekunde

Dieses Programm benötigen Sie natürlich nur einmal.

Leseprogramm:

SYS 2144,SADR,PADR verschiebt das Programm an die gewünschte Adresse 'SADR' und legt den Puffer nach 'PADR'

SYS SADR schreibt die Uhrzeit im 1/2 Sekundenzyklus in die Adresse 'padr'.

SYS 2144,8606093120000 stellt die Uhr auf 12.00 Uhr am Mittwoch (3. Wochentag) den 09.06.86. Für Sonntag wird 0 angegeben!

SYS 2144,51200,1024 verschiebt das Leseprogramm nach 51200 ($C800) und verwendet den Bildschirm als Puffer.

SYS 51200 würde die Zeichenkette JJMMTTWhhmmss in fortlaufender Zeit oben links im Bildschirm erscheinen lassen. Sie können selbstverständlich auch einen anderen Puffer-Bereich festlegen –($033c = 828) den Kassetten-Puffer zum Beispiel. Aus dem Modul c't Uhr können Sie sich zusätzlich noch ein einfaches BASIC-Programm abspeichern, mit dem Sie bei Eingebauter c't Uhr Datum und Uhrzeit im Klartext auf dem Bildschirm angezeigt bekommen. Zur Anwendung des Programms müssen Sie:

- Die c't Uhr in den Sockel des Kernals stecken und das Kernal-Rom in der Fassung auf der Uhren-Platinen unterbringen (Einbaurichtung beachten!!),

- das Leseprogramm laden,

- mit SYS 2144,51200,828 das Leseprogramm verschieben und den Puffer bei 828 definieren,

- mit SYS 51200 die Leseroutine im Interrupt aktivieren,

- das BASIC-Programm laden und mit RUN anstarten.

Wenn Sie die Uhr irgendwann einmal mit dem aktuellen Datum beschrieben haben (siehe oben), erscheint das augenblickliche Datum samt Uhrzeit auf dem Bildschirm.

Hinweise zur Bedienung

Bitte entfernen Sie eventuell vorhandene Steckmodule. Schalten Sie vor dem Laden von INPUT 64 ihren Rechner einmal kurz aus. Geben Sie nun zum Laden der Kassette *LOAD* und *RETURN* oder *SHIFT* und gleichzeitig *RUN/STOP* bzw. der Diskette *LOAD"INPUT*",8,1* und *RETURN* ein. Alles weitere geschieht von selbst.

Nach der Titelgrafik springt das Programm ins Inhaltsverzeichnis des Magazins. Dieses können Sie nun mit der *SPACE* (Leertaste) durchblättern. Mit *RETURN* wird das angezeigte Programm ausgewählt. Im Fenster unten rechts erhalten Kassettenbesitzer weitere Hinweise ("Bitte Band zurückspulen" und so weiter . . .). Haben Sie bei der Auswahl eines Programms eventuell nicht weit genug zurückgespult, und es wurde nicht gefunden, spulen Sie bis zum Bandanfang zurück. Diskettenbesitzer stellen bitte sicher, daß noch die INPUT 64-Diskette eingelegt ist. Auf der 2. Kassettenseite befindet sich eine Sicherheitskopie. Sollten Sie eventuell mit einem der Programme Ladeschwierigkeiten haben, versuchen Sie es auf Seite 2. Führt auch dies nicht zum Erfolg, lesen Sie bitte die entsprechenden Hinweise im Kapitel "Bei Ladeproblemen"!

Neben der Programmauswahl mit *SPACE* und dem Ladebefehl mit *RETURN* (im Inhaltsverzeichnis) werden die übrigen 'System-Befehle' mit der Kombination aus *CTRL*-Taste und einem *Buchstaben* eingegeben. Sie brauchen sich eigentlich nur *CTRL* und *H* zu merken (Aufruf der Hilfsseite), denn dort erscheinen die jeweils möglichen weiteren 'System-Befehle'. Nicht immer sind alle Optionen möglich (eventuell werden Sie zu Beginn des Programms auf Einschränkungen hingewiesen). Hier nun alle INPUT 64-Systembefehle:

CTRL und *Q* (ab Ausgabe 3/85)

Sie kürzen die Titelgrafik ab; INPUT 64 geht dann sofort ins Inhaltsverzeichnis.

CTRL und *H* (ab Ausgabe 1/85)

Es wird ein Hilfsfenster angezeigt, auf dem alle verfügbaren Befehle aufgeführt sind.

CTRL und *I* (ab Ausgabe 1/85)

Sie verlassen das Programm und kehren in das Inhaltsverzeichnis zurück.

CTRL und *F* (ab Ausgabe 1/86)

Ändert die Farbe des Bildschirm-Hintergrundes (auch im Inhaltsverzeichnis erreichbar).

CTRL und *R* (ab Ausgabe 1/86)

Ändert die Rahmenfarbe (auch im Inhaltsverzeichnis erreichbar).

CTRL und *B* (ab Ausgabe 4/85)

Sie erhalten einen Bildschirmausdruck – natürlich nicht von Grafikseiten oder Sprites! Angapaßt ist diese Hardcopy für Commodore-Drucker und kompatible Geräte. Das Programm wählt automatisch die richtige Geräteadresse (4, 5 oder 6) aus.

Fortsetzung Seite 30

Hinweise für Autoren

Falls Sie uns ein Programm zur Veröffentlichung anbieten wollen, beachten Sie bitte folgende Hinweise: Selbstverständlich können Sie uns Ihr Programm nur anbieten, wenn Sie es selbst erstellt haben und das Programm noch nicht veröffentlicht wurde. Ihr Programm sollte in C-64-BASIC oder in 6502/6510-Assembler geschrieben sein. Als Hilfsmittel können Sie die bisher in INPUT 64 erschienenen Tools (PRINT AT, INKEY, Hiresspeed und die Sprite-Befehle) benutzen, wobei Ihr Programm aber insgesamt nicht länger als 100 Blöcke (25 KByte) sein sollte. Das Programm muß auch ohne Floppy lauffähig sein. Floppy-Betrieb optional ist erlaubt und gewünscht. Es gibt außerdem einige, durch das INPUT 64-Betriebssystem bedingte, programmiertechnische Erfordernisse: 1. Belegen Sie nur den Bereich des normalen BASIC-RAM ($0801-$9FFF) und unter dem BASIC-ROM ($A000-$BFFF). 2. Das Programm muß als BASIC-File zu laden und mit RUN zu starten sein. 3. Die CTRL-Taste darf nicht benutzt werden.

Aber auch wenn Ihr Programm zur Zeit diese Anforderungen nicht erfüllt, sprechen Sie uns ruhig an. Bei ausgefallenen Programmentwicklungen sind wir gerne bereit, bei der Anpassung behilflich zu sein. Senden Sie uns Ihr Programm auf Kassette oder Diskette mit einer Programmbeschreibung und notieren bitte auf allen Einzelteilen Ihren Namen und Ihre Anschrift. Sowohl Auto-Start als auch List-Schutz erschweren uns nur die Arbeit! Wir werden deshalb Programme, deren Analyse absichtlich erschwert wurde, zukünftig ungeprüft zurücksenden.

CTRL und *S* (ab Ausgabe 1/85)

Wenn das Programm zum Sichern vorgesehen ist, erscheinen weitere Hilfsfenster. Sie haben die Wahl, ob Sie:

im Normalverfahren auf Cassette	*C*
im SuperTape-Format	*S*
auf Diskette	*D*

sichern wollen.(Die SuperTape-Option ist ab Ausgabe 1/86 realisiert.) Beachten Sie bitte, daß Sie die Programme von Ihrem Datenträger immer als normale BASIC-Programme mit *LOAD "Name ",1* bzw. *LOAD "Name ",8* laden müssen.

Bei Ladeproblemen:

Schimpfen Sie nicht auf uns, die Bänder sind normgerecht nach dem neuesten technischen Stand aufgezeichnet und sorgfältig geprüft. Sondern: Reinigen Sie zunächst Tonköpfe und Bandführung Ihres Kassettenrecorders. Die genaue Vorgehensweise ist im Handbuch der Datasette beschrieben. Führt auch dies nicht zum Erfolg, ist wahrscheinlich der Tonkopf Ihres Gerätes verstellt. Dieser Fehler tritt leider auch bei fabrikneuen Geräten auf.

Wir haben deshalb ein Programm entwickelt, mit dessen Hilfe Sie den Aufnahme-/Wiedergabekopf justieren können. Tippen Sie das Programm JUSTAGE ein, und speichern Sie es ab. Dieses Programm wertet ein etwa 30 Sekunden langes Synchronisationssignal aus, das sich am Ende jeder Kassettenseite befindet. Starten Sie das JUSTAGE-Programm mit RUN, jetzt sollte die Meldung PRESS PLAY ON TAPE kommen, drücken Sie also die PLAY-Taste. Nach dem Drücken der Taste geht der Bildschirm zunächst wie immer aus. Wird das Synchro-Signal erreicht, wechselt die Bildschirmfarbe; und zwar – bei nicht total verstellter Spurlage – völlig regelmäßig etwa dreimal pro Sekunde. Liegt die Spur des Tonkopfes grob außerhalb der zulässigen Toleranzgrenzen, geschieht entweder nichts oder die Farben wechseln unregelmäßig. Nehmen Sie jetzt einen kleinen Schraubenzieher und werfen Sie einen Blick auf Ihre Datasette. Über der REWIND-Taste befindet sich ein kleines Loch. Wenn Sie bei gedrückter PLAY-Taste durch dieses Loch schauen, sehen Sie den Kopf der Justierschraube für die Spurlage. Drehen Sie diese Einstellschraube. Aber Vorsicht: ganz langsam drehen, ohne dabei Druck auszuüben! Drehen Sie die Schraube nicht mehr als eine Umdrehung in jede Richtung. Nach etwas Ausprobieren wird der Bildschirm gleichmäßig die Farbe wechseln. Zur Feineinstellung lassen Sie das Synchro-Signal noch einmal von Anfang an laufen. Die Schraube jetzt nach links drehen, bis der Farbwechsel unregelmäßig wird. Diese Stellung genau merken, und die Schraube jetzt langsam wieder nach rechts drehen: Der Farbwechsel wird zunächst gleichmäßig, bei weiterem Drehen wieder unregelmäßig. Merken Sie sich auch diese Stellung, und drehen Sie die Schraube nun in Mittelstellung, das heißt zwischen die beiden Randstellungen. Denken Sie daran, daß während der Einstellung kein Druck auf den Schraubenkopf ausgeübt werden darf! Der Tonkopf Ihres Recorders ist jetzt justiert.

Sollte sich auch nach dieser Einstellung INPUT 64 nicht laden lassen, erhalten Sie von uns eine Ersatzkassette. Schicken Sie bitte die defekte Kassette mit einem entsprechenden Vermerk an den Verlag ein (Adresse siehe Impressum).

PS! In der Ausgabe 6/85 haben wir das Programm RECORDER-JUSTAGE veröffentlicht, das die Einstellung des Daten-Recorders zum Kinderspiel macht.

Listing Justage

```
800 fori=49199to49410:read d:ps=ps+d:poke i,d:next
900 ifps<>24716thenprint"falsch abgetippt - fehler korrigieren!":end
950 print"o.k."
970 sys49338
1000 rem von 49199 bis 49410
1010 data173, 13,220,169,217,174,  4,220,172,  5,220,141, 14,220, 48, 44, 56
1020 data102, 88, 36, 89, 48, 12,144, 10,165, 88,133, 90,169,128,133, 88,133
1030 data 91,192,121,144,  4,224,115,176,  7,169,  0,133, 92, 56,176, 11,165
1040 data 92, 73,128,133, 92, 36, 92, 16, 19, 24,102, 88, 36, 89, 48, 12,144
1050 data 10,165, 88,133, 90,169,128,133, 88,133, 91,104,168,104,170,104, 64
1060 data 96, 36, 91, 16,252,132, 91,165, 90, 96,160,128,132, 89,165, 88,201
1070 data 22,208,250,132, 88,160, 10,132, 89,132, 91, 36, 91, 16,252,132, 91
1080 data165, 90,201, 22,208,226,136,208,241, 32,133,192,201, 22,240,249, 96
1090 data 32,147,252,120, 32, 23,248,165,  1, 41, 31,133,  1,133,192,169, 47
1100 data141, 20,  3,169,192,141, 21,  3,169,127,141, 13,220,169,144,141, 13
1110 data220,173, 17,208, 41,239,141, 17,208,169, 70,141,  4,220,169,129,141
1120 data  5,220, 88, 32,142,192,201, 42,208,249,173, 32,208, 41, 15,168,200
1130 data140, 32,208, 76,237,192,208, 76

ready.
```

Am 7. Juli '86 an Ihrem Kiosk: INPUT 64 Ausgabe 7/86

Wir bringen unter anderem:

BUNDESLIGA

Pünktlich zum Saison-Beginn der Fußball-Bundesliga: Eine Sport-Tabellen-Verwaltung nicht nur für Fußball, sondern auch für andere Sportarten. Mit allen erforderlichen Speicher-, Lade-, Druck- und Sortier-Funktionen.

DEVISA

Wieviel griechische Drachmen gibt's denn für 98 DM? Und wieviel Mark muß ich auf den Tisch legen, um 999 schwedische Kronen in der Reisekasse zu haben? Ein Programm zur Umrechnung von über 30 nationalen Währungen – für Daheimgebliebene und Urlaupsplaner.

und außerdem:

Mathe mit Nico, 64er-Tips mit Tricks zum LIST-Befehl. . .

---

c't Magazin für Computertechnik

Ausgabe 7/86 – am 12.6.86 am Kiosk

* Das Betriebssystem des Atari ST, Teil 3: Das hierarchische Dateisystem * EPROM-Programmiergerät für den ST-Userport * Grafikpaket für den C128 unter CP/M * ECB-Adapter für den ZX Spectrum * OCCAM – die Programmiersprache für Transputer * Prüfstand: Videorecorder als Massenspeicher * u.v.a.m

elrad – Magazin für Elektronik

Ausgabe 7-8/86 – ab 30.6. am Kiosk

* Doppelheft 7-8/86 * Sonderteil IC-Magazin mit 19 Schaltungen * Titelgeschichte: Plattenspieler im Selbstbau * Report: 5 Satelliten-Direktempfangsanlagen im Vergleich * Bauanleitung Audio/Studio: Delta-Delay * Die elrad-Laborblätter: Schnittstellen zwischen Computer-Ausgang und Netz * Bauanleitung Labor: Mini-Maxi-Tester * u.v.a.m.

IMPRESSUM:

INPUT 64
Das elektronische Magazin

Verlag Heinz Heise GmbH
Bissendorfer Straße 8
3000 Hannover 61
Postanschrift:
Postfach 61 04 07
3000 Hannover 61
Tel.: (05 11) 53 52-0

Technische Anfragen
nur dienstags von 9.00—16.30 Uhr

Postgiroamt Hannover, Konto-Nr. 93 05-308
(BLZ 250 100 30)
Kreissparkasse Hannover, Konto-Nr. 000-01 99 68
(BLZ 250 502 99)

Herausgeber: Christian Heise

Redaktion:
Christian Persson (Chefredakteur)
Ralph Hülsenbusch
Wolfgang Möhle
Karl-Friedrich Probst
Jürgen Seeger

Ständige Mitarbeiter:
Peter S. Berk
Irene Heinen
Peter Sager
Hajo Schulz
Eckart Steffens

Vertrieb: Anita Kreutzer-Tjaden

Grafische Gestaltung:
Wolfgang Ulber, Dirk Wollschläger

Herstellung: Heiner Niens

Lithografie:
Reprotechnik Hannover

Druck:
Leunisman GmbH, Hannover
CW Niemeyer Hameln

Konfektionierung:
Lettershop Brendler, Hannover

Kassettenherstellung:
SONOPRESS GMBH, Gütersloh

INPUT 64 erscheint monatlich.
Einzelpreis DM 14,80
Jahresabonnement Inland Kassette DM 140,—
Diskette DM 198,—
Diskettenversion im Direktbezug: DM 16,80
+ DM 3,— Porto und Verpackung

Redaktion, Anzeigenverwaltung, Abonnementsverwaltung:
Verlag Heinz Heise GmbH
Postfach 61 04 07
3000 Hannover 61
Tel.: (05 11) 53 52-0

Abonnementsverwaltung Österreich:
Evb-Verlag GmbH & Co KG
Abt. Zeitschriftenvertrieb
z. Hd. Frau Pekatschek
Amerlingstr. 1
A-1061 Wien
Jahresabonnement: Kassette DM 152,—
Diskette DM 210,—

Vertrieb (auch für Österreich, Niederlande, Luxemburg und Schweiz):
Verlagsunion Zeitschriften-Vertrieb
Postfach 57 07
D-6200 Wiesbaden
Ruf (0 61 21) 2 66-0

Verantwortlich:
Christian Persson
Bissendorfer Straße 8
3000 Hannover 61

ISSN 0177-3771

Titelidee **INPUT 64**
Titelfoto: Bavaria

Titel-Grafik und -Musik:
Tim Pritlove
Fabian Rosenschein

Betriebssystem:
Hajo Schulz

hier abtrennen

INPUT 64-Abonnement Abruf-Coupon

Ja, übersenden Sie mir bis auf Widerruf alle künftigen INPUT64-Ausgaben ab Monat

_____

(Kündigung ist jederzeit mit Wirkung ab der jeweils übernächsten Ausgabe möglich. Überzahlte Abonnementsgebühren werden sofort anteilig erstattet.)

Das Jahresabonnement kostet: ○ auf Kassette DM 140,— inkl. Versandkosten und MwSt.

○ auf Diskette DM 198,— inkl. Versandkosten und MwSt.

(Bitte ankreuzen/Nichtzutreffendes streichen.)

Absender und Lieferanschrift

Bitte in jedes Feld nur einen Druckbuchstaben (ä = ae, ö = oe, ü = ue)

Vorname/Zuname

Beruf/Funktion

Straße/Nr.

PLZ Wohnort

_____

Datum/Unterschrift

Von meinem Recht zum schriftlichen Widerruf dieser Order innerhalb einer Woche habe ich Kenntnis genommen. Zur Wahrung der Frist genügt die rechtzeitige Absendung.

Unterschrift _____

Bitte beachten Sie, daß diese Bestellung nur dann bearbeitet werden kann, wenn beide Unterschriften eingetragen sind.

INPUT 64-Abonnement Abruf-Coupon

Ich wünsche Abbuchung der Abonnement-Gebühr von meinem nachstehenden Konto. Die Ermächtigung zum Einzug erteile ich hiermit.

Name des Kontoinhabers

Bankleitzahl Konto-Nr.

Geldinstitut Ort des Geldinstituts

Bankeinzug kann nur innerhalb Deutschlands und nur von einem Giro- oder Postscheckkonto erfolgen.

HEISE

Bitte im (Fenster-)Briefumschlag einsenden.
Nicht als Postkarte verwenden!

INPUT64

Vertriebsabteilung
Verlag Heinz Heise GmbH
Postfach 61 04 07

3000 Hannover 61