wie »G« (also GO). Es startet ein Maschinenprogramm an der als Argument angegebenen Adresse. Die Syntax ist ebenfalls die gleiche wie die von »G«, also startet beispielsweise »J $01300« ein in Bank 0 bei Adresse $1300 liegendes Maschinenprogramm.

## MERGE für den C 128

Außer dem OLD-Befehl vermißt mancher Benutzer des C 128 noch eine Möglichkeit, Basic-Programme zu verbinden, die durch einen MERGE-Befehl gegeben wird. Das Prinzip, das mit einem MERGE realisiert werden kann, unterstützt die strukturierte Programmierung. Es ist nämlich nun möglich, Programm-Module zu erstellen und zu speichern, die in sinnvoller Kombination zum Gesamtwerk verknüpft werden können.

Das hier abgedruckte Programm MERGE (Listing) können Sie wieder direkt mit dem Monitor eingeben. Es belegt den Adreßbereich von $01352 bis $013DD. Das Programm besteht aus zwei Teilen, zu denen der Inhalt der Speicherstelle $FA führt. Ist dieser Inhalt gleich Null, dann wurde MERGE zum ersten Mal aufgerufen. Der Vektor $2D/2E wird zwischengespeichert, vom Basic-Textende-Vektor $1210/1 zwei abgezogen und das Ergebnis nunmehr in den Textanfang-Vektor $2D/E eingetragen. Schließlich schreibt das Programm eine Kennzahl (hier $85) in die Flagge $FA und meldet sich mit einem Text:
»ANSCHLUSS-PROGRAMM EINLADEN (ZEILENNUMMERN OK?)«

Es besteht nun die Möglichkeit, ein weiteres Modul zu laden oder einzutippen. Außerdem können durch RENUMBER Zeilennummern hochgelegt werden, damit sie nicht in Konflikte mit dem ersten Programmteil geraten.

Ein erneuter Aufruf von MERGE verzweigt nach der Prüfung der Flagge nun in den anderen Programmteil, der lediglich den ursprünglichen Wert des Vektors $2D/E restauriert und in die Flagge wieder den Wert Null einträgt. Abschließend wird die Meldung
»DIE PROGRAMME SIND JETZT VERBUNDEN«
ausgegeben. Damit sind beide Programmteile nun zu einem kombiniert und können durch ein erneutes RENUMBER auch von den Zeilennummern her in eine ansprechende Form gebracht werden. Der Aufruf von MERGE geschieht durch

```
BANK15:SYS DEC(''1352'')
```

MERGE kann beliebig oft angewendet werden, auch dann, wenn der Basic-Start (beispielsweise durch Einrichten eines Grafikbildschirmes) verschoben wurde.

**PRIMM**

Für Assembler-Programmierer interessant sein dürfte ein neuer Kernel-Sprungvektor, der im Programm MERGE verwendet wurde: PRIMM heißt er und wird mittels »JSR $FF7D« aufgerufen. PRIMM druckt einen Text auf den Bildschirm, der direkt nach diesem Aufruf folgt (PRint IMMediate). Das Ende der Textsequenz ist durch ein Nullbyte zu kennzeichnen. Nach dem Ausdruck wird das Programm mit dem Befehl fortgesetzt, der auf das Null-Byte folgt.

Damit endet der zweite Zwischenbericht unseres Forschungsreisenden.

(Heimo Ponnath/ev)

```
MONITOR
   PC  SR AC XR YR SP
; FB00 00 00 00 00 F8

>01352 A5 FA C9 85 B0 51 A5 2D 85 FB A5 2E 85 FC 38 AD:
>01362 10 12 E9 02 85 2D AD 11 12 E9 00 85 2E A9 85 85:
>01372 FA 20 7D FF 0D 41 4E 53 43 48 4C 55 53 53 2D 50:
>01382 52 4F 47 52 41 4D 4D 0D 45 49 4E 4C 41 44 45 4E:
>01392 0D 28 5A 45 49 4C 45 4E 4E 55 4D 4D 45 52 4E 20:
>013A2 4F 4B 3F 29 0D 00 60 A5 FB 85 2D A5 FC 85 2E A9:
>013B2 00 85 FA 20 7D FF 0D 44 49 45 20 50 52 4F 47 52:
>013C2 41 4D 4D 45 20 53 49 4E 44 0D 4A 45 54 5A 54 20:
>013D2 56 45 52 42 55 4E 44 45 4E 0D 00 60 BF 00 BF 00:
```

**Listing »Merge«. Bitte mit dem Maschinensprache-Monitor des C 128 eingeben.**

# Tips und Tricks zum C 128

**Wir haben Ihnen wieder neue und nützliche Tips, Tricks und Befehle zu bieten. Einen kleinen Kurs, OLD- und Spritebefehle sowie eine bessere INPUT-Routine.**

Wieder einmal bieten wir Ihnen kleine Kniffe und Routinen, die Ihnen helfen werden, leichter mit Ihrem C 128 umzugehen.

## Die Floppy 1571 und Originalprogramme

Es häufen sich die Anfragen von Lesern, wonach sich ein Teil ihrer gekauften Programme nicht in den Computer laden lassen. Diesen Lesern möchten wir nun Antwort geben. Die in den Floppylaufwerken VC 1541 und VC 1571 eingebauten DOS-Versionen (Floppy-Betriebssystem) sind untereinander leider nicht voll kompatibel. Originalprogramme sind jedoch in der Regel mit einem Kopierschutz versehen. Wenn dieser Schutz einer der besseren Art ist, so benutzt er das Floppy-DOS für seine Routinen. Diese Routinen sind jedoch auf das DOS der VC 1541 zugeschnitten. Versucht der Kopierschutz nun, eine Routine in der VC 1571 auszuführen, die von der der VC 1541 abweicht, so kann die Kopierschutzabfrage nicht mehr richtig ausgeführt werden und das Programm läßt sich nicht weiter laden. Dagegen ist leider kein Kraut gewachsen. Hier sind die Benutzer von Raubkopien leider in der besseren Lage, da aus einem »gecrackten« Programm eben diese Abfragen entfernt wurden. (dm)

## Die Variablen-Behandlung beim C 128

Das Format der Integer- und Gleitkommavariablen im C 128-Modus ist identisch mit denen im C 64-Modus. Nur bei den Stringvariablen hat sich das Format geändert. Es wird genau wie beim C 64 ein Stringdeskriptor angelegt, der die Adresse, an der der String tatsächlich abgelegt wurde und die Länge des Strings beinhaltet. An den Inhalt des Strings, also an seine ASCII-Zeichen, wird wie beim CBM 8032 ein Zwei-Byte-Zeiger angehängt. Dieser zeigt auf das Längenbyte im Stringdeskriptor. Zwar wird dadurch mehr Platz im Speicher benötigt, der Vorteil ist jedoch unschätzbar — die berühmt gefürchtete »Garbage Collection« schlägt nicht mehr in dem vom C 64 gewohnten Ausmaß zu. Es wird nur noch höchstens eine Sekunde zur »Stringmüllbeseitigung« gebraucht.

## Die Variablenformate

In der folgenden Aufstellung steht »NAME« für den ASC-Wert eines Zeichens.

**Integer-Variable**

Die ganzzahligen Integer-Variablen werden in zwei Speicherzellen mit dem höchstwertigen Byte (MSB) an erster Stelle abgelegt. Es sind vorzeichenbehaftete Zahlen im Bereich von -32768 bis +32767. Ist das höchstwertige Bit in einer Zahl gesetzt, so ist sie negativ. Im Commodore-Basic werden Integer-Variablen durch ein nachgestelltes Prozentzeichen dargestellt (zum Beispiel AA% oder Z%). Für die interne Kennzeichnung werden in den beiden Namen-Bytes die höchstwertigen Bits gesetzt.

**Einzelvariable**

Jede Einzelvariable belegt 7 Byte. Den Aufbau ersehen Sie aus Bild 1:

**Felder**

Ein Feld belegt 5 + 2 x Dimension + (Dimension n x Dimension (n-1) x Dimension (n-2)...) x 2 Byte.

**Aufbau des Deskriptorblockes**

1. Byte Name + 128
2. Byte Name + 128
3. Byte Gesamtlänge Array (Lowbyte)
4. Byte Gesamtlänge Array (Highbyte)
5. Byte Anzahl der Dimensionen
6. Byte Anzahl Elemente Dimension n, Low
7. Byte Anzahl Elemente Dimension n, High
8. Byte Anzahl Elemente Dimension n-1, Low
9. Byte Anzahl Elemente Dimension n-1, High

und so weiter für die restlichen Dimensionen.

Die Feldelemente werden direkt hinter dem Deskriptorblock in je 2 Byte (MSB-LSB) gespeichert (siehe Bild 2).

Bei mehrdimensionalen Feldern wird die letzte Dimension als erste angelegt.

**Gleitkomma-Variable**

Die Gleitkommazahlen sind in 5 Byte abgelegt. Im ersten Byte wird der Exponent gespeichert. Die Mantisse liegt in den folgenden 4 Byte als 32-Bit vorzeichenbehaftete Zahl. Die Gleitkommazahlen werden im Commodore-Basic nur durch die ASC-Werte der ersten beiden Zeichen dargestellt (zum Beispiel AA oder B).

Intern wird die Gleitkommazahl mit ihrem Namen in 2 Byte mit den ASC-Werten gekennzeichnet. Besteht der Name nur aus einem Zeichen, so lautet das zweite Byte Null.

**Einzelvariable**

Jede Einzelvariable belegt 7 Byte, wie Sie aus Bild 3 ersehen können.

**Felder**

Ein Feld belegt 5 + 2 x Dimension + (Dimension n x Dimension (n-1) x Dimension (n-2)...) x 2 Byte.

**Aufbau des Deskriptorblockes**

1. Byte Name + 128
2. Byte Name + 128
3. Byte Gesamtlänge Array (Lowbyte)
4. Byte Gesamtlänge Array (Highbyte)
5. Byte Anzahl der Dimensionen
6. Byte Anzahl Elemente Dimension n, Low
7. Byte Anzahl Elemente Dimension n, High
8. Byte Anzahl Elemente Dimension n-1, Low
9. Byte Anzahl Elemente Dimension n-1, High

und so weiter für die restlichen Dimensionen.

Die Feldelemente werden direkt hinter dem Deskriptorblock in je fünf Speicherzellen gespeichert. (Siehe Bild 4).

Bei mehrdimensionalen Feldern wird die letzte Dimension als erste angelegt.

**Stringvariable**

Stringvariablen werden in zwei Teilen gespeichert. In einem Deskriptorblock finden sich die für die Verwaltung notwendigen Angaben. Im oberen RAM-Bereich wird der String selbst als Folge von ASC-Werten abgelegt. Im Deskriptorblock zeigt ein Zeiger auf eine Adresse im oberen RAM-Bereich, an der der String abgelegt ist. Jeder String wird von einem Zwei-Byte-Zeiger, der auf das Längen-Byte im Deskriptor zeigt, abgeschlossen.

**Aufbau der ASCII-Zeichenformate**

Text-String Zeiger
Lowbyte Highbyte

Im Commodore-Basic werden Stringvariable mit einem nachgestellten Dollarzeichen dargestellt (zum Beispiel A$ oder AB$). Intern werden Strings mit 2 Byte gekennzeichnet, wobei das höchstwertige Bit im zweiten Namen-Byte gesetzt ist. Besteht der Name nur aus einem Byte, so hat das zweite Namen-Byte den ASC-Wert 128.

**Einzelvariable**

Jede Einzelvariable belegt 7 Byte und im oberen RAM-Bereich zwei Byte + Länge des Strings. Der Aufbau wird aus Bild 5 ersichtlich.

| Name 1 + 126 | Name 2 + 128 | MSB | LSB | 00 | 00 | 00 |
|---|---|---|---|---|---|---|

**Bild 1. Aufbau einer Integer-Variable**

| MSB 1 | LSB 1 | MSB 2 | LSB 2 | etc. |
|---|---|---|---|---|

**Bild 2. So liegen die Feldelemente hintereinander**

| Name 1 | Name 2 | Exponent + 128 | Mantisse 32-Bit |
|---|---|---|---|

**Bild 3. Aufbau einer Gleitkomma-Variable**

| Expon 1 | Mantisse 1 | Expon 2 | Mantisse 2 | usw. |
|---|---|---|---|---|

**Bild 4. Und auf diese Weise liegen die Feldelemente im Speicher**

| Name 1 | Name 2 + 128 | String-länge | Zeiger in RAM low | high | 00 | 00 |
|---|---|---|---|---|---|---|

**Bild 5. Aufbau einer einzelnen Stringvariable**

| String 1 länge | Zeiger 1 low/high | String 2 länge | Zeiger 2 low/high | usw. |
|---|---|---|---|---|

**Bild 6. Lage der Feldelemente hinter dem Deskriptorblock**

| Name 1 + 128 | Name 2 | String-länge | Zeiger in RAM low | high | 00 | 00 |
|---|---|---|---|---|---|---|

**Bild 7. Anordnung einer Funktionsvariablen**

| Label | Adresse Dez | Adresse Hex | Funktion |
|---|---|---|---|
| TXTTAB | 45 | 002D | Zeiger auf Start BASIC B0 |
| VARTAB | 47 | 002F | Zeiger auf Start Variablen B1 |
| ARYTAB | 49 | 0031 | Zeiger auf Start Variablenfelder B1 |
| STREND | 51 | 0033 | Zeiger auf Ende der Variablenfelder + 1 B1 |
| FRETOP | 53 | 0035 | Zeiger auf Start String B1 |
| FRESPC | 55 | 0037 | Hilfszeiger für Stringverwaltung |
| MAXME1 | 57 | 0039 | Höchste verfügbare Variablenadresse B1 |
| BASTOP | 4624 | 1210 | Zeiger auf BASIC-Text Ende B0 |
| MAXME0 | 4626 | 1212 | Höchste verfügbare BASIC-Text-Adresse B0 |

**Tabelle 1. Wichtige Adressen (B0/B1 entspricht Bank 0/1)**

**Felder**

Jedes Feld belegt 5 + 2 x Dimension-Byte + (Dimension n x Dimension (n-1) x Dimension (n-2)...) x 3 und im oberen RAM-Bereich (Länge aller Strings + (2 x Anzahl aller Strings)). Es müssen nur Strings, deren Länge größer Null ist, berücksichtigt werden.

**Aufbau des Deskriptorblockes**

1. Byte Name + 128
2. Byte Name + 128
3. Byte Gesamtlänge Array (Lowbyte)
4. Byte Gesamtlänge Array (Highbyte)
5. Byte Anzahl der Dimensionen
6. Byte Anzahl Elemente Dimension n, Low
7. Byte Anzahl Elemente Dimension n, High
8. Byte Anzahl Elemente Dimension n-1, Low
9. Byte Anzahl Elemente Dimension n-1, High

und so weiter für die restlichen Dimensionen.

Die Feldelemente werden direkt hinter dem Deskriptorblock in je 3 Byte gespeichert (Bild 6).

Bei mehrdimensionalen Feldern wird die letzte Dimension als erste angelegt.

**Funktionsvariable**

Funktionen, die man mit der Anweisung
DEF FN name(var) = ausdr.
definiert, werden von Basic ebenfalls als Variable mit dem Namen »name« im RAM abgelegt.

Intern wird das höchstwertige Bit im ersten Namen-Byte gesetzt. Hat der Variablenname nur ein Zeichen, so ist das zweite Namen-Byte Null. Im Gegensatz zu den anderen Variablen gibt es aber hier keine Felder.

Jede Funktionsvariable belegt 7 Byte und im oberen RAM-Bereich 2 + Länge der Funktionsvariablen. Ansonsten entspricht die Ablage der einen einfachen Stringvariablen (Bild 7).

Damit sind alle Variablentypen, die der C 128 kennt, dargestellt.

Im Normalzustand werden die Variablen in Bank 1 ab Adresse $0400 (dezimal 1024) aufwärts und die Inhalte der Strings und Funktionsvariablen ab Adresse $FEFF (dezimal 65279) abwärts gelegt.

In Tabelle 1 sind die Adressen einiger Zeiger aus der erweiterten Zeropage, die für die Verwaltung der Variablen wichtig sind, aufgeführt. Die aktuellen Adressen in der jeweiligen Bank kann man leicht nach folgender Formel berechnen:
Adresse = PEEK(Zeigeradresse) +
PEEK(Zeigeradresse + 1) * 256

Doch Achtung: Zum Nachschauen muß man dann noch die richtige Bank angeben.

(Michael Bauer/dm)

## Anti-C 128-POKE

Wer seinen C 128 häufiger im C 64-Modus benutzt, wird feststellen, daß dies der Commodore-Taste nicht unbedingt gut tut. Und das Umschalten mit GO 64 wird mit der Zeit auch lästig.

Gibt man im C 128-Modus folgende Zeile im Direkt- oder Programmodus ein, so springt der C 128 in den C 64-Modus und ist auch durch einen Reset nicht mehr in den C 128-Modus zu bringen.

```
BANK1:POKE 65528,77:POKE 65529,255:GO 64
```

(M. Güthling/dm)

## Neue Befehle und Tricks für den C 128

Das Handbuch des Commodore 128 ist zwar etwas besser geraten als beim C 64, trotzdem lassen sich noch einige Fehler finden. So wurde beschrieben, daß die Umschaltung auf die DIN-Tastatur mit POKE 1,PEEK(1) AND 191 im Programm erfolgen kann. Vor diesem POKE ist aber noch das Datenrichtungsregister auf Ausgabe zu setzen. Dies erfolgt mit POKE 0, PEEK (0) OR 64. Mit POKE 1, PEEK(1) OR 64 wird die ASCII-Tastatur wieder aktiviert. Weiterhin wurde behauptet, daß mit POKE 2757,129 die Umschaltmöglichkeit auf die DIN-Tastatur verhindert werden kann und nur ein Druck auf den Reset-Knopf dies rückgängig macht. POKE 2757,0 tut es auch.

ESC 0 schaltet den Einfüge-, Anführungs- und Invers-Modus aus. Wenn Sie die ESC-Taste zweimal kurz hintereinander drücken, erreichen Sie den gleichen Effekt, nur schneller.

Der USR-Vektor wurde nur für den C 64-Modus beschrieben und das auch noch falsch. Im C 64-Modus liegt der USR-Vektor an den Speicherstellen 785 und 786 und im C 128-Modus bei 4633 und 4634.

Im C 128-Modus hat der Basic-Interpreter eine Verbesserung aufzuweisen. Während beim C 64 die Abfrage auf einen Leerstring beim ASC-Befehl die Fehlermeldung ILLEGAL QUANTITY hervorruft, erhalten Sie beim C 128 als Ergebnis den Wert 0.

Um Zahlen in andere Zahlensysteme umzurechnen, gibt es beim C 128 die Befehle DEC und HEX$. Mit dem eingebauten Maschinensprachenmonitor haben Sie eine sehr komforta-

```
MONITOR
    PC  SR AC XR YR SP
; B000 00 00 00 00 F8

.01300  78       SEI
.01301  85 FA    STA $FA
.01303  85 FB    STA $FB
.01305  A9 11    LDA #$11
.01307  8D 14 03 STA $0314
.0130A  A9 13    LDA #$13
.0130C  8D 15 03 STA $0315
.0130F  58       CLI
.01310  60       RTS
.01311  C6 FA    DEC $FA
.01313  A5 FA    LDA $FA
.01315  D0 0C    BNE $1323
.01317  A9 00    LDA #$00
.01319  8D FD 12 STA $12FD
.0131C  A5 FB    LDA $FB
.0131E  85 FA    STA $FA
.01320  4C 65 FA JMP $FA65
.01323  A9 01    LDA #$01
.01325  8D FD 12 STA $12FD
.01328  4C 65 FA JMP $FA65
```

**Listing 1. Das Monitorlisting der Routine »Spriteslow«**

ble Möglichkeit zur Zahlenumrechnung. Sie rufen mit dem Befehl MONITOR den Maschinensprachenmonitor auf und geben die umzurechnende Zahl mit vorangestelltem Umrechnungssymbol ein ($ = hexadezimal, + = dezimal, & = oktal und % = binär). Wenn Sie nun die RETURN-Taste drücken, so wird die Zahl in allen vier Zahlensystemen ausgegeben.

Zum Spritehandling gibt es beim C 128 komfortable Befehle. So können Sprites unabhängig vom laufenden Programm bewegt werden. Dieses geschieht in der Interruptroutine. In der Speicherstelle 4861 steht ein Flag, welches angibt, ob die Interruptroutine für die Spritesteuerung ausgeführt werden soll. Wenn Sie in diese Speicherstelle einen Wert ungleich Null schreiben, so bleiben alle Sprites sofort stehen. Aber Achtung! Der PLAY-Befehl funktioniert dann nicht mehr. Mit POKE 4861,0 bewegen sich die Sprites weiter. Für die Geschwindigkeit können beim MOVSPR-Befehl Werte bis 15 angegeben werden. Wenn Sie aber direkt die Register für die Spritegeschwindigkeit mit POKE ansprechen, so sind auch höhere Geschwindigkeiten möglich. Errechnen läßt sich die Speicherstelle durch 4478+(SN-1)*11. Hierbei steht SN für die Spritenummer (1 bis 8). Wollen Sie die Spritegeschwindigkeit drosseln, so können Sie dies mit dem Programm SPRITESLOW (Listing 1) tun. Mit

```
BANK 15: SYS DEC("1300"),X
```

wird das Programm aufgerufen. Für X setzen Sie den Wert der Verzögerung ein. Die Spritegeschwindigkeit, die mit dem Befehl MOVSPR eingegeben wurde, sollte aber nicht größer als 1 sein. Experimentieren Sie ruhig ein bißchen!

Kommen wir gleich zum zweiten Maschinenprogramm (Listing 2). Dies Programm rekonstruiert ein mit NEW gelöschtes Programm. Solange noch keine neuen Programmzeilen eingegeben wurden, kann mit

```
BANK 0: SYS DEC("1300")
```

ein mit NEW oder nach einem RESET gelöschtes Programm gerettet werden.

Das Basic V 7.0 wurde zwar um etliche Befehle erweitert, trotzdem gibt es immer noch keinen vernünftigen INPUT-Befehl. Die Eingaberoutine aus Listing 3 können Sie als Unterprogramm in Ihren eigenen Programmen benutzen.

Es erlaubt, nur bestimmte Zeichen einzugeben. Außerdem kann eine maximale Eingabelänge festgelegt werden. Zu diesem Programm nun noch einige Erklärungen:

In Zeile 45 werden die erlaubten Zeichen der Stringvariablen EZ$ übergeben. In Zeile 50 wird in der Variablen Q1 die maximale Eingabelänge festgelegt und das Unterprogramm mit GOSUB 100 aufgerufen. Nach der Rückkehr aus dem Unterprogramm steht die Eingabe in der Stringvariablen Y2$. In der Eingaberoutine wird in Zeile 110 der aktuelle Cursormodus der Variablen Q9 übergeben und mit POKE 2598,0 der blinkende Cursormodus eingeschaltet. POKE 2599,0 läßt den Cursor auch bei dem Befehl GET oder GETKEY blinken. Zeile 150 prüft auf die maximale Eingabelänge und Zeile 160 auf die erlaubten Zeichen. Wird versucht, unerlaubte oder zuviele Zeichen einzugeben, so wird dies durch ein akustisches Signal angezeigt. Zeile 180 wartet darauf, daß der Cursor sich nicht in der Blinkphase befindet. Wurde Return gedrückt, so wird mit POKE 2599,1 das Cursorblinken bei GET wieder ausgeschaltet und der vorher in Q9 festgehaltene Cursormodus in Speicherstelle 2598 eingetragen. (Herbert Kunz/dm)

## Eine Grafik-Spielerei

Mit dieser kleinen Routine können Sie bezaubernde Grafik-Bilder erzeugen. Durch Verändern des Wertes X können die Steigungen der Kurven variiert werden. Ebenfalls kann man einen verkleinernden oder vergrößernden Faktor wählen (Zeile 30).

```
10 GRAPHIC 1,1
20 FOR A = 200 TO 0 STEP-3
30 B=B+3
40 CIRCLE, 160, 100, A, A, 0, 0, B, X
50 NEXT
```

(Guido Adolphs/dm)

```
MONITOR
    PC  SR AC XR YR SP
; B000 00 00 00 00 F8

.01300  A5 2D     LDA $2D
.01302  A4 2E     LDY $2E
.01304  85 FA     STA $FA
.01306  84 FB     STY $FB
.01308  A0 03     LDY #$03
.0130A  C8        INY
.0130B  B1 FA     LDA ($FA),Y
.0130D  D0 FB     BNE $130A
.0130F  C8        INY
.01310  98        TYA
.01311  18        CLC
.01312  65 FA     ADC $FA
.01314  A0 00     LDY #$00
.01316  91 2D     STA ($2D),Y
.01318  A5 FB     LDA $FB
.0131A  69 00     ADC #$00
.0131C  C8        INY
.0131D  91 2D     STA ($2D),Y
.0131F  A5 2D     LDA $2D
.01321  85 FA     STA $FA
.01323  A5 2E     LDA $2E
.01325  85 FB     STA $FB
.01327  A0 01     LDY #$01
.01329  B1 FA     LDA ($FA),Y
.0132B  F0 0B     BEQ $1338
.0132D  AA        TAX
.0132E  88        DEY
.0132F  B1 FA     LDA ($FA),Y
.01331  85 FA     STA $FA
.01333  86 FB     STX $FB
.01335  4C 27 13  JMP $1327
.01338  A5 FA     LDA $FA
.0133A  18        CLC
.0133B  69 02     ADC #$02
.0133D  8D 10 12  STA $1210
.01340  90 02     BCC $1344
.01342  E6 FB     INC $FB
.01344  A5 FB     LDA $FB
.01346  8D 11 12  STA $1211
.01349  60        RTS
```

**Listing 2. Monitor-Auszug des neuen OLD-Befehls**

```
10 :REM  EINGABEROUTINE MIT GETKEY
20 :
30 :
40 :REM BEISPIEL EINER EINGABE
45 EZ$="1234567890+-."
50 PRINT"EINGABE : ";:Q1=5:GOSUB100:EG$=Y2$:END
60 :
100 REM  BEGINN DER EINGABEROUTINE
110 Q9=PEEK(2598):POKE2598,0:Q4=0:Y2$="":POKE2599,0
120 GETKEYY1$
130 IFY1$=CHR$(13)THEN220
140 IFY1$=CHR$(20)THEN270
150 IFQ4=Q1THENPRINTCHR$(7);:GOTO120
160 Q5=INSTR(EZ$,Y1$):IFQ5=0THENPRINTCHR$(7);:GOTO120
170 POKE2600,2
180 IFPEEK(2598)THEN180
190 PRINTY1$;:Y2$=Y2$+Y1$:Q4=Q4+1:GOTO120
200 :
210 : REM RETURN GEDRUECKT
220 POKE2600,2
230 IFPEEK(2598)THEN230
240 POKE2599,1:POKE2598,Q9:RETURN
250 :
260 : REM DEL TASTE GEDRUECKT
270 IFQ4=0THENPRINTCHR$(7);:GOTO120
280 POKE2600,2
290 IFPEEK(2598)THEN290
300 PRINTCHR$(20);:Q4=Q4-1:Y2$=LEFT$(Y2$,Q4):GOTO120
```

**Listing 3. Eine verbesserte Eingaberoutine für den C 128**