

Alle Tasten-, Zeichen- und Steuer-codes

Die gute und übersichtliche Tastatur der Computer von Commodore bietet gerade dem Anfänger sehr leichte Einstiegsmöglichkeiten. Wer aber tiefer in die Computerei eindringt, stößt beim VC 20 und Commodore 64 schnell auf eine verwirrende Vielfalt von Codes und Steuerzeichen, deren Kenntnis und Beherrschung jedoch auch dem Anfänger ein weites Anwendungsfeld eröffnen kann.

Haben Sie schon einmal einen Ball länger als ein paar Minuten auf derselben Stelle liegen sehen? Ich noch nicht, denn der nächste Vorbeikommer kickt ihn – irgendwohin. Sie doch auch, oder? Dasselbe Phänomen können Sie überall dort beobachten, wo ein Heimcomputer steht. Eingeschaltet oder nicht – jeder der vorbeikommt, drückt auf eine Taste. Und wenn das Ding gar reagiert – mit einem Buchstaben auf dem Bildschirm – dann bleibt selbst die Oma stehen und tippt nochmal und nochmal.

Die Tasten wirken auf die Menschen wie das Licht auf die Motten

Wenn vom VC 20 oder Commodore 64 die Rede ist, kommt dieselbe sehr rasch auf die Tastatur. Ihre Schreibmaschinen-Qualität und die klaren Steuer- und Funktionstasten heben sie aus der Schar der Konkurrenten heraus. Sie ist sehr benutzerfreundlich, besonders für Anfänger, während der ersten Tast(en)versuche.

Aber rasch wird offensichtlich, daß die Tasten auch ihre Geheimnisse haben, und zwar frühestens, wenn die vier Funktionstasten keine Reaktionen, zumindest keine sichtbaren, hervorrufen, und spätestens bei dem Versuch, in einem Programmablauf bestimmte Tastenfunktionen ausführen zu lassen, wie zum Beispiel »Cursor Down« oder »Bildschirm löschen«.

Ich will Ihnen deshalb in diesem Aufsatz die Grundlagen der Tastaturabfrage, das Geheimnis der Funktionstasten, die CHR\$-Anhänge von PRINT-Befehlen, die mysteriösen »negativen« Zeichen in Anführungsstrichen und gleichzeitig gedrückte Tasten erklären und mit Kochrezepten ausschmücken.

Übrigens: das hier Gesagte gilt sowohl für den VC 20 als auch für den Commodore 64. Nur die Programmbeispiele nicht, die sind auf den kleinen VC 20 zugeschnitten. Die 64er mögen mir das verzeihen. Ich scheue mich aber, Kochrezepte abzugeben, die ich nicht selbst ausprobiert habe. Und ich habe leider (noch) keinen C 64. Falls Unterschiede bei den Adressen auftreten, gebe ich den Wert für den C 64 in Klammern an. Als erstes möchte ich folgende Behauptung aufstellen:

Dem Computer sind alle Tasten gleich.

Und er behandelt sie auch alle gleich. 60mal in jeder Sekunde unterbricht der Computer was er auch immer gerade ausführt, merkt sich, wobei und wo er gerade unterbrochen hat, und schaut nach, ob eine der Tasten gedrückt worden ist.

Keine Regel ohne Ausnahme, auch beim Computern nicht: Die RESTORE-Taste und die SHIFT-LOCK-Taste fallen völlig aus dem Rahmen und haben mit den kommenden Erklärungen nichts zu tun. Ich werde sie deshalb auch nicht mehr erwähnen. Der Vollständigkeit halber sei hier nur kurz gesagt, daß die RESTORE-Taste den Computer bei jeglicher Arbeit unterbricht und ihn mit READY und blinkendem Cursor in den Anfangszustand zurücksetzt. Die SHIFT-LOCK-Taste ist (wie bei der Schreibmaschine) eine mechanische Arretierung der SHIFT-Taste. Es bleiben uns immerhin 64 Tasten, die vom Computer während seiner Verschnaufpause inspiziert werden. Die Funktionstasten sind immer dabei! Diese Inspektion – Tastaturabfrage genannt – wollen wir uns näher anschauen.

Der Computer erhält nicht, wie es eigentlich logisch wäre, von jeder gedrückten Taste ein spezielles Code-Signal. Das wäre für einen Heimcomputer zu aufwendig und zu teuer. Das Betriebssystem des Computers veranstaltet vielmehr eine Befragung seines Tastenvolkes, nach dessen Stimmenabgabe er dann entscheidet, welche Taste nun eigentlich gedrückt worden ist. Schauen Sie sich bitte das Bild 1 genauer an. Es zeigt Ihnen die 64 Tasten in einer 8 x 8-Matrix. Diese Anordnung entspricht der elektrischen Verbindung der Tasten. Die Abfrage selbst ist sehr einfach. Der Computer ruft alle senkrechten Spalten einzeln auf, indem er die Zahl, die an der jeweiligen Spalte in Bild 1 steht, in eine spezielle Speicherzelle hineinschiebt. Beim VC 20 ist das 37152, beim Commodore 64 die 56320.

Falls die gedrückte Taste in dieser Spalte liegt, meldet sie sich und die Zahl der waagrechten Zeile wird in die Adresse 37153 (56321) geschrieben. Wenn überhaupt keine Taste gedrückt war, erscheint in 37153 (56321) eine 255.

10 IF Sie jetzt fragen, warum die Zahlen so willkürlich und außerdem paarweise gleich sind, THEN lesen Sie weiter: GOTO nächsten Absatz.

20 REM ein bißchen Theorie würde nicht schaden: GOTO übernächsten Absatz.

Die Zahlen sind natürlich nicht willkürlich gewählt. Es sind vielmehr die Dezimalwerte von Dualzahlen, die beim Anwählen einer Spalte beziehungsweise durch Drücken einer Taste in den schon genannten Speicherzellen 37152 (56320) und 37153 (56321) entstehen. Bild 2 zeigt den Zusammenhang.

Am Rande der Matrix sehen Sie jetzt nicht die Zahlen, sondern die beiden Speicherzellen, im folgenden »Register« genannt, und ihre bitweise Verbindung mit Spalten und Zeilen der Tasten. Prinzipiell gilt, daß diejenige Spalte angewählt ist, an deren Stelle eine »Null« im Register 37152 (56320) steht.

Theorie und Praxis: eine untrennbare Einheit

Das ergibt im Register eine Dualzahl, die entsprechend der Bit-Numerierung zu lesen ist. Eine gedrückte Taste ihrerseits erzeugt eine Null im Register 37153 (56321) an der Stelle, wo ihre Zeile angeschlossen ist. Ich habe im Bild 2 ein Beispiel (Taste mit den Zeichen »«) eingezeichnet, und wenn Sie die Dualzahlen kennen, werden Sie beim Vergleich mit den Zahlen in Bild 1 die Übereinstimmung erkennen.

Wie bitte? Sie sagen, daß es aber doch möglich sein müßte, weit mehr als die im Bild 1 gezeigten acht Zahlen im Register 37153 (56321) zu erzeugen, indem man mehr als eine Taste gleichzeitig drückt? Richtig gesehen, das geht in der Tat, und

das wollen wir uns auch gleich einmal ansehen. Dazu brauchen wir ein Kochrezept. Ich schlage vor, das Rezept – und auch die noch folgenden – gleich auszuprobieren. Nehmen Sie diesen Artikel und verfolgen Sie die weiteren Zeilen meiner Beschreibung abwechselnd lesend und eintippend.

Die Abfrage der Tastatur wird auf dem Bildschirm dargestellt

Als erstes wollen wir, so wie der Computer es macht, eine Spalte anwählen, zum Beispiel wie im Bild 2, die vierte von rechts. Das ergibt die Dualzahl 11110111, in dezimal die Zahl 247.

```
100 POKE 37152,247
200 PRINT PEEK(37152),PEEK(37153)
300 GOTO 100
```

Zeile 200 druckt uns sowohl den von uns gePOKEten Wert 247 als auch den in Register 37153 (56321) stehenden Wert aus, der dort erscheint, sobald wir eine Taste in der Spalte 247 drücken. Der Rücksprung in Zeile 300 erzeugt auf dem Bildschirm zwei senkrechte, durchlaufende Zahlenstreifen.

Nach RUN läuft rechts die Zahl 255 (dual = lauter 1er, das heißt, keine Taste gedrückt). Drücken Sie nun die »/«-Taste und es erscheint die 191. Die X-Taste erzeugt 251, die links angeordnete SHIFT-Taste eine 253. Die rechte SHIFT-Taste dagegen zeigt keine Reaktion – ist auch klar, denn sie liegt ja in einer anderen Spalte.

Aber es ist interessant, sich zu merken, daß bei einer Tastaturabfrage die beiden SHIFT-Tasten völlig eigenständig behandelt werden, im Gegensatz zu ihrer Funktion. Sie sehen, ich habe recht gehabt: Alle Tasten sind gleich.

Sie können auch die RUN-STOP-Taste drücken, aber bitte nur ganz, ganz kurz antippen, denn das Programm bleibt natürlich entsprechend ihrer ungeSHIFTeten Funktion sofort stehen. Wenn das Betätigen der Taste aber kurz genug war, dann steht als letzte Zahl die 254 da.

Gleichzeitige Abfrage von zwei Tasten

Jetzt kommt das große Experiment. Nach neuem RUN drücken Sie »Cursor down« und »/« gleichzeitig – es erscheint 63. Der Blick auf Bild 2 zeigt uns für beide Tasten das Bitmuster 00111111 und das ist 63. Es geht also. Probieren Sie ruhig alle Kombinationen aus. Wenn Sie Zeit und die Gelenkigkeit eines Konzertpianisten haben, können Sie alle Zahlen bis 254 erzeugen, auch die Null. Dazu sind alle acht Tasten gleichzeitig zu drücken.

Erwähnenswert ist, daß jetzt das Programm nicht stehenbleibt, obwohl die STOP-Taste gedrückt ist. Aber die SHIFT-Taste ist auch gedrückt, und das ist laut Handbuch die LOAD-RUN-Funktion, die jetzt nicht zum Tragen kommt.

Sie wollen sicher endlich auch einmal die Funktionstasten zur Geltung bringen. Dazu müssen wir aber eine andere Spalte anwählen. Ändern Sie bitte die Zeile 100 ab:

```
100 POKE 37152,239
```

In der Spalte 239 liegt die F1-Taste. Nach RUN passiert aber etwas Komisches: Trotz des POKens von 239 läuft links wieder die 247, genauso wie vorhin. Und die Tasten der Spalte 239 reagieren nicht, nur die der Spalte 247.

Was passiert da? Es liegt daran, daß nur beim Eintippen eines Programms die Tastaturabfrage genauso funktioniert wie beschrieben. Wenn aber ein Programm abläuft, dann ist der Computer nur an der STOP-Taste interessiert und schiebt deshalb immer wieder eine 247 in das Register 37152 (56320).

Eine Ausnahme gibt es auch hier: INPUT- und GET-Befehle fragen auch die anderen Tasten ab.

Diese erzwungene Vorfahrt der Spalte 247 können wir durch unsere Zeile 100 nicht ändern, denn wir sind ja nach RUN in einem Programmablauf. Die einzige Möglichkeit, andere Spalten POKen zu können, ergibt sich für uns durch Programmierung in Maschinensprache. Aber darauf will ich nicht jetzt schon, sondern erst am Schluß des Aufsatzes eingehen.

Wir haben also bei unserem Versuch, das Ergebnis der Befragung schon bei der Stimmenabgabe – sozusagen im Wahllokal – auszukundschaften, Pech gehabt. Das macht aber nichts, denn irgendwann wird ja ein Wahlergebnis offiziell bekanntgegeben. Bei der Tastaturabfrage ist das auch so. Vorher aber wollen wir wenigstens aus dem Teilergebnis, das wir ausspioniert haben, Kapital schlagen und die mehrfache Tastenabfrage der Spalte 247 an einem klaren Beispiel demonstrieren.

Programmsteuerung mit zwei unabhängigen Tasten

Das Programm soll auf dem Bildschirm in den Spalten 6 und 15 zwei senkrechte Bänder mit Sternen darstellen, deren Farbe mit der »X«- und der »/«-Taste unabhängig voneinander, auch gleichzeitig, verändert werden kann.

```
100 PRINT CHR$(147)
```

Diese Zeile löscht den Bildschirm. Die Methode (ASCII-CODE) kennen Sie sicher aus dem Programmier-Handbuch. Ich werde sie aber noch genau behandeln.

```
110 BS=PEEK(648)*256
```

```
120 FS=4*(PEEK(36866) AND 128) + 37888
```

Zeile 110 und 120 machen das Programm unabhängig von Speichererweiterungen. BS und FS sind Variable für die Anfangsadressen des Bildschirm- beziehungsweise des Farbspeichers.

```
130 F=2:G=2
```

Das ist die Anfangsfarbe »Rot« für beide Sternreihen.

```
200 FOR Z=0 TO 22
```

Mit Z werden die 23 Zeilen abgezählt.

```
300 POKE BS+5+Z*22,42
```

```
310 POKE FS+5+Z*22,F
```

Keine Angst, ich mute Ihnen keine höhere Mathematik zu. Ab Zeile 300 wird der Bildschirm-Code (42) für den Stern (auch das behandle ich später noch) in Spalte 6 (BS+5) für jede Zeile (Z*22) untereinander gePOKEt, dazu die Farbe F in den gleichen Platz des Farbspeichers. Dasselbe gilt in Zeile 400 und 410 für die Spalte 15.

```
400 POKE BS+15+Z*22,42
```

```
410 POKE FS+15+Z*22,G
```

```
500 AA=PEEK(37153)
```

```
510 IF AA=191 OR AA=187 THEN F=F+1
```

```
520 IF AA=251 OR AA=187 THEN G=G+1
```

Ab Zeile 500 werden die »X«-Taste (191) und die »/«-Taste (251) abgefragt. Wenn eine davon gedrückt ist, wird die Farbzahl F beziehungsweise G um 1 erhöht. Die OR-Funktion, mit welcher der Wert 187 abgefragt wird, erlaubt ein gleichzeitiges Drücken beider Tasten, es werden sowohl F als auch G erhöht.

Um die Farben zwischen 2 (Rot) und 7 (Gelb) zu begrenzen, verwenden wir:

```
600 IF F=8 THEN F=2
```

```
610 IF G=8 THEN G=2
```

Zum Schluß wird der Zeilenzähler Z weitergesetzt. Wenn er 22 (das heißt die 23. Zeile) erreicht hat, springt das Programm an den Anfang zurück.

```
700 NEXT Z
```

```
800 GOTO 200
```

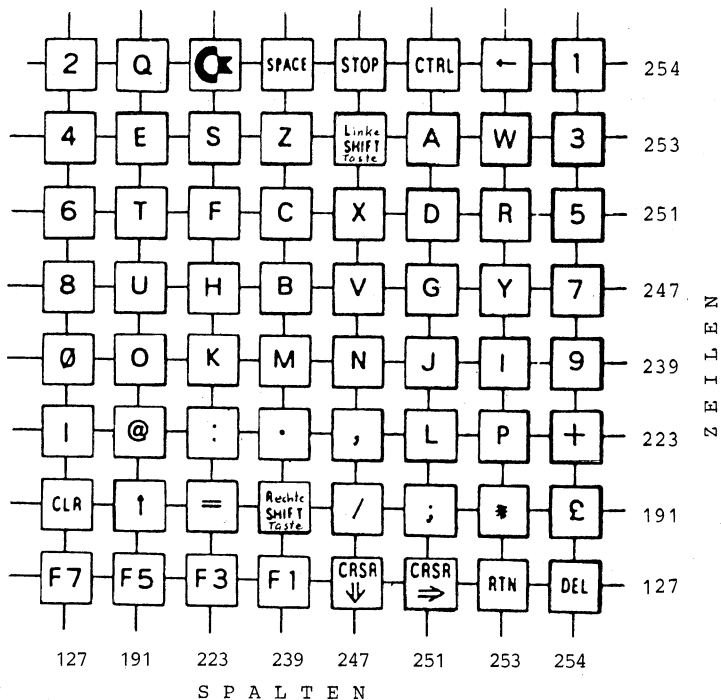


Bild 1. Die »elektrische« Anordnung der Tasten

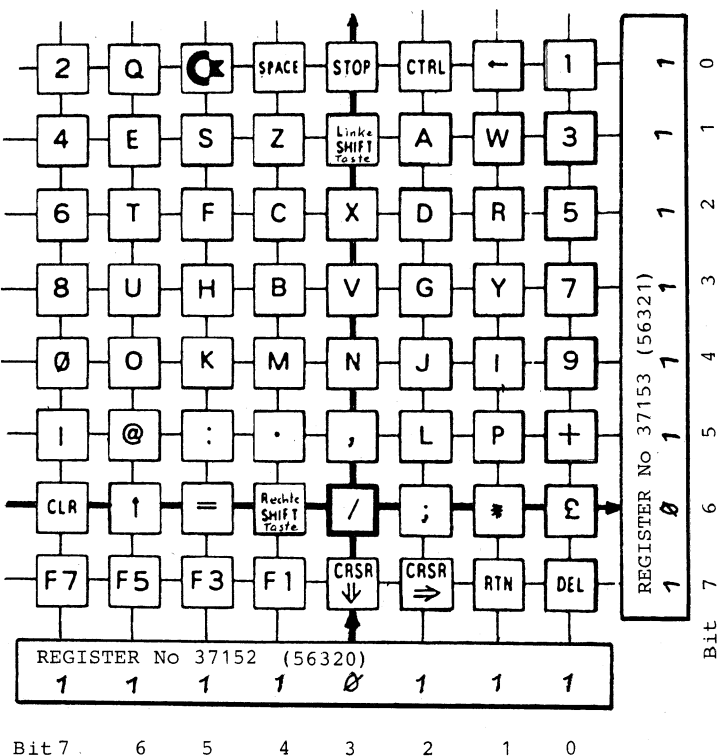


Bild 2. Die Tasten-Matrix mit ihren Registern

Nun geben Sie RUN ein. Wenn Sie eine der Tasten zu lange, das heißt länger als einen Z-Zyklus, drücken, springt die Farbe weiter. Um das etwas zu erleichtern, können Sie noch eine Verzögerung einbauen:

```
220 FOR T=1 TO 50: NEXT T
```

Ich gebe ja zu, das ist kein gewaltiges Programm. Aber es zeigt Ihnen wenigstens, wie auch in Basic eine mehrfache Tastenabfrage möglich ist.

```
100 PRINT CHR$(147)
```

```
110 BS=PEEK(648)*256
```

```
120 FS=4*(PEEK(36866) AND 128) + 37888
```

```
130 f=2:g=2
```

```
200 FOR Z=0 TO 22
```

```
220 FOR T=1 TO 50: NEXT T
```

```
300 POKE BS+5+Z*22,42
```

```
310 POKE FS+5+Z*22,F
```

```
400 POKE BS+15+Z*22,42
```

```
410 POKE FS+15+Z*22,G
```

```
500 AA=PEEK(37153)
```

```
510 IF AA=191 OR AA=187 THEN F=F+1
```

```
520 IF AA=251 OR AA=187 THEN G=G+1
```

```
600 IF F=8 THEN F=2
```

```
610 IF G=8 THEN F=2
```

```
700 NEXT Z
```

```
800 GOTO 200
```

Weiter geht's, und zwar mit dem schon erwähnten Bekanntgeben des Wahlergebnisses. In anderen Worten: Wie wertet der Computer die Tastenabfrage über die Register 37152 (56320) und 37153 (56321) weiter aus?

Sobald der Computer merkt, daß eine Taste gedrückt ist, nimmt er die beiden Zahlen, die in den Registern 37152 (56320) und 37153 (56321) stehen, und wandelt sie in eine Code-Zahl um, die er in der Speicherzelle 203 ablegt. Die Code-Zahl steht auch in der Speicherzelle 197. Mit dem Grund für diese Verdoppelung muß ich mich aber erst noch beschäftigen.

Wir bleiben bei Adresse 203. Wie bei der Abfrage der Tastatur-Matrix wollen wir uns den Inhalt dieser Speicherzelle ansehen. Löschen Sie bitte das alte Programm und geben Sie ein:

```
100 PRINT PEEK(203)
```

```
200 GOTO 100
```

Nach RUN sehen wir wieder die laufende Zahlenbank, jetzt aber mit 64. Das ist die Code-Zahl für »keine Taste gedrückt«. Die »/«-Taste ergibt jetzt 30, die X-Taste 26 und so weiter. Endlich ist es soweit! Die Funktionstasten reagieren und geben ihre Code-Zahl preis.

Die Funktionstasten reagieren doch!

Probieren Sie alle Tasten durch und schreiben Sie die Code-Zahlen auf die Tasten von Bild 1 oder Bild 2. Jetzt sehen Sie auch die Gesetzmäßigkeit, nach der der Computer die Spalten- und Zeilenzahl der Register ummodellt. Schreiben Sie sich am besten eine komplette Liste der Code-Zahlen für die weitere Verwendung. Die RUN-STOP-Taste läßt sich hier leichter als beim ersten Mal überlisten, natürlich nur mit gleichzeitigem SHIFT.

Apropos »gleichzeitig«! Wiederholen Sie das Experiment von vorhin. Hier erleiden wir unseren zweiten Fehlschlag: Mehrfach Tasten geben keinen Sinn, denn die Umcodierung verwehrt es uns. Wie gut, daß wir die Methode der Matrix-Abfrage haben, auch wenn sie in voller Eleganz nur in Maschinensprache möglich ist. Doch wie gesagt, davon später.

Zurück zu den einzelnen Tasten.

In der Liste der Code-Zahlen fehlen die Tasten RESTORE, SHIFT, C=, CTRL. Sie haben das nicht bemerkt? Dann haben

Sie auch noch nicht die von mir vorgeschlagene Liste gemacht!

Um auch diese Zahlen auf den Bildschirm zu bringen, ergänzen Sie bitte die Zeile 100 in

```
100 PRINT PEEK(203),PEEK(653)
```

```
200 GOTO 100
```

Jetzt sehen Sie zwei Zahlenreihen laufen. Zu der bekannten Reihe ist auf der zweiten Hälfte des Bildschirms (bedingt durch das Komma zwischen den PEEKs) eine 0-Reihe gekommen. Drücken Sie jetzt die SHIFT-Taste: Rechts läuft eine 1 – die Code-Zahl dieser Taste.

Die Steuertasten haben ihre Code-Zahl in der Speicherzelle 653

Die C=-Taste erzeugt eine 2, die CTRL-Taste eine 4 (und die natürlich ganz langsam). Drücken Sie mal SHIFT und C= gleichzeitig. Siehe da, bei der Speicherzelle 653 und ihren Steuertasten gibt das einen Sinn. Die Tabelle aller Kombinationen sieht so aus:

keine Taste	0
SHIFT	1
C=	2
SHIFT und C=	3
CTRL	4
SHIFT und CTRL	5
CTRL und C=	6
alle drei Tasten	7

Während wir das ausprobieren, läuft links unbeirrt die 64. Und in der Tat, durch das Aufspalten und Abspeichern der Code-Zahlen in zwei getrennte Speicherzellen können wir beide Tastenarten, Zeichentasten und Steuertasten, unabhängig voneinander und/oder gleichzeitig abfragen.

Das nutzen wir zum Beispiel bei den Funktionstasten aus. Jede von ihnen hat ihre eigene Code-Zahl in Adresse 203. Aber das gäbe uns nur vier Möglichkeiten, entsprechend der Aufschrift der ungeraden f-Zahlen.

Um auch f-2 bis f-8 zu erhalten, kombinieren wir die vier Zahlen in 203 einfach mit SHIFT-Taste gedrückt oder nicht (1 oder 0 in 653).

Aber Sie sehen schon, wie willkürlich das ist, denn wir könnten f-6 auch definieren als Kombination von der dritten Funktionstaste und CTRL (also 55 und 4).

Überhaupt, wir sind gar nicht auf acht Funktionstasten beschränkt, wie es uns durch den Ausdruck eingeredet wird. Die vier Funktionstasten ergeben zusammen mit den acht Kombinationen der Steuertasten 32 mögliche Funktionen. Natürlich gilt das für alle Tasten der Tastatur. Der Computer selbst nutzt allerdings nur wenige Kombinationen aus. SHIFT und C= (3) schaltet alle Buchstaben in Groß-/Kleinschrift um, die CTRL-Taste mit den Zahlentasten erzeugt die Vordergrund-Farben. Sie haben also viel Raum für phantasievolle Abfragekombinationen. Die Abfrage selbst und ihre Verwendung in einem Programm will ich abschließend mit den Funktionstasten demonstrieren.

Eine »Heimorgel« ganz besonderer Art

Wie man mit dem VC 20 Töne erzeugt, wissen Sie. In Zeile 10 definieren wir das Sopranregister und geben ihm den Namen Z, in Zeile 20 schalten wir die Lautstärke ein. Lautstärke des Fernsehers nicht vergessen!

```
10 Z=36876
```

```
20 POKE 36878,10
```

Ab Zeile 40 bis 110 wird jede einzelne Kombination der

Code-Zahlen von Funktions- und Steuertasten abgefragt. Sobald sie auftritt, wird ein entsprechender Ton der Tonleiter gePOKEt.

```
30 A=PEEK(203): B=PEEK(653)
```

```
40 IF A=39 AND B=0 THEN POKE Z,131
```

```
50 IF A=47 AND B=0 THEN POKE Z,145
```

```
60 IF A=55 AND B=0 THEN POKE Z,157
```

```
70 IF A=63 AND B=0 THEN POKE Z,162
```

```
80 IF A=39 AND B=1 THEN POKE Z,172
```

```
90 IF A=47 AND B=2 THEN POKE Z,181
```

```
100 IF A=55 AND B=4 THEN POKE Z,189
```

```
110 IF A=63 AND B=7 THEN POKE Z,193
```

Ein Rücksprung in die Zeile 30 verleiht dem Ton auch die Dauer.

```
120 GOTO 30
```

Damit ein Ton aber nur so lange klingt, wie eine Tastenkombination gedrückt ist, schieben wir noch Zeile 35 ein, die das Sopranregister auf 0 (Stille) setzt, sobald »keine Taste« gedrückt ist.

```
35 IF A=64 AND B=0 THEN POKE Z,0
```

Jetzt will ich Ihnen natürlich noch die von mir gewählten Tastenkombinationen verraten, damit Sie gezielt »Alle meine Entchen« spielen können. Es gilt der Reihe nach:

Zeile 40 f-1

Zeile 50 f-3

Zeile 60 f-5

Zeile 70 f-7

Zeile 80 f-1 und SHIFT

Zeile 90 f-3 und C=

Zeile 100 f-5 und CTRL

Zeile 110 f-7 und CTRL und C= und SHIFT

Ich gebe zu, daß diese Tastenauswahl nicht gerade eine bequeme Klaviatur ergibt.

Ihrem Ehrgeiz ist es überlassen, eine Orgel zu programmieren, die zwar immer noch einstimmig ist, aber eine »normale« Klaviatur hat und auch den vollen Tonumfang des VC 20 ausnutzt. Stellen Sie einfach die Code-Zahlen der Zeichentasten so zusammen, daß eine Tastenreihe die »weißen« Tasten und die darüberliegende Reihe die »schwarzen« Tasten darstellt.

Tastenabfrage und kein Ende: was es sonst noch alles gibt

Mit den Steuertasten können Sie die Oktaven umschalten, mit den Funktionstasten verschiedene Lautstärken. Die Zahlen für die Abfrage entnehmen Sie der Liste.

Statt Töne zu POKEn, können Sie natürlich mit dieser Abfragetechnik der Funktionstasten (und auch der anderen Tasten) alles mögliche per Programm steuern: Raumschiffe abschießen, Textseiten weiterschalten, den Hund rauslassen oder Toast rösten.

Ich habe Ihnen bisher gezeigt, wie alle Tasten des VC 20 beziehungsweise des C 64 in einer Matrix angeordnet sind. Sobald eine Taste gedrückt wird, steht eine spezielle, nur dieser einen Taste zugeteilte Code-Zahl im Register 37152 des VC20. Das entsprechende Register des C64 ist 56320.

Auch habe ich erklärt, wie die Code-Zahlen zustande kommen. In einem kleinen Demonstrationsprogramm haben wir dann durch Abfragen dieses Registers mit PEEK bestimmte Programmschritte mit Tastendruck gesteuert.

Wir haben herausgefunden, daß das Betriebssystem des Computers bei Verwendung von Basic eine Abfrage von nur acht Tasten zuläßt. Die Abfrage aller Tasten, auch mehrerer Tasten gleichzeitig, mit einem Programm in Maschinencode habe ich Ihnen für den Schluß versprochen.

```

LIST
10 PRINT CHR$(65)
20 PRINT CHR$(156)
30 PRINT CHR$(66)CHR$(13)CHR$(67)
99 END
READY.
RUN
A

B

C

Bild 3. Auf dem Bildschirm haben »B«
und »C« eine andere Farbe wie »A«
READY.

```

```

LIST
10 PRINT CHR$(65)
20 PRINT CHR$(156)
30 PRINT CHR$(66)CHR$(17)CHR$(67)
99 END
READY.
RUN
A

B

C

Bild 4. Resultat der Steuerfunktion 17,
«CURSOR DOWN»
READY.

```

```

LIST
5 POKE 198,5
10 POKE 631,65
20 POKE 632,156
30 POKE 633,66:POKE 634,17:POKE 635,67
99 END
READY.
RUN

READY.
AB
C

Bild 5. Die ASCII-WERTE wurden in den
Tastaturpuffer gePOKEt

```

```

LIST
5 POKE 198,10
10 POKE 631,65:POKE 632,13
20 POKE 633,156
30 POKE 634,66:POKE 635,17:POKE 636,67
99 END
READY.
RUN

READY.
A
?SYNTAX ERROR
READY.
B
C

Bild 6. Das Beispielprogramm mit dem
Codewert für RETURN

```

Wir sind aber noch einen Schritt weitergegangen und haben herausgefunden, daß diese Code-Zahlen der 64 Tasten umgerechnet und für Zeichen- und Steuertasten getrennt in die Speicherzellen 203 und 653 gebracht werden.

Mit dem folgenden kleinen Programm haben wir dann diese beiden Speicherzellen abgefragt und eine Tabelle (Tabelle 1) angefertigt.

Tippen Sie ein:

```

100 PRINT PEEK(203),PEEK(653)
200 GOTO 100

```

Der Grund, daß die Zahlen für die beiden Computer, trotz gleicher Tastatur, verschieden sind, liegt darin, daß die elektrische Anordnung der Tasten in einer 8 x 8-Matrix (die ich für den VC 20 gezeigt habe) beim C 64 anders sind.

Nun lesen Sie, wie man eine vollständige Tabelle der ASCII-Codes vom VC 20 und Commodore 64 erstellt. Außerdem werden Sie erfahren, daß die beiden Systeme wesentlich mehr Funktionstasten bieten, als Sie bisher vielleicht angenommen haben.

Die Wiederholung ist die Mutter der Weisheit

Im Gedenken an diesen Spruch meines Latein-Lehrers zeige ich noch einmal die Anwendung dieser Code-Zahlen im Listing 1. Zeilen in (), wie gesagt, sind für den C 64, aber nur dort, wo er sich vom VC 20 unterscheidet.

Natürlich wähle ich wieder die Funktionstasten und am besten auch noch eine andere Tastenkombination. Zweck der kleinen Demonstration soll das Umschalten auf verschiedene Bildschirmrahmen- und Hintergrundfarben sein. Ich schlage vor, Sie nehmen wie üblich die Zeitschrift zum Computer und lesen tippend weiter. Es folgt nun das Programm 1 zur Tastaturabfrage.

```

10 PRINT CHR$(147)

```

Diese Befehlsfolge löscht den Bildschirm. Die Code-Zahlen der Funktion CHR\$ werde ich noch erklären.

Die Zeilen 20 und 30 erleichtern die Tipperei und machen übrigens das Programm ein bißchen schneller. Sie ordnen den Variablen A und B den Inhalt der Speicherzellen 203 und 653 – die wir ja abfragen wollen – zu.

```

20 A=PEEK(203)

```

```

30 B=PEEK(653)

```

Jetzt geht's los mit der Fragerei. Die Taste f1, mit der wir die Farbkombination Blau/Gelb schalten wollen, hat folgende Code-Zahlen:

```

40 IF A=39 AND B=0 THEN POKE 36879,126

```

```

(40 IF A=4 AND B=0 THEN POKE 53280,6:POKE 53281,7 )

```

f2 ist dieselbe Taste, aber geSHIFTet (B=1). Die Farben sollen jetzt Rot/Grün sein:

```

50 IF A=39 AND B=1 THEN POKE 36879,45

```

```

(50 IF A=4 AND B=1 THEN POKE 53280,5: POKE 53281,2)

```

```

60 IF A=47 AND B=4 THEN POKE 36879,25

```

```

(60 IF A=5 AND B=4 THEN POKE 53280,1: POKE 53281,1)

```

Zeile 60 bestimmt ebenfalls eine Funktionstaste und zwar f3. Allerdings habe ich sie willkürlich mit der CTRL-Taste kombiniert, nicht, um Sie zu verwirren, sondern um zu zeigen, daß wir mit den f-Tasten mehr als acht Funktionen festlegen können, nämlich 32! (Vier f-Tasten mal acht Steuertasten-Kombinationen).

Mit dem Klammeraffen »@« schalten wir die Farben in den Normalzustand zurück.

```

70 IF A=53 AND B=0 THEN POKE 36879,27

```

```

(70 IF A=46 AND B=0 THEN POKE 53280,3: POKE 53281,1)

```

Zeile 80 läßt das Programm im Kreis laufen, so daß die Funktionstasten beliebig oft ausprobiert werden können.

```

80 GOTO 020

```

Funktionstasten im Überfluß

So, jetzt kommt der absolute Hit dieser Methode!

Ich habe gerade vorher gesagt, daß wir nicht acht, sondern 32 mögliche Funktionstasten haben, nämlich durch Verwen-

derung der vier f-Tasten mit den acht Steuertasten-Codes in Speicherzellen 653.

Dasselbe gilt für jede andere Taste natürlich auch!

Der Computer benützt allerdings einige davon, zum Beispiel die 1 (SHIFT) für die Zeichen über den Zahlen beziehungsweise rechts unten auf den Tasten, die 2 (C=) für die Zeichen links unten auf den Tasten und die 4 (CTRL) für die Farben.

Wir können daher mit den vom Computer benutzten Zahlen 3, 5, 6 und 7 aus Zelle 653 in Kombination mit allen 55 Tasten eine riesige Anzahl verschiedener »Funktionstasten« erfinden und sie in unseren Programmen einsetzen.

Der Computer geht bei der Abfrage noch einen Schritt weiter

Aber auch die 4 der CTRL-Taste, die ja nur die obere Reihe der Tasten beeinflusst, hat einen praktischen Wert, meiner Meinung nach sogar einen sehr großen, da sie ja nur einen einzigen Tastendruck erfordert und nicht eine Kombination.

Die CTRL-Taste kombiniert mit den 35 Tasten der unteren drei Reihen der Tastatur gibt uns mehr »Funktionstasten« als wir wahrscheinlich jemals brauchen werden.

Mehrere handelsübliche Zusatzmodule und -programme verwenden diese Methode, zum Beispiel auch die »Programmierhilfe« von Commodore. Verwenden Sie's doch auch! Das Kochrezept dazu steht oben in den Zeilen 50 bis 80.

Um das Ergebnis eines Tastendrucks weiter zu verarbeiten, wäre dem Computer die Abfrage zweier Speicherzellen zu langsam. Außerdem entsprechen diese Tastencodes keiner internationalen Norm, was in Verbindung mit anderen Geräten sehr lästig wäre.

Es gibt den international anerkannten ASCII-Code (American Standard Code for Information Interchange), der ursprünglich für die Zeichenübertragung von Fernschreibern erfunden wurde. Er besteht aus Dualzahlen mit einer Länge von 8 Bit oder falls Sie Dualzahlen nicht kennen (eine erste Einführung ins Dualsystem steht im Grafikkurs) aus Zahlen von 0 bis 255.

In diesen ASCII-Code wandelt der Computer nun die oben verwendeten Codezahlen der Tasten um. Die Umwandlung ist denkbar einfach.

Umrechnung des Tastatur-Codes in den ASCII-Code

Im nicht löschbaren Speicher (ROM) des Betriebssystems stehen vier Tabellen mit Zahlen.

Wenn man nun die Code-Zahl einer Taste zur Anfangsadresse der Tabellen addiert, erhält man eine Adresse, in der die ASCII-Codezahl gespeichert ist. Einfach, nicht wahr?

Probieren geht über studieren. Die Tabelle mit den ungeshifteten Zeichen, also der Zahlen und Großbuchstaben, beginnt beim VC 20 ab Speicherzeile 60510, beim C 64 ab 60289.

Nehmen wir das »G«, sein Tatencode aus der Tabelle ist 19 (26).

Zu 60510 (60289) dazugezählt ergibt das 60529 (60315). Nun wollen wir mal nachschauen, was in dieser Zelle steht.

Geben Sie »direkt«, (das heißt ohne Zeilenzahl) ein:

für VC 20: PRINT PEEK (60529)

für C 64: PRINT PEEK (60315)

Das Resultat ist 71. Ein Blick auf die ASCII-Liste des Handbuches (oder in jede andere ASCII-Tabelle) zeigt uns die Rich-

LIST

```
5 POKE 198,10
10 POKE 631,65:POKE 632,141
20 POKE 633,156
30 POKE 634,66:POKE 635,17:POKE 636,67
99 END
READY.
RUN

READY.
A
B
C
```

Bild 7. ▲
Die Wirkung der
ASCII-Zahl 141
(SHIFT RETURN)
im Beispiel-
programm

```
410 PRINT "SHIFT und CLR/HOME
"
```

```
420 GET A$
430 IF A$ = "" THEN 420
440 IF A$ = " f-1 " THEN POKE
36879,126
450 IF A$ = " f-2 " THEN POKE
36879,45
460 IF A$ = " f-3 " THEN POKE
36879,25
470 IF A$ = " G " THEN POKE
36879,27
480 GOTO 420
```

Für den C 64 gelten in den Zeilen
440 bis 470 andere POKE-Adressen:
440 POKE 53280,6:POKE 53281,7
450 POKE 53280,5:POKE 53281,2
460 POKE 53280,1:POKE 53281,1
470 POKE 53280,3:POKE 53281,1

Bild 8.
Programm zur
Abfrage mit
Gänsefüßchen

tigkeit dieser Aktion. Der ASCII-Code des Zeichens »G« ist 71.

Um alle ASCII-Codes abfragen zu können, schreiben wir wieder ein kleines Programm.

```
110 A=PEEK (203)
120 Z=60510
(120 Z=60289)
170 C=PEEK(Z+A)
```

Bitte verwenden Sie meine Zeilennummern, ich möchte nämlich später noch andere Zeilen einschieben.

Diese drei Zeilen sollten Ihnen klar sein. Zur Erinnerung: 60510 ist die Anfangsadresse der Code-Tabelle beim VC 20. Die 64er müssen also ihre eigene Adresse (60289) nehmen.

Als Ergebnis wollen wir noch den Tastaturcode (Zeile 110) und den ASCII-Code (Zeile 170) nebeneinander ausdrucken:

```
200 PRINT A;C
220 GOTO 110
```

ASCII-CODE	BILDSCHIRM-CODE
0 - 31	entspricht keinem Zeichen
32 - 64	32 - 64
64 - 95	0 - 31
96 - 127	64 - 95
128 - 159	entspricht keinem Zeichen
160 - 191	96 - 127
192 - 255	entspricht keinem ASCII-Code
	128 - 255

Bild 9. ASCII-Codes
mit entsprechenden
Bildschirm-Codes

Damit die beiden Zahlenbänder nicht zu schnell laufen, fügen wir noch eine Verzögerungsschleife ein.

```
210 FOR T=1 TO 200: NEXT
```

Dieser Programmablauf, der mit RUN 110 gestartet wird, reagiert jetzt auf jeden Tastendruck, links mit dem Tastencode, rechts mit ASCII. Worauf er aber nicht reagiert, sind geSHIFTete Zeichen, solche mit der Commodore-Taste »C« und die Farben (mit CTRL).

Sie wissen warum? Natürlich, denn wir fragen ja nur die Speicherzellen 203 ab und nicht zusätzlich auch 653.

Jetzt muß ich noch schnell erwähnen, daß auch der Computer diese Zellen abfragt.

Wo steht was?

Erinnern Sie sich, ich habe oben gesagt, daß im ROM Tabellen stehen:

- (1) ab 60510 (60289) für normale Zeichen
- (2) ab 60575 (60354) für Zeichen mit SHIFT
- (3) ab 60640 (60419) für Zeichen mit C=
- (4) ab 60835 (64632) für Zeichen (Farben) mit CTRL

Das bauen wir jetzt in das Programm ein:

Zeile 120 ändern wir ab. Sie fragt jetzt die Speicherzelle 653 nach den Steuertasten ab. In den Zeilen 130 bis 160 springen wir auf die vier Tabellenanfänge. Der Ausdruck in Zeile 200 schließlich wird mit der Codezahl aus 653 erweitert, nämlich B.

```
120 B=PEEK(653)
130 IF B=0 THEN Z=60510 (60289)
140 IF B=1 THEN Z=60757 (60354)
150 IF B=2 THEN Z=60640 (60419)
160 IF B=4 THEN Z=60835 (64632)
200 PRINT A;B;C
```

Die Reihenfolge der Zahlenbänder auf dem Bildschirm – entsprechend der Zeile 200 – ist jetzt von links Zeichentaste, Steuertaste, ASCII-Code.

Der ASCII-Code wird im Tastatur-Puffer abgelegt

Ehrlich gesagt, das Verfahren der Tastaturabfrage in dem vorherigen Programm ist immer noch recht kompliziert. Eine direkte Abfrage des ASCII-Codes einer Taste wäre viel besser.

Und in der Tat, der Computer bietet sie uns. Er bringt nämlich jeden ASCII-Wert in einen Speicher zur Zwischenlagerung, bis er von einem Programmschritt gebraucht wird.

Dieser Speicher heißt »Tastatur-Puffer« und liegt von Speicherzelle 631 bis einschließlich 640.

Es können also maximal zehn Werte gespeichert werden. Das erste Zeichen steht immer in 631, alle anderen werden der Reihe nach in die folgenden Zellen gebracht.

Als erstes wird das Zeichen aus 631 ausgelesen und alle anderen rücken nach.

Wenn das Programm die ASCII-Werte aus dem Tastaturpuffer auslesen kann, dann können wir das natürlich auch. Das folgende kleine Programm soll es beweisen.

```
310 PRINT CHR$(147)
330 A=PEEK(631)
```

Sie sehen, wir wollen ganz einfach in Zelle 631 des Tastaturpuffers nachschauen, welcher ASCII-Wert nach Drücken einer Taste dort steht (Zeile 330). In Zeile 340 drucken wir den Wert aus und springen dann zum PEEK-Befehl zurück.

```
340 PRINT A
350 GOTO 330
```

Wenn Sie dieses Programm mit RUN 310 laufenlassen, werden Sie merken, daß es nicht geht, ich will sagen: noch nicht geht. Den Grund dafür habe ich kurz vorher schon angedeutet.

Wo sind die Computer-Detektive? Haben Sie's gemerkt?

Nun, ich habe erklärt, daß die ASCII-Werte der gedrückten Tasten der Reihe nach im Tastaturpuffer gespeichert werden und dann, wenn ein Wert aus 631 ausgelesen wird, nachrücken. Das ist der springende Punkt: durch PEEKen lesen wir nicht aus, wir schauen nur nach!

Es gibt zwei Lösungen für dieses Problem:

1) Wir verwenden einen Befehl, der den Wert herausholt; das ist GET oder INPUT.

2) Wir gaukeln dem Computer vor, daß der Tastaturpuffer nur aus einer einzigen Speicherzelle besteht.

Die zweite Methode ist exotischer, deshalb zeige ich Sie Ihnen zuerst. Es gibt eine Speicherzelle 198. In dieser Zelle steht eine Zahl, die angibt, wieviele Zeichen im Tastaturpuffer Platz haben. Normalerweise steht da eine 10 (schauen Sie nach).

Diese Zelle kann mit kleineren Zahlen gePOKEt werden, auch mit einer 0. Die 0 löscht sozusagen den Puffer. Das machen wir jetzt in unserem Programm vor dem PEEKen:

```
320 POKE 198,0
```

```
350 GOTO 320
```

Jetzt läuft's.

Die Zeile 340 gibt uns also den ASCII-Code der gerade gedrückten Taste auf dem Bildschirm aus, zum Beispiel die Zahl 84 für das »T«, 163 für das »-« (T mit C=) und so weiter.

Unter Verwendung von GET würde unser Programm so aussehen:

```
310 PRINT CHR$(147)
```

```
320 GET A$
```

```
330 IF A$="" THEN 320
```

```
340 PRINT ASC(A$)
```

```
350 GOTO 320
```

Neu ist die Verwendung der Funktion ASC. Sie bildet den ASCII-Wert des Zeichens A\$.

Für diejenigen, die es noch nicht kennen: Zeile 330 nach dem GET ist erforderlich, da der GET-Befehl nicht auf das Drücken einer Taste wartet, sondern gleich weiterläuft. Solange aber keine Taste gedrückt ist, geht die Schleife nach 320 zurück.

Mit INPUT geht's noch kürzer, nur ist die Bedienung etwas umständlicher.

```
310 PRINT CHR$(147)
```

```
320 INPUT A$
```

```
340 PRINT ASC(A$)
```

```
350 GOTO 320
```

Der Umstand liegt daran, daß INPUT im Gegensatz zu GET auf einen Tastendruck wartet, der zusätzlich mit RETURN abgeschlossen werden muß.

Nach diesen Erklärungen und Versuchen müßten Sie eigentlich in der Lage sein, Listing 1 so umzuschreiben, daß statt der Abfrage der Speicherzellen 203 und 653 der Tastaturpuffer abgefragt wird. Ich schlage Ihnen vor, daß Sie das jetzt selbst ausprobieren, sozusagen als Hausaufgabe. Die Lösung zeigt Listing 4.

Das einzige, was ich Ihnen verraten will, wissen Sie eigentlich schon, nämlich, daß Sie sich die ASCII-Werte für die f-Tasten und den Klammeraffen »@« besorgen müssen. Aber wozu haben Sie Programm 3 gleich in drei Versionen?

Ich hoffe, daß Sie nach dieser Übung einsehen, daß Sie eine vollständige Liste aller ASCII-Codezahlen und ihrer Bedeutung unbedingt brauchen. Halt, werden Sie jetzt sagen, die Liste steht ja in jedem Handbuch – sogar in dem von Commodore.

Das stimmt, nur sind die meisten Listen nicht komplett.

Erinnern Sie sich? Ich habe vorher mal gesagt, daß der

ASCII-Code die Werte von 0 bis 255 hat. Diese werden von den Commodore-Computern in nicht immer ganz der Norm entsprechender Weise für alle möglichen Zeichen und Sonderfunktionen verwendet, wie zum Beispiel die Farben, die Sonderzeichen auf den Tasten, Zeichenumschaltung und so weiter.

Auch die Funktion »Bildschirm löschen und Cursor auf HOME-Position« (das heißt die CLR/HOME-Taste) ist dabei – mit dem Codewert 147. Merken Sie was? Schauen Sie mal die jeweiligen 10er-Zeilen der Programme bisher an!

Es lohnt sich also schon, alle Code-Werte und die dazugehörigen Zeichen und Funktionen anzusehen.

Ihnen die Liste einfach abzuzeichnen wäre zu simpel. Sie sollen ja durch Experimentieren Ihren Computer besser kennenlernen. Ich liefere Ihnen dazu die Versuchsanordnung.

Vorher aber brauchen wir noch ein Hilfsmittel, welches uns erlaubt, aus einem ASCII-Wert das Zeichen beziehungsweise die Funktion zu ermitteln. Es ist die Umkehrung der in Programm 3 verwendeten ASC-Funktion. Sie kommt ebenfalls aus Basic und heißt CHR\$(x).

Dieser Befehl liefert uns das Zeichen oder die Funktion des ASCII-Codes x.

Der Befehl PRINT CHR\$(x) bringt Zeichen auf den Bildschirm. (Mit dem Befehl PRINT # a,CHR\$ wird das Zeichen an ein beliebiges, mit der Nummer a bezeichnetes Peripheriegerät gebracht. Doch das will ich hier nicht weiter verfolgen.) Jetzt aber zurück zu dem Hilfsmittel.

Ergänzen Sie bitte in Programm 3 die Zeile 340 auf:

```
340 PRINT A, CHR$(A)
```

Jetzt druckt das Programm nach Start mit RUN 310 neben dem ASCII-Code auch das Zeichen, welches natürlich mit der gedrückten Taste identisch ist, auf den Bildschirm. Funktionen kann man allerdings nicht ausdrucken, sondern nur ihre Auswirkungen feststellen.

Doch nun zum Kochrezept. Der entscheidende Teil steht in Zeile 570 (Listing 5).

```
570 PRINT I;" [" CHR$(I) " ]" ;..." AAA"
```

I ist die ASCII-Codezahl, die in einer FOR-NEXT-Schleife von 0 bis 255 hochgezählt wird. Die beiden Klammern [und] stehen in Anführungszeichen, damit sie ausgedruckt werden. Zwischen ihnen soll das zum Wert I gehörige Zeichen stehen.

In den Fällen, wo der ASCII-Code nicht ein Zeichen, sondern eine Funktion bedeutet, bleibt die Klammer leer. Aber der Code wirkt sich durch die Form PRINT CHR\$(I) auf die 3 A aus (die Punkte ... stellen drei Leertasten dar). Zum Beispiel erscheinen sie nach I=28 in roter Farbe, bei I=17 (Cursor Down) eine Zeile tiefer.

Das Hochzählen von I in Zeile 520 wollen wir aber ein bißchen beeinflussen, und das natürlich mit Drücken von Funktionstasten und solchen, die wir dazu verdonnern.

In Zeile 600 werden daher solche Tasten abgefragt, mit der »alten« Methode in Zeile 203. Das ist reine Willkür beziehungsweise Sentimentalität von mir, die »neue« Methode über Zeile 631 geht genauso gut.

Wenn in 203 eine 64 steht (keine Taste gedrückt), dann wartet das Programm durch Rücksprung auf die Zeile 600.

Falls wir die f1-Taste drücken, schaltet Zeile 620 den Hintergrund auf Schwarz und geht wieder in Wartestellung. Diese und die beiden anderen Farbumschaltungen mit f3 (Zeile 630) auf Weiß und mit f5 auf Hellorange (Zeile 640) sind dann sehr nützlich, wenn die Farbe der drei A gegen den Hintergrund nur schlecht oder überhaupt nicht lesbar ist.

Zeile 650 gibt uns die Möglichkeit, mit der Minus-Taste in der Liste um 1 zurückzuschalten. Zeile 530 erlaubt ein Zurückschalten über die 0 nach 25. Erst wenn irgendeine beliebige Taste gedrückt wird, springt das Programm auf Zeile 660, wo nach kurzer Zeitverzögerung der Bildschirm gelöscht (70) und I weitergezählt wird (680 und 690).

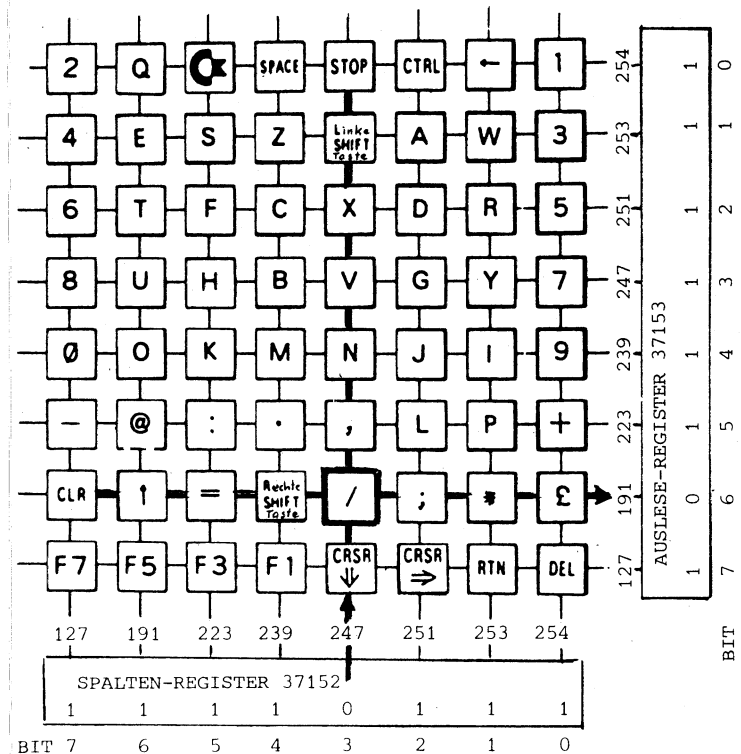


Bild 10. Die Tastenanordnung des VC 20

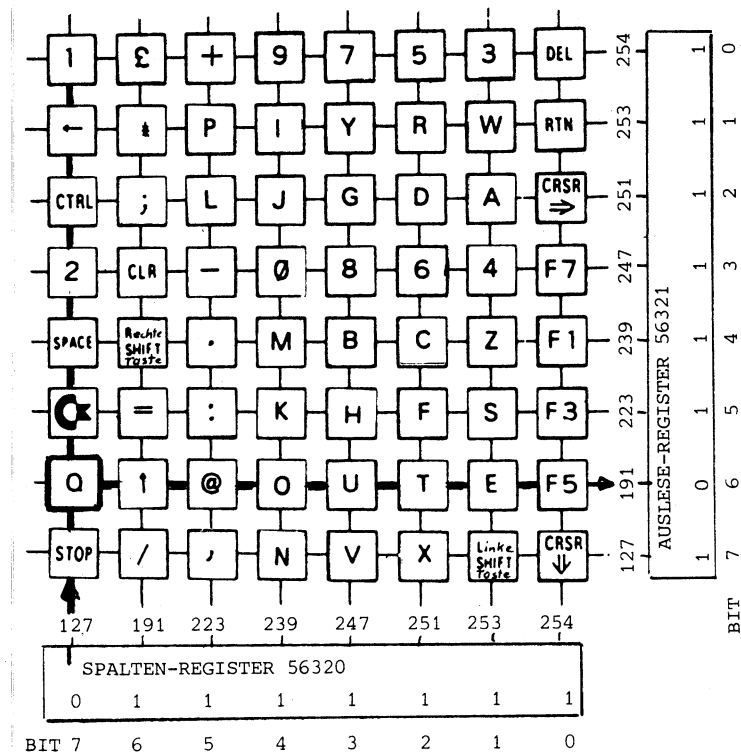


Bild 11. Die Tastenanordnung des C 64

```

5 REM*****C 64PROGRAMM 1***** <090>
10 PRINT CHR$(147) <039>
20 A=PEEK(203) <008>
30 B=PEEK(653) <153>
40 IF A=4 AND B=0 THEN POKE 53280,6: POKE <250>
   53281,2
50 IF A=4 AND B=1 THEN POKE 53280,5: POKE <002>
   53281,2
60 IF A=5 AND B=4 THEN POKE 53280,1: POKE <026>
   53281,1
70 IF A=46 AND B=0 THEN POKE 53280,3: POKE <113>
   53281,1
80 GOTO 20 <010>

```

Listing 1. Tastenabfrage in 203 und 653

```

105 REM *****C 64 PROGRAMM 2***** <042>
110 A=PEEK(203) <098>
120 B=PEEK(653) <243>
130 IF B=0 THEN Z=60289 <034>
140 IF B=1 THEN Z=60354 <131>
150 IF B=2 THEN Z=60419 <022>
160 IF B=4 THEN Z=64632 <191>
170 C=PEEK(Z+A) <251>
200 PRINT A;B;C <162>
210 FOR T=1 TO 200:NEXT <153>
220 GOTO 110 <164>

```

Listing 2. Umrechnung des Tastencodes in ASCII-Code

So können Sie sich bequem alle 256 Werte des ASCII-Codes und ihre Wirkung anschauen.

Ich möchte Sie hier noch auf folgende Codewerte aufmerksam machen, deren Funktion Sie in diesem Programm entweder nicht sehen können oder deren Funktion den Ablauf stören:

Code	Funktionen
3	entspricht der ungeSHIFTeten STOP/RUN-Taste
8	setzt die Umschaltung (mit SHIFT und C=) auf den 2. Zeichensatz außer Betrieb (ausprobieren!)
9	hebt die Sperre wieder auf
13	entspricht der RETURN-Taste
14	schaltet den 2. Zeichensatz per Programm ein (ich empfehle, danach wieder auf den »normalen« Zeichensatz zurückzuschalten)
131	entspricht LOAD/RUN (geSHIFTete STOP/RUN-Taste)
133-140	Funktionstasten f1 bis f8
141	geSHIFTete RETURN-Taste
142	schaltet den 1. Zeichensatz ein (Umkehrung von 14)
146	REVERSE-OFF (Taste 0 mit CTRL)
160	geSHIFTete SPACE-Taste (ja, ja, das gibt es auch!)

Ich empfehle Ihnen, mit dem Listing 5 zu experimentieren. Versuchen Sie, besonders die Funktionen zu identifizieren, es ist nicht schwer. Zusätzlich sollten Sie alle sinnvollen ASCII-Werte mit den Ihnen zur Verfügung stehenden ASCII-Listen vergleichen. Verbessern und vervollständigen Sie diese Listen. Sie gehören zu Ihrem wichtigsten Handwerkszeug.

Ich habe bisher versucht, Sie zum Experimentieren mit den ASCII-Codes anzuregen. In diesem Abschnitt habe ich für Sie die vollständige Liste aller 255 Codewerte vorbereitet. Zuvor aber möchte ich Sie erst mit einem Kochrezept einstimmen, welches uns ermöglicht, auch diejenigen ASCII-Werte einzusetzen, welche durch den Befehl PRINT CHR\$(.) nicht oder nur schwer darstellbar sind, wie zum Beispiel 8,8,131 und so weiter.

Dynamische Tastenabfrage

Dieses Kochrezept heißt »Dynamische Tastenabfrage« und ist mehrfach beschrieben (siehe Literaturangabe), wird aber, wie ich meine, nicht oft verwendet.

Nun bitte ich Sie, wie gewohnt am Rechner Platz zu nehmen und mir zu folgen.

Erinnern Sie sich? Der Codewert einer gedrückten Taste wird vom Betriebssystem des Rechners über eine fest eingespeicherte Tabelle in den ASCII-Codewert umgerechnet, dieser wiederum wird im »Tastaturpuffer« (Speicherplätze 631 bis 640) abgelegt. Aus diesem Puffer haben wir dann die Zahl herausgePEEKt.

Der Rechner macht genau dasselbe. Sooft wir auf eine Taste drücken, und wenn gerade kein Programm läuft, druckt er das Zeichen auf den Bildschirm oder führt die Funktion der Taste aus. Das ist der oft zitierte »Direkt-Modus«.

Wenn aber ein Programm läuft, dann bleiben die Codezahlen im Puffer so lange stehen, bis der Rechner fertig ist. Dann erst werden sie herausgeholt und verarbeitet. Das will ich Ihnen beweisen.

Tippen Sie im Direkt-Modus ein:

```
FOR K=1 TO 15000:NEXT K (RETURN)
```

Während diese an sich sinnlose Zeitschleife fünfzehntausendmal im Kreise rumrennt, haben Sie genügend Zeit, mehrere Tasten zu drücken, zum Beispiel die erste Buchstabenreihe (QWERTYUIOP@*!). Natürlich sehen Sie am Bildschirm gar nichts, denn das Programm der Schleife läuft ja noch. Sobald aber die Schleife zu Ende ist, erscheinen zehn der getippten Buchstaben. Quod erat demonstrandum! Warum nur zehn Buchstaben? Nun, der Tastaturpuffer hat halt nur zehn Plätze, logisch?

Jetzt ist eine gute Gelegenheit da, nochmal die Funktion der Speicherzelle 198 auszuprobieren. In 198 kann man nämlich eine Zahl hineinPOKEn, welche die Anzahl der Zeichen im Tastaturpuffer begrenzt.

Wiederholen Sie bitte das Experiment von oben, nur soll die direkt eingegebene Zeile erweitert werden:

```
FOR K=1 TO 15000:NEXT K:POKE 198,6 (RETURN)
```

Und siehe da, jetzt werden nur die sechs Buchstaben Q bis Y ausgedruckt. Diese Anwendung des Tastaturpuffers nützen wir für das Kochrezept »Dynamische Tastenabfrage« aus.

Löschen Sie bitte den Bildschirm und geben Sie ein (identisch für VC 20 und C 64):

```

10 PRINT CHR$(65)
20 PRINT CHR$(165)
30 PRINT CHR$(66)CHR$(13)CHR$(67)

```

```

305 REM*****C 64 PROGRAMM 3 ***** <248>
310 PRINT CHR$(147) <085>
320 POKE 198,0 <230>
330 A=PEEK(631) <192>
340 PRINT A <198>
350 GOTO 320 <072>
405 REM*****C 64 PROGRAMM 4***** <096>
410 PRINT CHR$(147) <185>
420 POKE 198,0 <074>
430 A=PEEK(631) <036>
440 IF A=133 THEN POKE 53280,6: POKE 53281 <161>
   ,7
450 IF A=137 THEN POKE 53380,5: POKE 53281 <178>
   ,2

```

Listing 3. Abfrage des ASCII-Codes aus dem Tastaturpuffer

```

405 REM*****C 64 PROGRAMM 4***** <096>
410 PRINT CHR$(147) <185>
420 POKE 198,0 <074>
430 A=PEEK(631) <036>
440 IF A=133 THEN POKE 53280,6: POKE 53281 <161>
   ,7
450 IF A=137 THEN POKE 53380,5: POKE 53281 <178>
   ,2
460 IF A=134 THEN POKE 53280,1: POKE 53281 <037>
   ,1
470 IF A=64 THEN POKE 53280,3: POKE 53281, <255>
   1
480 GOTO 420 <210>

```

Listing 4. Tastenabfrage aus Tastaturpuffer

65 ist der Code für A, 165 für die Farbe »purple«, 66 für B, 13 für »RETURN« und 67 für C.

Bild 3 zeigt den Ausdruck auf dem Bildschirm, wenn Sie diese Zahlen LISTen und RUNen.

Zur Erklärung: Die Leerzeile zwischen A und B ist bedingt durch die PRINT-Anweisung in Zeile 20, welche nur die Farbe umschaltet. Obwohl die Codes für B und C zusammen in einer Zeile stehen, werden sie doch durch das »RETURN« (13) untereinander gesetzt. Anstelle der 13 können Sie alle möglichen anderen Steuerfunktionen setzen. Bild 4 zeigt das Resultat von 17, nämlich »CURSOR DOWN«.

Wenn Sie die 8 nehmen, können Sie den Zeichensatz nicht mehr ändern. Der Einsatz der gleichzeitig gedrückten SHIFT- und C=-Tasten funktioniert erst nach CHR\$(9) wieder. Das Resultat von Bild 4 wollen wir jetzt durch POKEn der ASCII-Werte in den Tastaturpuffer versuchen.

```
5 POKE 198,5
10 POKE 631,65
20 POKE 632,156
30 POKE 633,66: POKE 634,17. POKE 635,67
99 END
```

Prinzipiell macht dieses Programm das gleiche wie das Programm in Bild 4. Trotzdem erhalten wir nach LIST und RUN ein anderes Ergebnis, nämlich das von Bild 5.

Ist das ein Fehler? Natürlich nicht. Schauen Sie her: Nach RUN laufen zuerst mal alle POKE-Befehle ab. Zeile 5 gibt an, wieviele Zeichen im Puffer stehen sollen. In Zeile 99 findet das Programm das ENDe und meldet sich mit READY. Jetzt erst wird im Tastaturpuffer nachgeschaut. Dort findet der Rechner zuerst das A, dann »purpur«, dann das B, welches sofort neben das A gesetzt wird. Das ist auch logisch, denn es fehlt ja jede Angabe, eine Zeile tiefer zu gehen. Um das zu erreichen, müssen wir in der Zeile 10 den Codewert für RETURN einschieben:

```
10 POKE 631,65: POKE 632,13
```

Vorsicht!! Sie müssen in den Zeilen 20 und 30 alle POKE-Adressen um 1 erhöhen und auch die Zahl in Zeile 5. Nehmen Sie 10, dann haben Sie Platz für Erweiterungen. So, jetzt LIST und RUN und es erscheint Bild 6 – und wir haben schon wieder ein Problem!

Aber alles im Rechner ist logisch! Nach dem A findet er den Wert für »RETURN«, also führt er den Befehl aus, auf dem der Cursor gerade steht. Der steht auf dem A. Da das kein gültiger Basic-Befehl ist, druckt der Rechner die Fehler-Meldung und zeigt READY an. Danach allerdings macht er weiter wie oben.

Und jetzt kommt die ASCII-Zahl 141 (SHIFT RETURN) voll zur Geltung. Diese Kombination nämlich setzt den Cursor an den Anfang der nächsten Zeile, ohne die Befehlsausführungsfunktion von RETURN. Ersetzen Sie also die 13 in Zeile 10 durch 141, dann läuft's (Bild 7).

Es gibt übrigens noch eine interessante ASCII-Codezahl, die in keiner Tabelle steht, nämlich 131. Sie bedeutet dasselbe wie die geSHIFTete STOP-Taste, also die Funktion »LOAD+RUN«. Wenn Sie diesen Code mit PRINT CHR\$(131) ausprobieren, funktioniert er allerdings nicht. Deshalb steht er wohl auch nicht in den Tabellen. In den Tastaturpuffer gePOKEt bringt er aber seine Wirkung. Setzen Sie bitte in Zeile 30 an die Stelle von 67 die Zahl 131 und anstelle des C erscheint

```
LOAD
PRESS PLAY ON TAPE
```

So, jetzt haben wir alle Zutaten für unser Kochrezept zusammen. Löschen Sie bitte alles bisherige und tippen Sie ein:

```
10 PRINT CHR$(147)
20 FOR I=1 TO 5: PRINT I
30 FOR T=1 TO 500: NEXT T
40 NEXT T
```

Nach Löschen des Bildschirms (Zeile 10) drucken wir zum Ausschmücken die Zahlen 1 bis 5 untereinander (Zeile 20 und 40) und damit es nicht zu schnell geht, bremsen wir mit der Zeile 30.

```
50 PRINT "LIST"
Das ist natürlich sehr einfach, aber jetzt kommt's!
60 POKE 198,5
70 POKE 631,145:POKE 632,145:POKE 633,145
80 POKE 634,13
90 END
```

Nach RUN erscheinen erst die fünf Zahlen, dann wird in einer neuen Zeile das Wort LIST geschrieben. Nach END wird erst (wie immer) eine Zeile ausgelassen, dann READY gedruckt und schließlich springt der Cursor an den Anfang der Zeile darunter. Während der Cursor anfangen will, somit drei Zeilen unter dem Wort LIST zu blinken, findet der Rechner im Puffer dreimal den ASCII-Code für »CURSOR UP«. Also geht dieser auch drei Zeilen hoch und will jetzt auf dem Wort LIST blinken.

Damit wird es aber wieder nichts, dann im Puffer steht ja noch der Code für RETURN (13). Das wird ausgeführt und zwar für das LIST. Es hat dieselbe Wirkung, als ob Sie direkt LIST tippen und danach die RETURN-Taste drücken, nämlich das Programm wird ausgeLISTet.

Alle sinnvollen Basic-Befehle, die Sie in der Zeile 50 PRINTen, werden durch diese dynamische Manipulation des Cursors ausgeführt. Versuchen Sie es mit

```
50 PRINT "PRINT 16 * 35"
50 PRINT "LOAD"
50 PRINT "GOTO 10"
50 PRINT "RUN"
50 PRINT "RUN 50"
und falls Sie dieses kleine Programm geSAVet haben
50 PRINT "SYS 64824"
(50 PRINT "SYS 64763")
```

Die Kunst ist also, mit entsprechenden Codezahlen den Cursor an diejenige Stelle des Bildschirms zu bringen, wo innerhalb eines Programms eine geeignete Anweisung gedruckt worden ist. Man kann damit getrennte Programmteile nach laden (LOAD), mit SYS-Befehlen Maschinenprogramme aufrufen, oder gar Programme durch sich selbst ändern lassen.

Die Weiterführung dieses Themas ist einen eigenen Beitrag wert, deshalb will ich hier abbrechen, denn ich bin Ihnen zum Thema »Tastenabfrage« noch einiges schuldig.

Kehren wir also wieder zur Liste der ASCII-Codes zurück.

Ich habe für Sie die vollständige Liste aller 255 Codewerte vorbereitet und zwar in einer Art, die sicher einiger Erklärungen bedarf.

ASCII-Codes, die von Commodore verschwiegen werden

Ich habe nämlich noch ein paar zusätzliche Überraschungen parat, auf die man nur durch Zufall kommt oder durch Studium des Betriebssystems oder aber, wenn man den Aufsatz von G. Urbanczyk in Computer persönlich vom 19.10.1983, Seite 76, gelesen hat.

Tippen Sie bitte Listing 3 ein, nämlich eine der drei Versionen zur Tastaturpuffer-Abfrage. (Ich verwende unten die GET-Version).

```
310 PRINT CHR$(147)
320 GET A$
330 IF A$= " " THEN 320
340 PRINT ASC(A$)
350 GOTO 320
```

Auf beiden, VC 20 und C 64 erhalten Sie nach RUN 310

```

505 REM *** Programm 5 *****
506 REM DER VOLLSTÄNDIGE ASCII-CODE *
507 REM *****
510 PRINT CHR$(147)
520 FOR I=0 TO 255
530 IF I<0 THEN I=255
540 PRINT:PRINT:PRINT:PRINT:PRINT
550 PRINT "-----"
560 PRINT
570 PRINT I;"[" CHR$(I);"]";"... AAA"
580 PRINT
590 PRINT "-----"
600 Z=PEEK(203)

610 IF Z=64 THEN 600
620 IF Z=4 THEN POKE 53280,3: POKE 53281,0: GOTO 600
630 IF Z=5 THEN POKE 53280,3: POKE 53281,1: GOTO 600
640 IF Z=6 THEN POKE 53280,3: POKE 53281,10: GOTO 600
650 IF Z=43 THEN I=I-2

660 FOR T=1 TO 100:NEXT T
670 PRINT CHR$(147)
680 NEXT I
690 GOTO 520

```

Listing 5. Der vollständige ASCII-Code

und Drücken der RETURN-Taste (natürlich) den ASCII-Code 82. Wenn Sie zuerst die CTRL-Taste drücken und halten und dann erst das R drücken, dürfte eigentlich nichts passieren, denn die CTRL-Taste gilt ja angeblich nur für die Farben. Ja, denkste! Wir erhalten nämlich die Zahl 18. Ein Blick in die ASCII-Tabelle zeigt uns für 18 die Funktion »REVERSE-ON«.

Versuchen Sie dasselbe mit CTRL und der ←-Taste. Wir erhalten die 6, und nicht 95, wie es eigentlich sein sollte.

Für den VC20 ist das alles. Aber immerhin, wir haben sozusagen noch zwei zusätzliche Funktionstasten gefunden.

Beim C 64 aber geht es erst richtig los:

Der Versuch wird Ihnen zeigen, daß alle Buchstaben, von A bis Z, zusammen mit CTRL gedrückt, einen anderen ASCII-Wert, nämlich 1 bis 26, ergeben, als allein gedrückt.

Des weiteren biete ich Ihnen noch:

```

CTRL - ! = 30
CTRL - = = 31
CTRL - $ = 28
CTRL - : = 27
CTRL - ; = 29

```

Das heißt aber, daß einige ASCII-Codezahlen zwei Bedeutungen haben. Oder umgekehrt, zwei verschiedene Tasten (kombiniert) haben denselben ASCII-Code.

Schwierigkeiten dadurch, daß einige ASCII-Werte zwei Bedeutungen haben, gibt es deswegen nicht, weil die Kombination mit CTRL nicht PRINT-bar ist (PRINT CHR\$(19) schickt immer den Cursor »home«, mit dem »S« passiert gar nichts).

Andersherum kann es allerdings vorkommen, daß eine Tastenabfrage, zum Beispiel

```
GET A$:IF A$ = CHR$(19) THEN .....
```

sowohl auf die Taste »HOME« als auch auf »CTRL-S« reagiert. Da ist sicher etwas Vorsicht angebracht. Aber ein Blick in meine ASCII-Tabelle zeigt Ihnen ja die Doppeldeutigkeiten.

An dieser Stelle erwarte ich eigentlich einige Einsprüche, wie: »Wozu das alles, die acht Funktionstasten, oder gar erweitert auf 32, reichen doch völlig aus!« Für den Hausbeziehungsweise Spielgebrauch ist das sicher richtig. Aber bei professioneller Software, welche benutzerfreundlich aufgebaut ist, kann es oft gar nicht genügend Funktionstasten – besonders solche, die eine optische Buchstabenbeziehung zu der Abfrage haben sollen – geben. Wenn in einem

Programm gefragt wird, ob Sie »LOADen« oder »SAVen« wollen, ist CTRL-L oder CTRL-S halt klarer als f-1 oder f-3.

Ich finde es schade, daß diese großartige Möglichkeit nur auf dem C64 gegeben ist, der doch professioneller ist als der VC 20.

Die vollständige ASCII-Tabelle

So, jetzt können Sie meine ASCII-Tabelle erst richtig interpretieren (Tabelle 2).

Leere Kästchen haben keine Bedeutung für die betreffende Codezahl.

Jeweils zwei Zeichen nebeneinander mit derselben Codezahl stellen die beiden Zeichensätze dar, in die mit C=SHIFT (Commodore-Taste) umgeschaltet werden kann. Wo nur ein Zeichen steht, ist es in beiden Zeichensätzen identisch.

Die Funktionen der Codezahlen 129 und 149 bis 155 gelten nur für den C 64. Interessant ist übrigens, daß die 4. und 7. Spalte identisch ist, ebenso die 6. und 8. Spalte (außer dem Zeichen für 255).

Ich möchte jetzt gern die Szene wechseln, ohne aber den ASCII-Code aus den Augen zu verlieren. Wir haben den ASCII-Code bisher verwendet, um Tasten abzufragen oder Funktionen auszuführen. PRINT CHR\$(66) druckt zum Beispiel den Buchstaben B auf den Bildschirm.

Welche Methoden kennen Sie noch, mit denen das gleiche erzielt werden kann?

Die erste, die jeder aus dem Handbuch lernt, ist
PRINT " B " .

Die komplizierteste ist:

```

POKE 7680,2: POKE 38400,7
(POKE 1024,2: POKE 55296,7)

```

Diese beiden Vorgehensweisen wollen wir uns näher anschauen und prüfen, ob wir sie in Analogie zu dem ASCII-Code für Tastenabfragen einsetzen können.

Der Gänsefüßchen-Modus

Es ist sicher viel bequemer, längere Buchstabenreihen oder gar Texte zwischen Gänsefüßchen gestellt einzutippen als eine Serie von CHR\$-Werten, ganz abgesehen vom erforderlichen Speicherplatz.

Nicht ganz so bequem ist der Gänsefüßchen-Modus bei Steuerzeichen, wie zum Beispiel Cursor-Bewegungen, besonders, wenn man diese herbeiführen will, aber statt dessen die reversen Darstellungen auf dem Bildschirm erzeugt.

Geben Sie es zu. Sie haben deswegen auch schon herzhaft geflucht. Auch jeder Redakteur bittet um Listings mit CHR\$-Darstellung anstelle der reversen Zeichen, die bei der Druckwiedergabe oft zu Schwierigkeiten führen.

Jetzt wissen Sie, warum ich bei meinen Progrämmchen immer PRINT CHR\$(147) statt PRINT " " verwende.

Genauso austauschbar wie bei PRINT ist der ASCII-Code mit dem Gänsefüßchen-Modus bei der Tastenabfrage.

Statt:

```

10 GET A$
20 IF A$ < > CHR$(65) THEN 10
30 PRINT CHR$(88)

```

können wir schreiben:

```

10 GET A$
20 IF A$ < > " A " THEN 10
30 PRINT " X "

```

Beide Programme sind gleichwertig. Nach RUN rührt sich gar nichts. Erst, wenn die A-Taste gedrückt wird (Zeile 20), druckt Zeile 30 den Buchstaben X.

In Zeile 20 können wir natürlich statt des A jeden beliebigen Buchstaben, Zahl oder Zeichen nehmen.

LISTEN Sie einfach die 2. Version der drei Zeilen, fahren mit dem Cursor auf das A und verändern Sie es Ihren Wünschen entsprechend. Wie ich Sie einschätze, machen Sie das sicher auch mit den Funktionstasten.

Nein? Dann machen Sie es mal. Sie haben nämlich Pech, so geht es nicht. Aber es geht, wenn Sie sich mit Absicht in die Lage begeben, die wie vorhin beschrieben, Flüche auslöst. Fahren Sie mit dem Cursor auf das 1. Gänsefüßchen, tippen Sie es noch einmal ein und drücken Sie dann eine Funktionstaste. Siehe da, es erscheint ein reverses Zeichen. Mit RETURN wird es »fixiert«, nach RUN wird das X erst mit der verwendeten Funktionstaste ausgelöst.

Der Trick besteht also darin, durch eine ungerade Anzahl von Gänsefüßchen-Eingaben diesen Modus herbeizuführen. Es geht ebenso durch Drücken der INSERT(INST)-Taste, allerdings nur für so viele Zeichen, wie oft sie gedrückt worden ist.

Im Gänsefüßchen-Modus erscheinen alle Steuer- und Funktionstasten in reverser Darstellung

Sie haben oben ein reverses Zeichen für die Funktionstasten erhalten. Die Zeichen für die Farben und Cursorbewegungen, also alle »gängigen« Funktionen, kennen Sie inzwischen sicher schon. Aber alle Steuer- und Funktionstasten?

Es gibt zwei Möglichkeiten, diese Zeichen zu finden: Die erste Methode verwendet entweder ganz primitiv im Direkt-Modus den Befehl: PRINT " mit nachfolgendem Drücken der Steuer- oder Funktionstaste oder sehr elegant den Dreizeiler

```
10 GET A$: IF A$=" " THEN 10
20 PRINT CHR$(34) A$ CHR$(34) ASC(A$)
30 GOTO 10
```

Auch hier ist ein kleiner Pfiff drin. In Zeile 20 wird zuerst ein Gänsefüßchen (34) gedrückt, wodurch wir in dem danach benannten Modus sind. Das nachfolgende A\$ erscheint im Fall einer Steuertaste als reverses Zeichen, nach dem abschließenden 2. Gänsefüßchen gibt uns die ASC-Funktion noch den ASCII-Code der gedrückten Taste.

Mit Gänsefüßchen statt CHR\$ hätten wir lediglich die beiden Zeichen A und \$ auf den Bildschirm gedruckt. Mit dieser Methode erhalten Sie zum Beispiel für:

```
ASCII-Code 17 = █
Cursor Down .....
ASCII-Code 147 = █
CLR .....
```

Die zweite Methode ist viel einfacher. Schauen Sie in meine ASCII-Tabelle (Tabelle 2). Sie ist in Spalten zu je 32 Zeichen angeordnet. Die Steuerzeichen und die Farben stehen alle in Spalte 1 und 5.

Da finden Sie zum Beispiel über der Codezahl 17 die Funktion »Cursor-Down«. Wenn Sie jetzt in die 3. Spalte waagrecht übergehen – also den Wert um 64 erhöhen – steht da das █. Oder: In Spalte 5 ist der Taste CLR die Codezahl 147 zugeordnet. Zwei Spalten weiter (64 höher) steht das █.

Wenn Sie die Ergebnisse der ersten Methode oben mit den durch Spaltenhüpfen gefundenen Zeichen vergleichen, sehen Sie, daß es dieselben Zeichen sind, halt nur reversiert. Machen wir die Probe:

Mit Methode 1 erhalten wir für »Rot« das reverse Pfund-Zeichen █. In der ASCII-Tabelle finden wir »Rot« unter 28. Zwei Spalten weiter, unter 28 + 64 = 92, steht dasselbe Zeichen. Das gilt auch für alle CTRL-Kombinationen, nicht nur für die der Farben.

```
5 REM*****
6 REM***** SPIEL MIT FINKELN *****
7 REM*****
10 PRINT CHR$(147)
20 F=0
30 R=65
40 FOR T=1 TO 600:NEXT T
50 A=INT(RND(0)*7)+65
60 IF R>71 THEN 400
70 PRINT CHR$(A);
80 FOR T=1 TO 1000:NEXT T
100 GET A$
110 IF A$<>" " THEN 300
120 PRINT " ";
200 IF A=R THEN F=F+1
210 GOTO 50
300 IF A<>R THEN F=F+1
310 R=R+1
320 GOTO 50
400 PRINT "DAS SPIEL IST ZU ENDE"; "
"F"FEHLER"
```

Listing 6. Programmtext zum »Spiel mit Finkeln« für C64

Bei beiden Computern entspricht dem CTRL-█ das █, beim C 64 erzeugt CTRL-█ ein █. Alle Kombinationen der Buchstaben mit CTRL erzeugen diese Buchstaben in reverser Darstellung.

Um das in einer kleinen praktischen Anwendung zu verdeutlichen, schlage ich vor, dieselbe Aufgabenstellung, die mit Tastencode-Abfrage und mit Tastaturpuffer-Abfrage gelöst wurde, noch einmal zu verwenden, jetzt aber die Gänsefüßchen-Methode einzusetzen.

Um beim Eintippen des Programms in Bild 8 sicherzustellen, daß alles klappt, habe ich statt der reversen Zeichen die Tasten angegeben beziehungsweise umrahmt, die nach dem ersten Gänsefüßchen gedrückt werden müssen.

Das Programm schaltet, wie die beiden anderen Versionen auch, die Bildschirm-Farben mit f-1, f-2, f-3 und @ um.

Ein letztes Problem bleibt uns noch. Wie schaffen wir es, daß wir im Gänsefüßchen-Modus auch Funktionen einsetzen können, die entweder keine eigene Taste haben (zum Beispiel 14 = Text, 8 = Lock) oder die beim Eintippen sofort die Funktion auslösen (zum Beispiel 13 = RETURN, 20 = DELETE)?

Hier müssen wir eine Methode anwenden, die meine Kinder und ich »finkeln« getauft haben und zwar deswegen, weil wir sie zum ersten und einzigen Mal vom Commodore-Software-Spezialisten Andy Finkel im amerikanischen Handbuch gefunden haben.

Sein Trick besteht darin, daß er in einer ASCII-Tabelle das entsprechende Zeichen für die Funktion herausucht und es in mehreren Schritten an seinen vorgesehenen Platz bringt.

Ich will Ihnen zeigen, was ich damit meine:

```
3 REM*****
4 REM**CODE-WANDLER**
5 REM*****
10 PRINT CHR$(147)
20 INPUT "ASCII-CODE";ACII
30 IF ACII=0 THEN END
40 IF ACII AND 128 THEN BILD=ACII AND 127 OR 64:GOTO 80
50 IF NOT ACII AND 64 THEN BILD=ACII:GOTO 80
60 IF ACII AND 32 THEN BILD=ACII AND 95:GOTO 80
70 BILD=ACII AND 63
80 PRINT TAB(5) "BILDSCH.CODE:"BILD
90 GOTO 20
```

Listing 7. Ein ASCII-Code Wandler

```

4 REM *****
5 REM   VIELFACHTASTEN
6 REM   ABFRAGE
7 REM *****
10 PRINT CHR$(147)
20 POKE 51,235
30 POKE 52,29
40 POKE 55,235
50 POKE 56,29
100 DATA 169,254,141,0
101 DATA 220,173,1,220
102 DATA 141,255,29,96
105 REM -----VC20-----
106 REM   100 DATA169,254,141,32
107 REM   101 DATA145,173,33,145
108 REM   102 DATA141,255,29,96
109 REM -----VC20-ENDE-----
110 FOR K=1 TO 12
120 READ X
130 POKE 7660+K,X
140 NEXT K
200 POKE 7662,127
210 SYS 7661
220 PRINT " 8"PEEK(7679);
300 POKE 7662,191
310 SYS 7661
320 PRINT PEEK(7679);
400 POKE 7662,223
410 SYS 7661
420 PRINT PEEK(7679);
500 POKE 7662,239
510 SYS 7661
511 REM -----VC20-----
512 REM   520 PRINT PEEK(7679)
513 REM   550 GET A$
514 REM   560 IF A$<>" " THEN 200
515 REM -----VC20-ENDE-----
520 PRINT PEEK(7679);
600 POKE 7662,247
610 SYS 7661
620 PRINT PEEK(7679);
700 POKE 7662,251
710 SYS 7661
720 PRINT PEEK(7679);
800 POKE 7662,253
810 SYS 7661
820 PRINT PEEK(7679);
900 POKE 7662,254
910 SYS 7661
920 PRINT PEEK(7679)
940 REM -----VC20-----
950 REM   GET A$
960 REM   IF A$<>" " THEN 600
970 REM -----VC20-ENDE-----
1000 GOTO 200

```

READY.

Listing 8. Programm zum Abfragen und Anzeigen aller Tasten

Bitte, versuchen Sie mit der Gänsefüßchen-Methode die DELETE-Taste in eine PRINT-Anweisung zu bringen
10 PRINT " INST/DEL "
Sie werden es nicht schaffen, da die DEL-Taste, statt ein reverses Zeichen zu drucken, ihrer Funktion nachgeht und das vorherige Zeichen löscht.

Jetzt »finkeln« wir:

1. Schritt:

10 PRINT " "
(mit RETURN abschließen)

```

0 REM*****
1 REM** SPIEL FUER **
2 REM** 4 PERSONEN **
3 REM*****
4 REM
5 REM
6 REM*****
7 REM** EINGABE **
8 REM**MASCHINENCODE**
9 REM*****
10 PRINT CHR$(147)
20 POKE 51,235
30 POKE 52,29
40 POKE 55,235
50 POKE 56,29

```

VC 20: 100 DATA 169,0,141,32,145,173,33,145,141,255,29,96

C 64: 100 DATA 169,0,141,0,220,173,1,220,141,255,29,96

```

110 FOR K=1 TO 12
120 READ X
130 POKE 7660+K,X
140 NEXT K
199 REM*****
200 REM**ANWEISUNGEN**
205 REM*****
210 PRINT CHR$(147)
215 POKE830,0

```

```

220 PRINT"100SPIELER 1:(W)TASTE
230 PRINT"200SPIELER 2:(Z)TASTE
240 PRINT"300SPIELER 3:(DEL)TASTE
250 PRINT"400SPIELER 4:(CRSR+)TASTE

```

C 64: 220 PRINT"100SPIELER 1 DRUECKT '+'
230 PRINT"200SPIELER 2 DRUECKT 'COMMODORE'
240 PRINT"300SPIELER 3 DRUECKT 'INST/DEL'
250 PRINT"400SPIELER 4 DRUECKT 'CRSR+'
260 PRINT TAB(125)" 26SPACE"

```

270 GET A$:IF A$=" " THEN 270
280 PRINT CHR$(147)
300 POKE 36879,8
310 PRINT"100WER SIEHT ALS ERSTER
320 PRINT"200DEN BUCHSTABEN
330 BU = INT(RND(0)*10)+1

```

VC 20: 340 POKE 7808,BU
350 POKE 38528,7
360 PRINT"100UND DRUECKT SEINE 1TASTE ?

C 64: 340 POKE 1224+17,BU
350 POKE 55496+17,7
360 PRINT"100UND DRUECKT SEINE TASTE ?

```

370 PRINT TAB(165)" 26SPACE"
380 GET A$:IF A$=" " THEN 380
390 PRINT CHR$(147)
400 FOR T=1 TO 800:NEXT T
490 REM*****
495 REM* BUCHSTABEN *
500 REM** WUERFELN **
510 REM*****
520 VB=INT(RND(0)*10)+1
530 IF VB=BU THEN POKE 830,250

```

VC 20: 540 C=INT(RND(0)*505)
550 POKE 7680+C,VB
560 POKE 38400+C,1

C 64: 540 C=INT(RND(1)*501)*2
550 POKE 1024+C,VB
560 POKE 55296+C,1

```

590 REM*****
595 REM** ABFRAGE ***
600 REM* DER TASTEN *
605 REM*****
610 FOR Z=1 TO 15
620 POKE 7662,253
630 SYS 7661
640 IF PEEK(7679)=253 AND PEEK(830)=250 THEN 810
650 POKE 7662,239
660 SYS 7661
670 IF PEEK(7679)=253 AND PEEK(830)=250 THEN 820
680 POKE 7662,254
690 SYS 7661
700 IF PEEK(7679)=127 AND PEEK(830)=250 THEN 830
710 POKE 7662,251
720 SYS 7661
730 IF PEEK(7679)=127 AND PEEK(830)=250 THEN 840
740 NEXT Z

```

Listing 9. Ein Tastenspiel für vier Personen

2. Schritt:

Mit dem Cursor auf die Leerstelle zwischen den Gänsefüßen fahren.

3. Schritt:

Aus der ASCII-Tabelle das Zeichen der DEL-Taste holen (T).

4. Schritt:

Die reverse Darstellung mit CTRL-REV.ON einschalten (der Cursor bleibt auf seiner Stelle) und das T drücken, mit RETURN abschließen. Jetzt steht das Zeichen drin und das Programm läuft.

Um Ihnen den Schritt 3 für alle widerborstigen Funktionen zu erleichtern, habe ich sie alle in der Tabelle 3 zusammengefaßt.

Da ich hoffe, daß Sie in Zukunft fleißig finkeln werden, muß ich Sie noch über einen lästigen Nebeneffekt aufklären, der bei ein paar Finkleien auftritt. Einige der Funktionen, nämlich RETURN, DElete (schon wieder) und das SHIFT-RETURN wirken nicht nur im Programmablauf wie vorgesehen, sondern auch beim LISTen, was lästig sein kann. Allerdings ergeben sich dadurch auch ungeahnte Möglichkeiten – siehe Artikel »Synthetische Steuerzeichen«. Das geSHIF-Tete RETURN (ASCII-Code 14) ist sehr nützlich bei Platz- und Speichermangel. Sie können nämlich mit " █ " in einer langen Programmzeile den Cursor mit nur drei Zeichen auf den Anfang der nächsten Zeile bringen, mit CHR\$(141) bräuchten Sie schon neun Zeichen, mit SPC(...) müssen Sie sehr genau die Cursorposition berechnen, mit einer entsprechenden Anzahl von »Cursor-Rechts«-Zeichen geht es auch nur mühsam.

Also, nützlich ist SHIFT-RETURN durchaus!

Nur: Beim LISTen wird es auch ausgeführt und die Zeile, in der es steht, sieht recht blöd aus. Zusätzlich kann eine derartige geLISTete Zeile nicht mehr geändert werden, sondern muß bei Verbesserungen völlig neu geschrieben und gefinkelt werden. Alles Gute hat seinen Preis!

Soviel sei zur Methode gesagt. Jetzt wollen wir zur Erholung und zur Übung ein kleines Spiel programmieren, in dem wir (fast) alles Gelernte auch anwenden.

Eine kleine Seltenheit ist bemerkenswert: Das Programm ist für VC 20 und C 64 identisch!

Die Spielaufgabe soll darin bestehen, die ersten sieben Buchstaben des Alphabets möglichst in der richtigen Reihenfolge auf den Bildschirm zu bringen.

Klingt einfach, aber die Buchstaben sollen in zufälliger Reihenfolge auftauchen. Zusätzlich hat der Spieler, falls der Buchstabe nicht der Reihenfolge entspricht, lediglich die Möglichkeit, ihn mit der DEL-Taste zurückzuweisen, wenn er

schnell genug ist. Das Programm zählt die Felder und zeigt am Schluß das Ergebnis an.

Wir brauchen dazu:

- Einen Zufalls-Buchstaben-Erzeuger von A bis G (ASCII-Code 65 bis 71)
- einen Buchstaben-Drucker
- einen Buchstaben-Reihenfolgezähler
- eine Möglichkeit, die DEL-Taste zu drücken und damit den gedruckten Buchstaben rückgängig zu machen
- einen Fehlerzähler
- eine Prüfung, ob der letzte Buchstabe (71) erreicht ist.

Normalerweise müßte ich jetzt ein Flußdiagramm zeichnen und »strukturiert« vorgehen, so wie die ausgezeichnete Serie in diesem Heft lehrt. Man möge mir aber verzeihen, daß ich aus Erklärungsgründen in einzelnen Schritten vorgehe, welche uns erlauben, jederzeit Zwischenresultate mit Probeläufen zu überprüfen (Listing 6).

Auf geht's!

Den Buchstaben-Erzeuger und -drucker erhalten wir durch Zeile 50, welche für eine Variable A zufällige ASCII-Codes zwischen 65 und 71 erzeugt, sowie durch Zeile 70, die das Zeichen für den ASCII-Code ausdrückt.

```
50 A=INT(RND(0)*7)+65
```

```
70 PRINT CHR$(A);
```

Für weniger Versierte sei gesagt, daß RND(0) eine Zufallszahl zwischen 0 und 0,99 erzeugt, mit 7 multipliziert gibt das eine Zahl zwischen 0 bis 6,93. Die Funktion INT macht daraus eine ganze Zahl zwischen 0 und 6, mit 65 addiert letztlich eine Zahl zwischen 65 und 71 = ASCII-Werte der Buchstaben A bis G.

Den Ausdruck der Buchstaben nebeneinander erreichen wir durch das Semikolon in Zeile 70, die laufende Wiederholung durch einen Rücksprung in Zeile 320.

```
320 GOTO 50
```

Damit es nicht zu schnell geht, verzögern wir das Ganze mit einer Warteschleife in Zeile 80.

```
80 FOR T=1 TO 1000:NEXT T
```

Probieren Sie es mit RUN aus. Zeile 80 übrigens erlaubt Ihnen später den Schwierigkeitsgrad zu verändern.

Die geforderte richtige Reihenfolge der Buchstaben A, ich nenne sie hier R, setzen wir am Anfang auf 65 und erhöhen sie schrittweise um 1.

```
30 R = 65
```

```
310 R = R+1
```

In Zeile 60 prüfen wir, ob die Endzahl 71 für das G überschritten ist. Wenn ja, springen wir auf Zeile 400, mit der wir das Spielende anzeigen.

C 64

```

620 POKE 7662,127
630 SYS 7661
640 IF PEEK(7679)=253 AND PEEK(830)=250 THEN 810
650 IF PEEK(7679)=223 AND PEEK(830)=250 THEN 820
660 IF PEEK(7679)=221 AND PEEK(830)=250 THEN 850
670 POKE 7662,254
680 IF PEEK(7679)=254 AND PEEK(830)=250 THEN 830
690 IF PEEK(7679)=251 AND PEEK(830)=250 THEN 840
700 IF PEEK(7679)=250 AND PEEK(830)=250 THEN 850
710 IF PEEK(7679)=250 AND PEEK(830)=250 THEN 850
720 NEXT Z
750 GOTO 500: REM (NEUER BUCHSTABE)
799 REM*****
800 REM** ERGEBNIS **
805 REM*****
810 PRINT"SPIELER 1 IST SIEGER":GOTO 860
820 PRINT"SPIELER 2 IST SIEGER":GOTO 860
830 PRINT"SPIELER 3 IST SIEGER":GOTO 860
840 PRINT"SPIELER 4 IST SIEGER":GOTO 860
850 PRINT"ZWEI SPIELER WAREN GLEICHZEITIG..."
860 PRINT"NOCH EINMAL ? (J/N)"
870 GET A$:IF A$="J" THEN 200
880 IF A$="N" THEN END
890 GOTO 870
READY.
```

Listing 9.
Ein Tastenspiel
für vier Personen (Schluß)

TASTE	VC-20		C-64	
	203	653	203	653
nichts	64	0	64	0
f-1	39	0	4	0
f-3	47	0	5	0
f-5	55	0	6	0
f-7	63	0	3	0
A	17	0	10	0
B	35	0	28	0
C	34	0	20	0
D	18	0	18	0
E	49	0	14	0
F	42	0	21	0
G	19	0	26	0
H	43	0	29	0
I	12	0	33	0
J	20	0	34	0
K	44	0	37	0
L	21	0	42	0
M	36	0	36	0
N	28	0	39	0
O	52	0	38	0
P	13	0	41	0
Q	48	0	62	0
R	10	0	17	0
S	41	0	13	0
T	50	0	22	0
U	51	0	30	0
V	27	0	31	0
W	9	0	3	0
X	26	0	23	0
Y	11	0	25	0
Z	33	0	12	0
1	0	0	56	0
2	56	0	59	0
3	1	0	8	0
4	57	0	11	0
5	2	0	16	0
6	58	0	19	0
7	3	0	24	0
8	59	0	27	0
9	4	0	32	0
0	60	0	35	0
+	5	0	40	0
-	61	0	43	0
*	14	0	49	0
/	30	0	55	0
=	46	0	53	0
↑	54	0	54	0
↓	8	0	57	0
.	37	0	44	0
:	45	0	45	0
,	29	0	47	0
;	22	0	50	0
'	6	0	48	0
~	53	0	46	0
CRSR+	23	0	2	0
CRSR↑	31	0	7	0
DEL	7	0	0	0
HOME	62	0	51	0
STOP	24	0	63	0
RETURN	15	0	1	0
SPACE	32	0	60	0
SHIFT	64	1	64	1
C=	64	2	64	2
CTRL	64	4	64	4
SHIFT u. C=	64	3	64	3
SHIFT u. CTRL	64	5	64	5
C= u. CTRL	64	6	64	6
SHIFT u. C= u. CTRL	64	7	64	7

Tabelle 1. Tabelle des Tastencodes in den Registern 203 und 653. Eine komplette Tabelle, in der auch die Codes für Grafikzeichen enthalten sind, finden Sie im Sonderheft 8/85 (Assembler)

```
60 IF R > 71 THEN 400
400 PRINT "SPELENDEN"
SPELENDEN
```

Bitte RUNnen Sie das Fragment wieder zur Probe.

Jetzt kommt die Beeinflussung der Reihenfolge mit der DEL-Taste. Wie gelernt fragen wir diese Taste mit einer GET-Schleife ab (Zeilen 100,110), ihre Lösch-Wirkung erreichen wir in Zeile 120 durch einen PRINT-Befehl (mit Semikolon!). Nach Drücken der DEL-Taste darf der Reihenfolge-Zähler der Zeile 310 natürlich nicht wirken, deshalb springen wir schon vorher aus der Zeile 210 zurück.

```
100 GET A$
110 IF A$ <> " " THEN 300
120 PRINT " ";
210 GOTO 50
```

Sie sehen oben, daß ich für die Abfrage der DEL-Taste die Finkel-Methode vorschlage. Die anderen Methoden gehen natürlich auch.

Nach RUN springt das Programm auf die noch nicht existierende Zeile 300 (was prompt zur Fehlermeldung führt), es sei denn, Sie drücken rechtzeitig die DEL-Taste.

In der Zeile 300 wollen wir prüfen, ob ein Fehler gemacht wurde, das heißt, ob A mit der Reihenfolge R übereinstimmt. Im Fehlerfall wird die Fehlerzahl F um 1 erhöht. Vorher aber muß F auf 0 gesetzt werden.

```
20 F=0
300 IF A <> R THEN F=F+1
```

Sie können jetzt schon das Spiel üben. Aber es fehlen noch ein paar Feinheiten.

```
10 PRINT CHR$(147)
410 PRINT F " Fehler"
```

Zeile 10 ist klar, Zeile 410 druckt am Spielende die Fehlerzahl F aus.

Aber es gibt noch einen Fehler des Spielers, nämlich wenn er aus Versehen einen richtigen Buchstaben zurückweist. Deshalb fragen wir nach erfolgtem Drücken der DEL-Taste in den Zeilen 100 bis 120 nach, ob der Buchstabe tatsächlich falsch war. Wenn nicht, wird die Fehlerzahl F um 1 erhöht.

```
200 IF A=R THEN F=F+1
```

Damit uns nach RUN der erste Buchstabe nicht überrascht, verzögern wir sein Erscheinen mit

```
40 FOR T=1 TO 600: NEXT T
```

Zum Finkeln-Üben arrangieren wir die Anzeige des Spielendes und der Fehler etwas um. Alle Anweisungen sollen in nur einer Zeile stehen. Löschen Sie bitte die Zeile 410. In Zeile 400 wird gefinkelt und zwar mit dem Zeichen für SHIFT RETURN, welches laut Tabelle 2 mit SHIFT M erzeugt wird.

```
400 PRINT "SPELENDEN"; "Fehler"
```

Bei LIST und bei Ausdruck mit einem Drucker sehen die gefinkelten Zeilen 110, 120 und 400 natürlich kurios aus und wie gesagt, sie lassen sich bei einem Tippfehler nicht korrigieren, sondern müssen neu geschrieben werden.

Der Bildschirm-Code

Der Vollständigkeit halber will ich noch die letzte der vorher genannten drei Methoden erwähnen, mit denen man ein Zeichen auf den Bildschirm bringt, insbesondere, weil der dabei verwendete Bildschirm-Code (auch Video-Code genannt) oft zu Verwechslungen mit dem ASCII-Code führt.

Auf Anhieb ist es auch nicht einzusehen, warum Commodore einen anderen Code verwendet, wenn ein Zeichen direkt auf den Bildschirm – oder genauer gesagt, in den Bildschirm-Speicher – gePOKEt werden soll.

Der Grund dafür liegt darin, daß dem ASCII-Code nicht nur Zeichen zugeordnet sind, sondern auch Farben und Funktio-

KOMBINATIONEN		1
VC 20	C 64	
	CTRL - A	000
	CTRL - B	001
	CTRL - C	002
	CTRL - D	003
	CTRL - E	004
	CTRL - F	005
CTRL - -	CTRL - -	006
	CTRL - G	007
	CTRL - H	008
	CTRL - I	009
	CTRL - J	010
	CTRL - K	011
	CTRL - L	012
	CTRL - M	013
	CTRL - N	014
	CTRL - O	015
	CTRL - P	016
	CTRL - Q	017
CTRL - R	CTRL - R	018
	CTRL - S	019
	CTRL - T	020
	CTRL - U	021
	CTRL - V	022
	CTRL - W	023
	CTRL - X	024
	CTRL - Y	025
	CTRL - Z	026
	CTRL - :	027
	CTRL - £	028
	CTRL -]	029
	CTRL - !	030
	CTRL - =	031

2	3	4	5	6	7	8

Tabelle 2. ASCII-Code

nen. Außerdem sind im ASCII-Code die reversen Zeichen nicht enthalten, sondern müssen – wie Sie ja inzwischen wissen – jeweils umgeschaltet werden. Das alles ist für ein Betriebssystem viel zu kompliziert.

Es ist viel einfacher, im Festspeicher (ROM) alle Zeichen der zwei Zeichensätze fest zu verankern, von wo sie das Betriebssystem herausholen und auf den Bildschirm bringen kann.

Die Reihenfolge der Zeichen und ihr Code sehen Sie in der Tabelle 4. Sie ähnelt in mehreren Bereichen der ASCII-Reihenfolge, einige Spalten sind sogar identisch. Das macht eine Umrechnung – auch für das Betriebssystem – sehr einfach.

Folgende Blöcke der beiden Codearten entsprechen einander:

Ein Programm zur Umrechnung von ASCII-Code in Bildschirm-Code zeigt Listing 7.

Dabei habe ich als Variable gewählt:

– ACII = ASCII-Code

– Bild = Bildschirm-Code

In Bild 9 sind Bildschirm-Codes und entsprechende Funktionen nochmal aufgeführt:

Eine Zusammenfassung der Methoden, wie man in Basic Tasten abfragen kann.

Wir haben insgesamt vier Methoden kennengelernt und verwendet, um das Drücken einer Taste im Programm abzufragen:

1. Tastencode in Speicherzellen 203/653

```
10 A=PEEK(203)
```

```
20 B=PEEK(653)
```

```
30 IF A=ZEICHENCODE AND  
   B=STEUERCODE THEN  
   AKTION
```

2. ASCII-Code im Tastaturpuffer

```
100 POKE 198,0
```

```
110 A=PEEK(631)
```

```
120 IF A=ASCII-CODE THEN AKTION
```

3. Abfrage des ASCII-Codes mit GET/INPUT

```
200 GET A$
```

```
210 IF A$ <> CHR$(ASCII-CODE) THEN AKTION 1
```

```
220 AKTION 2
```

4. Abfrage des Gänsefußchen-Modus mit GET/INPUT

```
300 GET A$
```

```
310 IF A$ »ZEICHEN« THEN AKTION 1
```

```
320 AKTION 2
```

Diese vier Methoden haben alle eins gemeinsam:

Sie können immer nur eine einzelne Taste abfragen. Zwei oder gar mehrere Tasten gleichzeitig oder kurz hintereinander gedrückt ergeben keine sinnvollen Resultate.

Jetzt möchte ich Sie an meine allererste Darstellung (Bild 1 und 2) erinnern, nämlich wie die Tasten elektrisch angeordnet sind und wie das Betriebssystem sie abfragt. Ich möchte das hier noch einmal darstellen, erstens weil es eine gute Überleitung bildet zu meiner Methode der Vielfach-Tastenabfrage, zweitens weil diese Darstellung nicht vollständig war.

In Bild 10 ist noch einmal die VC 20-Tastenanordnung dargestellt, in Bild 11 diejenige des C 64. Trotz des Unterschiedes der elektrischen Anordnung ist bei beiden Computern die Abfragemethode identisch, nur die dafür benötigten Register haben unterschiedliche Adressen.

Zur Erinnerung:

Das Betriebssystem wählt der Reihe nach die senkrechten Spalten dadurch an, daß es die Zahl, die am Fuß jeder Spalte steht, in das Spalten-Register 37152 (beziehungsweise

BEDEUTUNG	ASCII-CODE	REVERSE DARSTELLUNG	FINKELN
LOCK (Sperrung der Zeichen- satz-Umschaltung)	8		H
UNLOCK (Sperrung aufheben)	9		I
RETURN	13		M
TEXT (2. Zeichen- satz)	14		N
DEL (Zeichen löschen)	20		T
SHIFT RETURN (Cursor auf Anfang der nächsten Zeile)	141		SHIFT M
GRAF (1. Zeichen- satz)	142		SHIFT N

Tabelle 3. Funktionen, die im Gänsefüßchen-Modus nur durch »Finkeln« eingetippt werden können

56320) schreibt. Diese Zahl ergibt in dualer Darstellung eine 0 an dieser Stelle.

Für jede Taste, die in der angewählten Spalte gedrückt ist, wird eine 0 in das andere Register 37153 (56321) geschrieben. Diese Dualzahl ergibt einen Dezimalwert, welcher aus dem Register herausPEEKbar ist.

Das, was das Betriebssystem macht, machen wir ihm nach, zuerst für den VC 20:

```
10 POKE 37152,247
20 PRINT PEEK (37152);PEEK(37153)
30 GOTO 10
und für den C 64:
10 POKE 56320,127
20 PRINT PEEK(56320);PEEK(56321)
30 GOTO 10
```

In Zeile 10 ist noch ein weiterer Unterschied zwischen den beiden Computern zu sehen. Beim VC 20 habe ich die Spaltenzahl 247 gewählt, weil in dieser Spalte die STOP-Taste liegt. Diese Spalte ist nämlich, wie wir ganz am Anfang ja schon festgestellt haben, die einzige Spalte, die wir mit Basic abfragen können. Bei POKEN der anderen sieben Spaltenzahlen in Zeile 10 wirft uns die 60mal in der Sekunde stattfindende Überprüfung der STOP-Taste aus dem Programm.

Beim C 64 ist das die Spalte 127, wie es uns ein Blick auf das Bild 11 zeigt.

Ich freue mich übrigens, daß ein aufmerksamer Leser aus Wien diesen Unterschied sofort bemerkt und mich darauf aufmerksam gemacht hat. Aber ich hatte damals nur einen VC 20 zur Verfügung, und mangels eigener Erprobung ist es mir nicht aufgefallen. Das hat sich übrigens jetzt geändert.

Diese Tastenabfrage hat den großen Vorteil, daß mehrere Tasten gleichzeitig drück- und abfragbar sind. Jede gedrückte Taste erzeugt eine 0 im »Reihen-Register«. Mit dem kleinen Programm oben können Sie es ausprobieren. Drücken Sie alle Tasten der Spalte 247 (127), die STOP-Taste bitte als letzte! Der rechte Zahlenstreifen auf dem Bildschirm zeigt die 0.

Lassen Sie die STOP-Taste (die 1-Taste) los, und es erscheint die 1, bei Loslassen der linken SHIFT-Taste (der --Taste) zusätzlich erhalten wir die 3. Das ist die Dezimalzahl für 00000011, die beiden 1er entstehen durch die losgelassenen Tasten.

Jede mögliche Tastenkombination in einer Spalte hat ihre

spezielle und abfragbare (!) Codezahl im Register 37153 (56321), insgesamt 256 Kombinationen. Ist das nichts?

Diese Methode der Mehrfach-Tastenabfrage haben wir bereits erfolgreich eingesetzt.

Nun möchte ich diese Methode auf alle acht Spalten, also auf alle Tasten erweitern. Dazu müssen wir das oben erwähnte Hindernis, nämlich den Rausschmiß durch das Betriebssystem, überwinden. Dazu gibt es zwei Methoden. Methode 1 will ich nur kurz erwähnen – sie ist einen eigenen Aufsatz wert.

Man kann durch Beeinflussung der Speicherzellen 788 und 789 die Pause (Interrupt) zur Tastenabfrage des Betriebssystems künstlich verlängern und sie zur eigenen Abfrage benutzen.

Die zweite Methode ist einfacher (Listing 8). Wir schreiben das Programm oben (Zeilen 10 und 20) in Maschinensprache. Das läuft dann so schnell ab, daß es innerhalb zweier Unterbrechungen ausgeführt und somit vom Betriebssystem nicht gestört wird.

Da ich nicht annehme, daß alle Leser dieses Kurses in Maschinencode programmieren können, zeige ich es Ihnen einfach als Kochrezept. Es besteht aus einer Reihe von Zahlen, die über READ ... DATA in vorbestimmte Speicherplätze gePOKEt und von dort dann mit SYS gestartet werden.

Für den VC 20 gilt:

```
100 DATA 169,254 ,141,32,145,173,33,145,141,255,
29,96
```

Für den C 64 gilt:

```
100 DATA 169, 254 ,141,0,220,173,1,220,141,255,
29,96
```

Ich habe einige Zahlen gekennzeichnet:

- Die zweite Zahl der DATA-Reihe in den Kästchen ist die Zahl der Spalte, die angewählt wird (ich habe 254 gewählt).
- Die 32 und 145 (0,220 beim C 64) ergeben die Registeradresse 37152 (56320), die DATA-Werte 33 und 145 (1,220) die Registeradresse 37153 (56321), die Werte 255,29 ergeben eine Speicherzelle 7679, auf die ich noch zurückkomme.

Die Registeradressen sind, wie immer bei Commodores 8-Bit-Computern, mit zwei Zahlen, das heißt mit 2 Byte dargestellt.

REGEL:

LSB + 256 * MSB = Adresse

32 + 256 * 145 = 37152

1 + 256 * 220 = 56321

LSB = niederwertiges Byte

MSB = höherwertiges Byte

Diese 12 DATA-Zahlen, ich nenne sie X, werden eingelesen mit:

```
110 FOR K=1 TO 12
```

```
120 READ X
```

```
140 NEXT K
```

Damit diese Zahlen vom Basic-Programm nicht überschrieben oder gelöscht werden (was dasselbe ist), POKEN wir sie an das Ende des VC 20-Speichers ohne Erweiterung, nämlich ab Speicherzelle 7661. Für den C 64 geht es mit denselben Adressen natürlich allemal.

```
130 POKE 7660+K,X
```

Um ganz sicher zu gehen, daß mit dem Maschinenprogramm nichts passiert, schützen wir es, indem wir dem Computer vorgaukeln, daß sein für Basic zur Verfügung stehender Speicher bei Adresse 7659 aufhört. Diese Speichergrenze steht in den Speicherzellen 51/52 und 55/56, und sie kann natürlich durch POKEN anderer Zahlen willkürlich verändert werden.

Die entsprechenden Werte für die Grenze 7659 sind 235 und 29. Machen wir die Probe:

PRINT 235+256+29 (RETURN)
ergibt 7659.

20 POKE 51,235
30 POKE 52,29
40 POKE 55,235
50 POKE 56,29

Falls Sie diese Speicherzellen und ihre Anwendung nicht kennen, bitte ich um Zuschrift, da ich es hier aus Platzgründen nicht näher erläutern kann.

So, nun ist das Maschinenprogramm gespeichert und gesichert. Wir starten das Programm mit SYS-Aufruf der 1. Adresse und PEEKen danach das Register 37153 (56321), genau wie vorher.

150 SYS 7661
160 PRINT PEEK(37153): REM VC 20
160 PRINT PEEK(56321): REM C 64
170 GOTO 150

Nach RUN erzeugt der Rücksprung den schon obligatorischen Zahlenstreifen mit 255 für »keine Taste gedrückt«. Wenn wir die Taste 3 (RETURN beim C 64) drücken, dann muß entsprechend Bild 10 und 11 (Spalte 254) die Zahl 253 erscheinen. Das tut sie auch, aber immer wieder unterbrochen durch 255. Das kommt daher, daß das Auslesen des Registers 37153 (56321) in Basic doch schon zu langsam ist. Wir müssen daher die Funktion der Zeile 160 ebenfalls in das Maschinenprogramm einbauen, was ich in weiser Voraussicht in Zeile 100 schon vorbereitet habe. Der Trick ist nämlich dabei, daß der Inhalt des Registers vom Maschinenprogramm in eine vor dem bösen Basic geschützte Speicheradresse gebracht wird. Da wir ja einen gesicherten Bereich bereits haben, habe ich die Zelle 7679 ausgewählt.

Bitte ändern Sie die Zeile 160 ab:

160 PRINT PEEK(7679)

Jetzt haben wir's geschafft.

Natürlich können wir mit diesem Programm alle acht Spalten abfragen. Wir brauchen bloß in Zeile 100 die Zahl, die hier im Kästchen steht, abändern, zum Beispiel in 253, und schon reagiert das Programm auf alle Tasten dieser Spalte.

Ich habe mir den Luxus erlaubt und unser Programm von oben erweitert, indem ich der Reihe nach alle acht Spalten aufrufe (siehe Listing 1). Dazu habe ich in Zeile 100 die Spaltennummer auf 0 gesetzt, was eigentlich unnötig aber einleuchtend ist, POKE aber dann jeweils die Spaltenzahl (in den Zeilen 200, 300, 400 etc.) auf diesen Platz, der die Adresse 7662 hat. Das Programm ist immer noch schnell genug, um innerhalb der zur Verfügung stehenden Zeit von 1/60 Sekunde alle Abfragen durchzuführen.

Ein Appell an alle 64er: Wenn Sie die Semikolons in den Zeilen 220, 320, 420 etc. (außer 920!!!) richtig plaziert haben, erscheinen die Zahlen für alle acht Spalten brav nebeneinander.

Beim VC 20 haben wir ein kleines Problem, hervorgerufen durch die kleinere Zeilenlänge, die uns eine Darstellung aller acht Zahlen nebeneinander nicht erlaubt.

Ich schlage deshalb vor, nach vier Zahlen ab Zeile 520 die Reihenfolge zu unterbrechen (Semikolon weglassen) und an den Anfang (Zeile 200) zurückzuspringen, es sei denn, die f-1-Taste wurde gedrückt (Zeile 560). Dann nämlich kommen die nächsten vier Zahlen dran – sogar in einer anderen Farbe. Eine weitere Tastenabfrage in Zeile 960, diesmal derTaste – schauen Sie in der ASCII-Tabelle nach, welche Taste dieses Zeichen hat (Methode des waagrechten Sprunges zwei Spalten weiter) – entscheidet zwischen Wiederholung der zweiten Gruppe oder Rücksprung auf die ersten vier Zahlen. Wie gesagt, die Verwendung der Semikolons an der richtigen Stelle ist wichtig.

Jetzt ist die Gelegenheit da zum Experimentieren:

- mehrere Tasten aus verschiedenen Spalten
- mehrere Tasten aus derselben Spalte.

Tabelle 4. Bildschirm-Code

Ja, und irgendwann bleibt Ihnen das Programm mit BREAK IN stehen!!

Leider ist das schon wieder der STOP-Tasten-Effekt, den wir nicht ganz loswerden. Wenn nämlich Tasten der waag-rechten Reihe 254 (beziehungsweise 127) gedrückt wer-den, interpretiert dies das Betriebssystem als STOP-Taste, die ja in dieser Reihe liegt, und stoppt das Programm.

Uns soll das aber nicht bedrücken, denn wir können in unserem Programm den Gebrauch dieser Tastenreihe ein-fach vermeiden. Für unsere Zwecke reichen alle anderen Tasten allemal aus. Man muß diese Einschränkung nur berücksichtigen.

Zusammenfassend sei gesagt, daß Sie in einem Programm jetzt alle Kombinationen aller Tasten abfragen können mit
IF PEEK(7679)= THEN

Sie müssen sich lediglich die entstehenden Dualzahlen im Register 37153 (56321) in Dezimalzahlen umrechnen oder sie vorher ausprobieren.

Ich will auch am Schluß meiner Darstellung der Gewohnheit treu bleiben und das Kochrezept in einem kleinen Gericht anwenden (Listing 9).

Ein Spiel für vier Personen

Um mir und Ihnen die Sache leichter zu machen, verwende ich möglichst viele Teile aus den vorherigen Programmen.

Spielidee:

- Jeder Spieler wählt eine von vier bestimmten Tasten.
- Auf dem Bildschirm wird ein beliebiger Buchstabe »angesagt«
- Der Bildschirm füllt sich langsam mit Zufalls-Buchstaben
- Sobald der »angesagte« Buchstabe erscheint, sollen die Spieler ihre Taste drücken
- Wer zuerst drückt, hat gewonnen!

Programmbeschreibung:

Zeile 10 bis 50

schützt das Maschinenprogramm (wie vorher)

Zeile 100 bis 140

liest das Kochrezept »Maschinenprogramm« ein

Zeile 215 und Zeile 530

stellen einen besonderen Trick dar. Man nennt das eine »Flagge setzen« (set a flag). Die Abfrage der Tasten soll näm-lich nur dann funktionieren, wenn ein Buchstabe (später VB genannt) mit dem anfangs »angesagten« Buchstaben BU übereinstimmt. Wenn VB=BU, dann wird in eine »sichere« Speicherzelle (ich habe 830 gewählt) eine Zahl (hier 250) hineingePOKEt. Diese Zahl in 830 wird bei der Abfrage ebenfalls abgefragt. Bei Beginn des Spieles muß diese Flagge natürlich eingeholt, das heißt auf 0 gesetzt werden (Zeile 215).

Zeile 220 bis 270

gibt Anweisungen und teilt den Spielern ihre Taste zu (von mir willkürlich ausgewählt)

Zeile 280/300

löscht den Bildschirm und färbt alles schwarz

Zeile 310/320

gibt Anweisungen

Zeile 330

wählt per Zufallsgenerator (wie vorher) einen Buchstaben BU aus; zwischen 1 und 10 (A bis J in Bildschirm-Code!)

Zeile 340/350

drückt diesen Buchstaben BU genau an das Ende des Textes der Zeile 320 (Bildschirmspeicher-Platz und Farbspeicher-Platz ausrechnen!)

Zeile 260/270 und 370/380

schalten den Text weiter mit irgendeiner Taste

Zeile 400

verzögert das Erscheinen des 1. Buchstabens

Zeile 520

wählt wieder per Zufallsgenerator einen Buchstaben, dies-mal VB, aus (zwischen A und J)

Zeile 540

wählt per Zufallsgenerator einen Platz C auf dem Bildschirm aus, auf den der Buchstabe VB gePOKEt wird

Beim VC 20 fülle ich den ganzen Bildschirm von Platz 0 bis 505. Beim C 64 wäre das der Bereich von 0 bis 1000. Damit aber den Spielern wegen der viel kleineren Buchstaben des C 64 die Augen nicht übergehen, habe ich die eigentliche Formel $C = \text{INT}(\text{RND}(0) * 1000)$ in Zeile 540 so abgeändert, daß die Buchstaben nur auf jeden 2. Platz gePOKEt werden können.

Zeile 550/560

die Buchstaben VB werden in Weiß gePOKEt

Zeile 610

leierte die Tastenabfrage 15mal hintereinander durch. Diese Zahl bestimmt auch die Geschwindigkeit, mit der die Buch-staben auf dem Bildschirm erscheinen. Ihre Verkleinerung erhöht die Schwierigkeit des Spieles.

Zeile 620 bis 740

fragt die in Zeile 220 bis 250 definierten Tasten und die Flagge in 830 ab. Beim VC 20 ist jede der gewählten Tasten in einer anderen Spalte, daher sind vier POKes in 7662 not-wendig. Beim C 64 liegen je zwei Tasten in einer Spalte, daher nur zwei POKes in 7662. Da aber gleichzeitige Tastendrücke vorkommen können (in einer Spalte natürlich), fragen Zeile 660 und 710 diesen speziellen Fall ab.

810 bis 840 beziehungsweise 850

meldet den Sieger (Aussprung aus der Abfrage)

Zusammenfassung

Diese »gleichzeitige« Mehrfach-Tastenabfrage ist genau genommen nicht ganz gleichzeitig.

Nichts in einem einzelnen Computer ist gleichzeitig! Alles erfolgt in einer Sequenz.

In dem Spielprogramm für vier Personen erfolgt die Tasten-abfrage natürlich auch hintereinander, wodurch die zuerst abgefragten Tasten ihren Besitzern einen kleinen Vorteil gewähren, aber halt nur einen ganz winzigen!

Dieser Effekt wird durch das 15fache Durchlaufen der Abfrage und durch die Schnelligkeit des Maschinenprogram-mes gemildert. Das ganze Programm in Maschinensprache geschrieben, würde natürlich durch die Geschwindigkeit auch die letzte Ungerechtigkeit ausmerzen.

So, liebe VC 20er und C 64er. Das war's.

Ich hoffe, Sie kennen jetzt die verschiedenen Codes und fühlen sich wohl bei der Tasten-Abfrage. Methoden und Kochrezepte haben Sie dazu ja jetzt genug.

Ich habe mich bemüht, Sie zum Experimentieren anzure-gen, denn gerade diese spielhafte Auswirkung der Neugier unterscheidet den Amateur vom Profi und fördert das Ver-ständnis der Zusammenhänge.

Ich habe mir am Anfang das Ziel gesetzt, allgemeinver-ständlich und ohne Fachchinesisch zu schreiben.

Ob das gelungen ist, können nur Sie mir sagen. Falls etwas unklar geblieben ist und Sie Fragen haben, schicken Sie sie mir, ich werde Ihnen antworten. Und wenn es sogenannte »gute Fragen« sind, schreibe ich vielleicht darüber den näch-sten Kurs.

(Dr. Helmuth Hauck/cg)

Literatur:

1. VIC REVEALED von N. Hampshire, Computabits Ltd., 1981
2. M. Bassman, S. Lederman in COMPUTE!S FIRST BOOK OF VIC COMPUTE!, Books Publ., 1982
3. VC 20 SPIELE-BUCH 1 von A. Dripke, Computer Life Verlag, 1983
4. VC-INTERN von Angerhausen und Englisch, Data Becker, 1983
5. 64-INTERN von Angerhausen et al., Data Becker, 1983
6. DAS VC-20-Buch von M. Hegenbarth, M. Schäfer, Markt & Technik Verlag, 1983
7. VIC-20-PROGRAMMERS REFERENCE GUIDE von A. Finkel et al., Howard W. Sams & Co, 1982