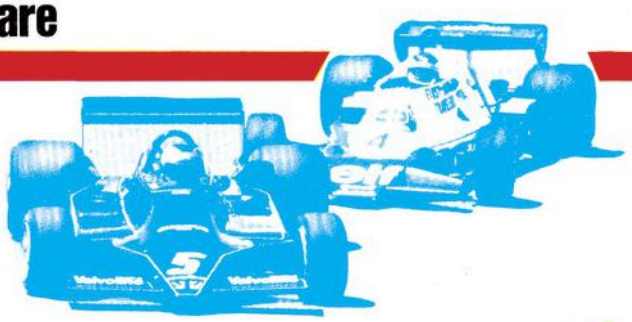

So macht man Basic-Programm

Basic-Programme können nach drei Gesichtspunkten optimiert werden: strukturiert und lesbar, schnell sowie speicherplatzsparend. Nach der Serie über strukturiertes Programmieren wird in diesem Beitrag der Aspekt der Laufgeschwindigkeit und ihrer Verbesserung behandelt. Alle Aussagen und Beispiele gelten sowohl für den VC 20 als auch den C 64.

Laufzeiten der Programmversionen für »Buchstaben auf dem Bildschirm«

Version Nr.	Programmier-Methode	Laufzeit (Sek.) VC 20	C 64
1	Buchstaben POKEn,Zählschleife mit IF ... THEN	8,25	10,48
2	POKE-Adressen vordefinieren	5,78	7,30
3	alle Variablen vordefinieren	5,15	6,15
4	Variablen mit 2 Buchstaben	4,71	5,63
5	Variable mit 1 Buchstaben	4,63	5,51
6	PRINT CHR $ statt POKE	3,40	4,05
7	CHR $ vordefinieren	2,58	3,10
8	PRINT "A"	2,53	3,03
9	Schleife mit IF-GOTO statt IF-THEN	2,51	3,00
10	IF Z<>S statt IF Z = S	2,50	2,98
11	Schleife mit FOR-NEXT	0,91	1,08
12	NEXT ohne Zählvariable	0,83	1,00
13	Listing mit REMs	0,98	1,20
14	Listings ohne REM, ohne Abstände	0,81	0,98
15	Alles in einer Zeile	0,78	0,93
16	Maschinensprache	0,033	0,066

Es gibt einige Dinge auf der Welt, die man sehr wohl einzeln, aber nicht alle gleichzeitig haben kann. Wirtschaftswachstum, keine Inflation und niedrige Arbeitslosigkeit lassen sich eben nicht unter einen Hut bringen.

Bei der Computerei, oder genauer gesagt beim Prorammieren, gibt es in einer höheren Sprache wie Basic ebenfalls so ein magisches Dreieck: strukturierte Programme, minimaler Speicherbedarf, kürzere Laufzeiten.

Gut lesbare und klar gegliederte Programme brauchen oft mehr Speicherplatz als es sich zum Beispiel Besitzer der VC 20-Grundversion leisten können. Deswegen soll mein heutiger Beitrag Sie zum Expe-

...e schneller

rimentieren anregen, mit welchen verschiedenen Methoden Basic-Programme schneller gemacht werden können.

Basic ist nicht immer gleich Basic

Die Commodore-Handbücher sagen leider zu diesem Thema recht wenig. Ich bitte Sie an Ihrem Computer, VC 20 oder C 64, Platz zu nehmen.

Als erstes wollen wir uns einen einfachen aber typischen Programmablauf überlegen, welchen wir mit mehreren Basic-Möglichkeiten programmieren können. Übrigens: Wegen der guten Lesbarkeit schreiben Sie die nachfolgenden Basic-Zeilen mit Leerstellen zwischen den Befehlen.

Ich weiß, es geht auch kürzer, aber bei meinen Experimenten spielt Speicherplatz keine Rolle und außerdem habe ich noch einen Hintergedanken, den ich erst später erklären kann. Nicht zuletzt will ich dadurch auch erreichen, daß die Laufzeiten der Programme mit den Ihren übereinstimmen.

Schalten Sie bitte auch Programmierhilfen (Toolkit, Simon's Basic etc.) und Disk Operating-Programme (DOS 5.1) aus, denn sie verlangsamen den Programmablauf.

Die interne Uhr mißt die Zeit

Beide Computer, VC 20 und C 64, haben eine interne Uhr, deren Zeit abgefragt, ausgedruckt und somit zur Zeitmessung verwendet werden kann. Im Befehlssatz der Commodore-Handbücher finden Sie dazu die beiden Funktionen TI und TI$. Mein Zeitmessungsprogramm besteht aus zwei Zeilen. Die »Stoppuhr« wird gestartet mit:

```
10 TI$="000000"
```

Sie wird am Ende des Testprogramms gestoppt mit:

```
1000 PRINT TI/60 "SEKUNDEN" :END
```

Zwischen diese beiden Zeilen stecken wir dann die Prüflinge, das heißt die Programme, deren Laufzeit wir messen wollen.

Zur Prüfung geben wir zuerst die Zeile 15 mit einer simplen Verzögerungsschleife ein.

```
15 FOR T=1 TO 100:NEXT T
```

Das Programm dieser drei Zeilen hat die Laufzeit von 0,13 Sekunden mit dem VC 20 und 0,15 Sekunden mit dem C 64.

Das erste Testprogramm

Löschen Sie bitte wieder die Zeile 15. Als Programm, welches wir in mehreren Versionen programmieren wollen, habe ich mir einen Ablauf ausgesucht, der auch optisch verfolgbar ist. Auf dem Bildschirm soll nämlich der Buchstabe A gleich 374 mal nebeneinander gedruckt werden.

Die Zahl 374 hat nichts Magisches an sich. Es sind ganz einfach 17 Zeilen voller A's auf dem Schirm des VC 20, der mit seiner begrenzten Spaltenzahl hier den Ton angibt. 17 Zeilen lassen uns genug Platz, um die gestoppte Zeit darunter gut lesbar anzuzeigen.

Ich bleibe für den C 64 bei derselben Zahl, damit wir beide Computer miteinander vergleichen können und damit die Programme möglichst identisch sind.

In der **Version 1** des Programms verwenden wir POKE-Befehle, mit denen wir das A und die dazugehörige Farbe auf den Bildschirm, das heißt in den Bildschirmspeicher (Video-RAM) und den Farbspeicher (Color-RAM) bringen. Beim VC 20 (ohne Erweiterung) beginnt der Bildschirmspeicher ab Speicherzelle 7680, der Farbspeicher ab 38400. Beim C 64 sind es die Speicherzellen 1024 und 55296. Schauen Sie bitte in den Commodore-Handbüchern nach und vergewissern Sie sich, daß Sie dieses System des Zeichen-POKEns verstehen. Es ist dort gut beschrieben. Der Buchstabe A hat den Bildschirm-Codewert 1. Als Farbe wähle ich die »Normalfarben« der beiden Computer. Der Farbcode für das Blau des VC 20 ist 6, für das Hellblau des C 64 ist er 14.

Das Programm beginnt mit dem Löschen des Bildschirms (Zeile 20) und POKEt dann in Zeile 40 das erste A in den ersten Platz des Bildschirms.

```
20 PRINT CHR$ (147)
40 POKE 7680+Z:POKE 38400+Z,6
(40 POKE 1024+Z:POKE 55296+Z,14)
```

Die Zeile in Klammern gilt für den C 64. In Zeile 40 finden Sie zusätzlich eine Variable »Z«. Wie geplant, soll das A 374mal gePOKEt werden. Also müssen wie die Zahl der Speicherzelle laufend um 1 erhöhen. Dazu erfinden wir diese Variable Z, die zur Speicherzelle addiert wird.

Wir setzen Z am Anfang des Programms (in Zeile 30) auf Null und zählen es in Zeile 50 um 1 weiter.

```
30 Z=0
50 Z=Z+1
```

Als nächstes müssen wir prüfen, ob Z den Schlußwert 374 erreicht hat. Wenn nicht, dann soll der nächste POKE-Befehl ausgeführt werden, das heißt wir springen auf Zeile 40 zurück. Dann kommt die Zeile 1000 zum Zug mit dem Stoppen und Ausdrucken der Laufzeit. Also:

```
60 IF Z=374 THEN 1000
70 GOTO 40
```

Ich schlage vor, daß Sie das kleine Programm nochmal LISTen, damit wir es komplett sehen können.

Die Laufzeit wird nur davon beeinflußt, was zwischen den Zeilen 10 und 1000 steht. Sie können vor der Zeile 10 dem Programm hinzufügen, was Sie wollen.

Ein erster Probelauf mit RUN bringt das gewünschte Ergebnis, nur eins ist noch unschön: Der PRINT-Befehl in Zeile 1000 druckt uns die Zeit oben in die 2. Zeile, wo sie schlecht erkennbar ist. Wir könnten sie in einer anderen Farbe drucken, aber ich habe einen besseren Vorschlag.

Ersatz für den Befehl »PRINT-AT«

Wir brauchen ein Kochrezept, um mit einem PRINT-Befehl an einen ganz bestimmten Platz auf dem Bildschirm drucken zu können. Einige Basic-Dialekte kennen den Befehl »PRINT-AT«. Welche Möglichkeiten bietet uns das Basic von Commodore?

1) PRINT"[Cursor Down] [Cursor Right]"
2) PRINT TAB(X)
3) PRINT SPC(X)

Um zum Beispiel an den 3. Platz in der 20. Zeile die Zeit zu drucken, müßten wir 18mal das inverse Q für Cursor Down und 1mal Cursor Right eingeben.

Mit TAB (X) geht es besser. Wir haben nur ein Problem, daß nämlich der höchste zulässige Wert für X nur 255 ist (X nennt man das »Argument«). Wir müssen deshalb zwei TAB-Befehle hintereinander setzen, um einen Abstand von 400 Leerstellen zu erzeugen.

```
1000 PRINT TAB (255) TAB (155) TI/60"SEKUNDEN":END
```

Für das Argument von SPC gilt dieselbe Begrenzung von 255. Eine doppelte Verwendung von SPC geht natürlich auch, allerdings zählt SPC nicht vom Anfang der Zeile, sondern ab der letzten Cursor-Stelle. Durch Rechnen oder einfach durch Probieren finden wir die Gesamtzahl von 397, das gibt:

```
1000 PRINT SPC (255) SPC (142) TI/60"SEKUNDEN":END
```

Es gibt noch eine dritte Methode, um PRINT-AT zu simulieren.

In die Speicherzelle 214 kann die Zahl einer Zeile hineingePOKEt werden, auf die mit einem nachfolgenden PRINT der Cursor gesetzt wird. Das gleiche gilt für einen Platz in einer Zeile mit der Speicherzelle 211. Versuchen Sie es mit der direkten Eingabe:

```
POKE 214,8:PRINT:POKE 211,4:PRINT"A"
```

Drei Formen für »PRINT AT«

Das druckt den Buchstaben A in die 4. Spalte auf der 9. Zeile. Für unseren Anwendungsfall in Zeile 1000 müssen wir die Zahl 18 nach 214 POKEn, 211 können wir vernachlässigen.

```
1000 POKE 214,18:PRINT:PRINT TI/60 "SEKUNDEN":END
```

Alle drei Methoden sind gleichwertig, sowohl in Auswirkung als auch beim Speicherbedarf. Ich bleibe im folgenden bei der 214-Methode.

Zurück zur Version 1 des Testprogramms. Das Programm unterscheidet sich für die beiden Computer nur in der Zeile 40, allerdings auch durch die Laufzeit. Nach RUN erhalten wir mit dem VC 20 8,25 Sekunden, mit dem C 64 10,48 Sekunden. Dieses Auffüllen des Bildschirms mit A geht halt recht langsam. Schon beim Zuschauen wird man ungeduldig. Der Teil in diesem Programm, welcher die Laufzeit am nachhaltigsten beeinflußt, ist die 374fache Wiederholung des POKE-Befehls. Bei einem POKE-Befehl ist die Umwandlung der Zahlen aus dem ASCII-Code sehr zeitaufwendig. Hoppla, was heißt denn das schon wieder, sagen Sie jetzt vielleicht.

Jede Zahl, wie zum Beispiel 7680 oder 1024, wird zuerst als vier einzelne ASCII-Codezahlen gespeichert. Wenn das Programm abläuft, werden diese ASCII-Zahlen zuerst in ganze Zahlen, dann in Fließkomma-Zahlen umgewandelt — das für den Fall, daß mit den Zahlen arithmetische Funktionen ausgeführt werden. Schließlich werden sie wieder in eine ganzzahlige POKE-Adresse umgewandelt, und das in unserem Fall 374mal!

Hier können wir einen ersten innerbetrieblichen Verbesserungsvorschlag einreichen. Wenn wir eine so häufig vorkommende Zahl wie die POKE-Adresse in Zeile 40 am Anfang des Programms einer Variablen zuweisen, dann erfolgt die oben genannte Umwandlungssequenz nur einmal, nämlich am Anfang des Programms. Das Programm muß dann 374 mal nur den Wert der Variablen im Speicher suchen und das geht viel schneller. Wollen Sie es sehen? Ändern Sie bitte für diese **Version 2** die Zeilen 30 und 40.

Die Anfangsadresse im Bildschirmspeicher definieren wir als Variable mit dem schönen und zutreffenden Namen »VIDEO«, die des Farbspeichers mit »FARBE«. In Zeile 30 erhalten sie dann die Werte

```
30 Z=0:VIDEO=7680:FARBE=38400:REM VC 20
(30 Z=0:VIDEO=1024:FARBE=55296:REM C 64)
```

Zeile 40 ergibt sich dann eigentlich zwangsläufig:

```
40 POKE VIDEO+Z, 1:POKE FARBE+Z,6
(40 POKE VIDEO+Z,1:POKE FARBE+Z,14)
```

Alles andere bleibt gleich. Tippen Sie RUN ein. Ergebnis: der VC 20 braucht 5,78 Sekunden, der C 64 7,3 Sekunden. Wir haben also eine Verkürzung von zirka drei Sekunden erzielt, das sind 30 Prozent!

Sicher ist Ihnen aufgefallen, daß der VC 20 schon wieder schneller ist als der C 64.

In der Tat können die oft mitleidig behandelten VC 20-Besitzer mächtig stolz sein: Der VC 20 ist immer schneller als der C 64 und auch (fast immer) schneller als die »Neuen« C-116 und Plus/4 (und übrigens einer der schnellsten Heimcomputer überhaupt).

Der VC 20 ist der schnellste

Eine Beschleunigung von 30 Prozent ist gut, aber noch nicht alles, was wir erreichen können. Da die Methode der vordefinierten Variablen so effektiv ist, wollen wir sie auf alle oft verwendeten Zahlen des Programms anwenden. Neben Z, VIDEO und FARBE gibt es noch die 1 für den Buchstaben A und die 6 (14) für die Farbe sowie den Schlußwert 374 der Schleife.

Sie wissen schon, wie das geht. Wir ändern folgende Zeilen:

```
30 Z=0: SCHLUSSWERT=374:VIDEO=7680:FARBE=38400:BUCHSTA=1:DRUCK=6
(30 Z=0: SCHLUSSWERT=374:VIDEO=1024:FARBE=55296:BUCHSTA=1:DRUCK=14)
40 POKE VIDEO+Z, BUCHSTA:POKE FARBE+Z,DRUCK
60 IF Z=SCHLUSSWERT THEN 1000
```

Das ist **Version 3** des Programms, mit Laufzeiten von 5,15 Sekunden (VC 20) beziehungsweise 6,15 Sekunden (C 64). Das ist eine Verbesserung von 3,10 (4,23) Sekunden gegenüber der ersten Version.

Bisher haben wir die Variablen im Sinn einer guten Lesbarkeit mit langen und verständlichen Namen versehen. Aber das kostet natürlich Speicherplatz und auch Geschwindigkeit. Der Grund ist immer derselbe: Bei der Variablen VIDEO sind fünf Zeichen 374 mal zu bearbeiten. Wenn wir sie nur VI nennen, ist das erheblich weniger.

In **Version 4** des Programms reduzieren wir also alle langen Variablennamen auf zwei Zeichen. Wir wollen mal schauen, was das bringt:

```
30 Z=0: SC=374: VI=7680:FA=38400:BU=1:DR=6
(30 Z=0: SC=374: VI=1024:FA=55296:BU=1:DR=14)
40 POKE VI+Z,BU:POKE FA+Z,DR
60 IF Z=SC THEN 1000
```

Ergebnis: VC 20: 4,71 Sekunden
C 64: 5,63 Sekunden

Gegenüber der Version 1 des Programms haben wir schon eine Verbesserung von 43% (46%) erreicht.

Sie wissen sicher, daß der Computer alle Variablennamen immer auf zwei Stellen reduziert, daß aber einstellige Variablen durchaus zu-

gelassen sind. Damit müßte unser Programm eigentlich noch schneller werden. Die folgenden Änderungen für **Version 5** zielen genau darauf:

```
30 Z=0: S=374: V=7680:F=38400:
B=1:D=6
(30 S=374: V=1024: F=55296:B=1:
D=16)
40 POKE V+Z,B:POKE F+Z,D
60 IF Z=S THEN 1000
```

Die erzielte Verbesserung ist meßbar, aber nicht gerade überwältigend. Der VC 20 braucht nun 4,63 und der C 64 5,51 Sekunden. Mit dem bisher Gesagten läßt sich bereits eine erste generelle Regel für schnellere Basic-Programme aufstellen.

Regel 1
* Häufig vorkommende Zahlen, Adressen und Variable, besonders innerhalb von Schleifen, werden am Anfang des Programms vordefiniert.
* Variable, die sich im Lauf des Programms verändern, werden trotzdem vordefiniert, allerdings mit einem unschädlichen Anfangswert (dummy).
* Die Zahl, die am häufigsten vorkommt, wird als erste vordefiniert (damit sie schneller »gefunden« wird).
* Variablennamen sollen möglichst einstellig, aber höchstens zweistellig sein.

Halt! Löschen Sie das Programm noch nicht. Wir liegen noch unter 50 Prozent mit unseren Verbesserungen, und das reicht noch lange nicht. Was können wir am bisherigen Programm noch ändern? Denken Sie mal nach. Wie kann man Buchstaben auf den Bildschirm bringen? Natürlich, mit PRINT statt der POKEs.

In **Version 6** des Programms ersetzen wir die Buchstaben-POKE-Befehle durch eine PRINT-Anweisung, die prinzipiell noch den Vorteil hat, daß keine Farbanweisung nötig ist. Den Buchstaben A wollen wir in dieser Version zunächst einmal mit seiner ASCII-Codezahl 65 angeben, ein Semikolon setzt alle A hintereinander.

```
40 PRINT CHR$(65);
```

Für die Schleife brauchen wir nur die beiden Variablen Z und S:

```
30 Z=0:S=374
```

Alle anderen Zeilen bleiben unverändert.
Nach RUN sehen wir als Ergebnis:
VC 20: 3,4 Sekunden
C 64: 4,05 Sekunden

Das ergibt eine weitere Verbesserung von 1,2 (1,5) Sekunden. Unser Programmbeispiel wird also durch die Verwendung von PRINT statt POKE stark beschleunigt. Das geht natürlich deswegen besonders gut, weil alle Buchstaben automatisch hintereinander gesetzt werden. Wenn wir jedesmal den Platz mit angeben müßten, wohin gePRINTet werden soll, wäre der Vorteil rasch verspielt.

Die Anweisung PRINT CHR$(65) ist zwar gut lesbar, aber wir haben ja vorher gelernt, daß Vordefinieren von Variablen schneller ist. In **Version7** machen wir das auch mit der PRINT-Variablen.

```
30 Z=0: S=374: A$=CHR$(65)
40 PRINT A$
```

Das Resultat ist wieder beeindruckend: 2,58 Sekunden beim VC 20, 3,10 Sekunden beim C 64. Das sind schon 69 Prozent Verbesserung gegenüber der 1. Version.

Aber selbst — oder gerade — jedem Anfänger ist die direkte PRINT-Anweisung mit Gänsefüßen am geläufigsten. Sie braucht auch weniger Speicherplatz und macht sogar ein Vordefinieren unnötig. Wir wollen schauen, ob sie auch schneller ist.

Ändern Sie für **Version 8** die Zeilen 30 und 40:

```
30 Z=0:S=374
40 PRINT "A";
```

Erstaunlicherweise bringt das fast gar nichts, beim VC 20 nur 0,05 Sekunden Verbesserung, beim C 64 0,07 Sekunden. Die Erklärung liegt darin, daß beide Darstellungen, CHR$(65) und "A" ASCII-Code-Verwender sind. Damit ist der einzige Unterschied zwischen Version 7 und 8 die Anzahl der Zeichen im Programm.

Fassen wir zusammen:

Regel 2
* In Schleifen mit aneinandergereihten Druckanweisungen ist PRINT viel schneller als POKE.
* Die PRINT-Variablen sollen entweder vordefiniert oder im Gänsefuß-Modus eingesetzt werden.

In der PRINT-Schleife (Zeile 40) und nachfolgender IF-Abfrage (Zeile 60) gibt es noch zwei Feinheiten. Basic erlaubt uns bei bedingten Sprüngen statt IF .. THEN GOTO .. nur IF .. THEN .. zu schreiben und das haben wir bisher auch brav gemacht.

Man kann aber auch IF .. GOTO verwenden. Eigentlich ist nicht zu erwarten, daß zwischen den beiden Schreibarten ein Zeitunterschied besteht. Der Fall ist tatsächlich fast akademisch, wie **Version 9** beweist:

```
60 IF Z=S GOTO 1000
```

Laufzeit des VC 20: 2,51 Sekunden, die des C 64: 3,0 Sekunden.

Eine andere Änderung bringt auch nur ganz wenig in der Geschwindigkeit, aber sie spart uns eine ganze Zeile und damit Speicherplatz:

```
60 IF Z<>S GOTO 40
70 entfällt
```

Diese **Version 10** gewinnt nur 0,01 (0,02) Sekunden.

Regel 3
* Bei Schleifen mit Sprunganweisungen ist IF .. GOTO schneller als IF..THEN
* Prüfung auf Ungleichheit (<>) bietet Vorteile, wenn sie eine Zeile erspart.

Für die etwas weiter Fortgeschrittenen unter Ihnen gebe ich hier noch eine weitere Anregung, die wir mit unserem Prüfprogramm nicht testen können.

Die Prüfung mit IF..THEN hängt oft von mehr als einer Bedingung ab. Zum Beispiel kann sie so lauten:

```
100 IF (A=1 AND B=2 AND C=3)
THEN 999
110 GOTO 50
```

Zeile 100 prüft jedesmal, ob alle drei Bedingungen erfüllt sind, erst nach dem THEN wird entschieden, ob das Programm auf Zeile 999 springt oder auf 110 weiterläuft. Nehmen wir an, A ist im 20. Durchlauf erfüllt, B im 50. Durchlauf, C aber erst im 300. Durchlauf. Das Programm muß also 300mal alle drei Bedingungen nachprüfen. Wenn wir die Zeile 100 so schreiben:

```
100 IF C=3 THEN IF B=2 THEN IF
A=1 THEN 999
```

dann bricht das Programm 300mal die Prüfung nach dem C sofort ab und geht in 110 weiter. B und A werden erst dann zur Prüfung herangezogen, wenn C=3 ist. Es ist wohl klar und einzusehen, daß die zweite Schreibweise schneller ist.

Regel 4
Bei IF..THEN-Prüfungen mit mehreren Bedingungen sollen diese Bedingungen in einzelnen IF..THEN-Prüfungen hintereinander gesetzt werden. Dabei wird die Bedingung an die erste Stelle gesetzt, welche als erste nicht erfüllt ist.

Ich habe mit Absicht bis hierher die 374malige Wiederholung der Schleife mit der Zählvariablen Z hochgezählt und mit IF..THEN beziehungsweise IF.GOTO den Schlußwert abgefragt. Das gab mir die Gelegenheit, die Schnelligkeit dieser Methode ganz auszureizen. Und wir haben auch die ursprüngliche Laufzeit von 8,25 (10,48) Sekunden auf 2,50 (2,88) Sekunden reduziert!

Jetzt wollen wir noch eine andere Basic-Möglichkeit austesten, die Sie ja alle kennen, nämlich die Schleife mit FOR..TO..NEXT zu programmmieren.

Wir wollen in **Version 11** des immer noch gleichen Programms dazu die Zeilen 30, 50 und 60 ändern:

```
30 FOR Z=1 TO 374
50 NEXT Z
60 entfällt
```

Lassen Sie's laufen. Huiiih! Das pfeift runter! Der VC 20 meldet 0,91 Sekunden, der C 64 1,08 Sekunden. Das bringt gegenüber der IF..THEN-Schleife der Version 10 eine ganze Menge, nämlich ungefähr 2 Sekunden, oder, auf die Zeit der Version 1 bezogen, 89 Prozent.

Das einzige, was wir in der FOR..NEXT-Schleife noch verbessern können, ist die vielgeübte Praxis des Weglassens der Schleifenvariable Z nach dem NEXT-Befehl.

50 NEXT

Der Geschwindigkeitsgewinn dieser **Version 12** ist nicht sehr groß, nämlich nur etwa 0,1 Sekunden, aber wir wollen die Methode doch in die nächste Regel mit aufnehmen.

Regel 5
Schleifen sollen nicht mit IF..THEN, sondern mit FOR..TO..NEXT gebildet werden. Die Schleifenvarible nach NEXT soll dann weggelassen werden, wenn es nicht zu Verwechslungen mit anderen Schleifen führen kann.

Ich bin fast am Ende meines Beschleunigungslateins. Nur eines bleibt noch, nämlich die bisher hochgehaltene Lesbarkeit des Programms zu opfern. Ich hoffe nämlich, Sie haben bisher meiner Eingangsforderung Folge geleistet und alles schön mit Leerzeichen geschrieben. Das behalten wir zunächst noch bei, im Gegenteil, wir wollen zunächst die Lesbarkeit noch erhöhen und REM-Erläuterungen einfügen. Ich schlage vor, die **Version 13** so auszuschmücken:

```
  10 TI$="000000":REM UHR AUF NULL
  12 REM ************************
  13 REM*                       *
  14 REM* TEST-PROGRAMM*        *
  15 REM*                       *
  16 REM ************************
  20 PRINT CHR$(147);:REM ALLES LOESCHEN
  30 FOR Z=1 TO 374:REM 374 ZEICHEN
  40 PRINT"A";
  50 NEXT
 999 REM ZEIT AUSDRUCKEN
1000 POKE 214,18:PRINT TI/60 "SEKUNDEN":END
```

Sieht gut aus, nicht wahr?

Aber leider, REM-Erläuterungen kosten Zeit. Wir sind um 0,15 (0,2) Sekunden langsamer geworden.

Wir schmeißen deshalb alle REMs wieder raus und haben damit wieder Version 12. Jetzt aber gehen wir einen Schritt in der anderen Richtung weiter und entfernen alle Leerstellen und Abstände. Mit dieser **Version 14** will ich Ihnen zeigen, daß das auch einen Einfluß auf die Laufgeschwindigkeit hat.

```
 10 TI$="000000"
 20 PRINTCHR$(147);
 30 FORZ=1TO374
 40 PRINT"A";
 50 NEXT
1000 POKE214,18:PRINT:PRINT-
     TI/60"SEKUNDEN":END
```

Das Ergebnis ist für den VC 20 0,81 Sekunden, für den C 64 0,98 Sekunden.

In **Version 15** treiben wir die Schrumpfung ins Extrem, indem wir das Programm im Prinzip unverändert aber mit einem Minimum an Zeilen schreiben, also möglichst viele Befehle in eine Zeile packen.

Sie wissen, die maximale Zeilenlänge beträgt 88 Zeichen beim VC 20, 80 Zeichen beim C 64. Unser Programm können wir sogar in einer einzigen Zeile unterbringen — fast unglaublich, aber es geht. Sie müssen allerdings alle Abkürzungsmöglichkeiten ausschöpfen, die das Commodore-System bietet. Im Anhang der Commodore-Handbücher finden Sie die Liste aller Abkürzungen beim Eintippen: C und geSHIFTEtes H für CHR$, ? für PRINT und so weiter. Im nachfolgenden Ausdruck ist das natürlich nicht zu sehen, weil der Befehl LIST die Abkürzungen nicht berücksichtigt. So kommen auch mehr als 88 (80) Zeichen in eine Zeile des Listings, woran Sie sich nicht stören dürfen.

```
10 TI$="000000":PRINTCHR$(147);
:FORZ=1TO374:PRINT"A";:NEXT
:POKE214,18:PRINT:PRINT TI/60
"SEKUNDEN":END
```

Und siehe da, diese »Kurzform« des Programms ist auch die allerschnellste Version. Der VC 20 braucht 0,78 Sekunden, der C 64 0,93 Sekunden. Diese letzte Beschleunigung wird dadurch erreicht, daß das Betriebssystem des Computers nur einmal einen Zeilenanfang und Zeilenende suchen und erkennen muß, statt sechsmal in der Version 14.

Das Ausnützen der vollen Kapazität einer Zeile bringt also nicht nur den Vorteil eines kleineren Speicherbedarfs, sondern auch Zeitgewinn.

Regel 6
* Programme ohne REM-Erläuterungen und ohne Leerstellen zwischen den Zeichen laufen schneller.
* Zur Reduzierung der Zeilenzahl sollen möglichst viele Befehle in eine Zeile geschrieben werden.

»Einzeiler« können auch Spaß und Herausforderung zugleich sein. Man sollte eigentlich annehmen, daß mit einer Zeile nicht viel anzufangen sei.
Weit gefehlt!

Einzeilige Programme

Eine englische Zeitschrift hat darüber sogar einige Male einen Wettbewerb ausgeschrieben (die 64'er ist unabhängig davon auch auf diese Idee gekommen, siehe Ausgabe 8; d. Red.). Ich schreibe die Einzeiler unten lesbar, das heißt mit Leerstellen. Sie müssen die Programme aber wieder mit allen Abkürzungstricks schreiben, sonst geht's schief.

□Von A. Boyd (Manchester) stammt ein Primzahlen-Erzeuger, der für die obere Grenze von 65000 viele Stunden braucht.

```
1 FOR N=1 TO 65000:F=0FOR J=2
TO N—1:F=F+((N—J*INT(N/J))=0)
:NEXT:X=—(F=0):PRINT
RIGHT$(STR$(X*N),6*X):NEXT
```

□Ein anderer Einzeiler wurde von A.M. Simmelt (Sheffield) geschrieben zur Konvertierung von Dezimalzahlen in Dualzahlen.

```
1 INPUT A:FOR I=14 TO 0 STEP—1
:Z=A AND 2↑I:A=A—Z:Z=Z/2↑
I:Z$=RIGHT$(STR$(Z),1):PRINT Z$;:
NEXT:GOTO1
```

□Zuletzt noch ein Juwel, nur für den VC 20 geeignet, von M. Dooling (Dewsbury). Hochauflösende Grafik und Scrolling in einer Zeile — wer hält's für möglich?

```
1 POKE 36869,255: PRINT"[SHIFT
CLR/HOME]@@@@@@@@@@
@@":FOR X=7168 TO 7175:POKE
X, PEEK(X+1):NEXT:POKE7176,
PEEK(A+8↑5):A=A+1:GOTO 1
```

Lassen wir's gut sein mit diesem Programmsport und kehren wir zurück zu einer abschließenden Betrachtung der Zeitgewinne.

Wir haben in Version 1 mit 8,25 (10,48) Sekunden begonnen. Diese Laufzeit wurde ohne Änderung des Programmresultats stetig verkürzt, bis wir schließlich in Version 15 bei 0,78 (0,93) Sekunden gelandet sind.

Ich nenne diese Beschleunigung um 90 Prozent schlicht und einfach spektakulär.

Mehr allerdings kann ich nicht herausholen, es sei denn — na ja, eigentlich habe ich am Anfang ganz laut »Basic« gesagt.

Aber ich kann doch nicht widerstehen und Sie scharf machen auf ultima velocitas — zu deutsch Maschinensprache. Dazu aber in der nächsten Ausgabe.

(Dr. Helmuth Hauck/aa)