BERND SEBASTIAN KAMPS

BASIC FÜR MOLLIS & MÜSLIS

PROGRAMMIERUNG VON LERNPROGRAMMEN
FÜR DEN FREMDSPRACHENUNTERRICHT FÜR
ANFÄNGER UND FORTGESCHRITTENE
EIN LEHRBUCH FÜR COMMODORE 64

STEINHÄUSER

BERND SEBASTIAN KAMPS

BASIC FÜR MOLLIS & MÜSLIS

PROGRAMMIERUNG VON LERNPROGRAMMEN
FÜR DEN FREMDSPRACHENUNTERRICHT FÜR
ANFÄNGER UND FORTGESCHRITTENE
EIN LEHRBUCH FÜR COMMODORE 64

Lektorat: Martin Obladen

STEINHÄUSER

Am Kriegermal 34
5600 Wuppertal 23

Druck: Druckstore, Bonner Wall 47, 5000 Köln

Computerisierte Zukunftswelt, neue Denkstrukturen, Generationenkonflikte durch Wissensunterschiede: die Verbreitung von Klein- und Personalcomputern bringt Veränderungen auch außerhalb von Herstellungs- und Informationsbetrieben. Für die Jugend ist die Auseinandersetzung mit der neuen Technologie Voraussetzung für das Verständnis der eigenen Zukunft, für die Älteren, im Alter über dreißig stehend, wird das Begreifen des Neuen bald der Brückenschlag zu den Nachwachsenden.
Der Computer drängt mit seinen Veränderungen in Gebiete vor, die scheinbar nichts zu befürchten hatten. Sein Einfluß im Schulbereich beschränkt sich nur scheinbar auf den neu eingeführten Informatikunterricht. Wissensvermittlung kann in einigen Fächern zum Teil von Computern übernommen werden. Bei den Schulfächern wird mit den größten Veränderungen in den Fremdsprachen gerechnet. Die Art des zu bewältigenden Wissens - Tausende von Einzelbegriffen (Vokabeln und konjugierte Verbformen), die von Lehrplänen in die Schülerköpfe gelangen sollen - scheint auf die Funktionsweise der Computer wie zugeschnitten. Trainingsprogramme für Wortschatz und Grammatik verkürzen die Lernzeiten auf weniger als die Hälfte. Eine Revolution im Fremdsprachenunterricht zeichnet sich ab.

Die Konzeption des vorliegenden Buches sollte zwei Fliegen mit einer Klappe schlagen: Einmal wird der Leser systematisch in die Programmiersprache BASIC eingeführt. Er gewinnt Einblick in die innere Organisation seines Rechners. Gleichzeitig lernt er, funktionsfähige und sehr leistungsfähige Trainingsprogramme für den Fremdsprachenunterricht zu schreiben. Zu beachten ist auch der logische Aufbau der Stoffdarbietung: Nicht eine Fülle von kleinen und untereinander nicht zusammenhängenden Nonsensprogrammen wird dem Leser und Jungprogrammierer zur Abschrift und Probe vorgelegt, stattdessen wird von Anfang bis Ende des Buches

an dem "Großprojekt Fremdsprachen" gearbeitet. Von Kapitel zu Kapitel werden die Programme professioneller und leistungsstärker, mit dem wachsenden Wissen wird zwiebelschalenartig ein immer komplizierteres Programmgebäude errichtet.
Der erste Teil des Buches stellt die Grundbegriffe des Programmierens dar. Die Funktionsweise eines Vokabeltrainers, mit dem die Schüler bis zu einhundert neue Wörter einer Fremdsprache innerhalb einer Stunde lernen können, wird verständlich. Der zweite Teil erläutert, wie der englische Zeichensatz des COMMODORE 64 verändert wird. Nach dieser Operation erscheinen auf dem Monitor deutsche, französische, italienische, spanische und portugiesische Sonderbuchstaben (z.B. die deutschen ä,Ä,ö,Ö,ü,Ü,ß). Es werden die Voraussetzungen geschaffen für die Programmierung aller denkbaren Alphabete bis hin zum Russischen und Arabischen. Im Anhang schließlich sind die Versuchsergebnisse einer Langzeitstudie zum computerassistierten Fremdsprachenunterricht sowie ein Interview mit Lehrer und Schüler einer solchen Versuchsreihe abgedruckt.

Das vorliegende Lehrbuch ist all denen bestimmt, die schnell und konsequent BASIC lernen wollen. Darüberhinaus richtet sich an zwei Gruppen: an Schüler jedes Alters, die schon früher der Faszination von Fremdsprachen erlagen und auf diesem Gebiet mit Begeisterung lernen und arbeiten. Die Aussicht, den gleichen Stoff in einer kürzeren Zeit oder aber in der gleichen Zeit einen umfangreicheren Stoff bearbeiten zu können, muß sie verwirren und betören. Nicht weniger betroffen als die Schüler sind die Lehrer. Es ist wichtig, daß sie die möglichen Konsequenzen einer neuen technologischen Entwicklung für ihr Fach früh genug bedenken und mit eigener kritischer Erfahrung den Veränderungen im Fremdsprachenunterricht begegnen. Erst dadurch werden sie schließlich dem Computer den ihm angemessenen Platz innerhalb des ihnen zur Verfügung stehenden Hilfsinstrumentariums zuweisen können.

Cagliari, im Juli 1985

Teil A

”Teil A“

Prolog

Rechner, Diskettenlaufwerk und Monitor werden entsprechend den Anweisungen im Commodore-Handbuch (siehe die Abschnitte über Anschluß und Inbetriebnahme) miteinander verkabelt und an das Stromnetz angeschlossen. Die Geräte werden in der Reihenfolge Bildschirm - Diskettenstation - Computer angeschaltet. Du führst eine Diskette (Demodiskette oder Spieldiskette von Freunden) vorsichtig bis zum Anschlag ins Laufwerk und verriegelst den Ausgang. Als erstes schreibst Du:

on

load "$",8

load "$",8

Wie nach jeder anderen erfragten oder ungefragten Eingabe an den Rechner drückst Du auf die **RETURN-Taste.** Der Rechner lädt sich das Inhaltsverzeichnis der Diskette in seinen Speicher. Sobald er zu verstehen gibt, daß er fertig ist -auf dem Bildschirm erscheint die Bemerkung "ready"-, blendest Du das Inhaltverzeichnis mit der Anweisung

list

list

auf dem Monitor ein. Wenn das Inhaltsverzeichnis für die zur Verfügung stehenden 24 Bildschirmzeilen zu lang ist, kannst Du den List-Befehl wiederholen und die Auflistung durch Niederhalten der **CTRL-Taste** verlangsamen. Mit der Taste RUN/STOP kann die Auflistung gestoppt werden. Du entscheidest Dich für eines der aufgeführten Programme und rufst es mit folgendem Befehl von der Diskettenstation in den Rechnerspeicher:

load "(Programmname)",8

Ein Programm mit dem Namen **"mollis"** müßte also in der Form

load "mollis",8

eingeladen werden. Wenn der Rechner das Ende des Ladevorgangs mit einem erneuten "ready" anzeigt, starten wir das Programm mit

run **run**

(+ RETURN, wie jede gefragte und ungefragte Eingabe an den Rechner, Du erinnerst Dich...). Unzählige Spielprogramme werden auf diese Weise geladen und gestartet. Mit Joy-Sticks wird tage-, nächte- und wochenlang gefahren, geschossen und geflogen.

****

Nach einiger Zeit (je älter der Spieler, um so kürzer diese Zeit) verfliegt die Lust an neurotischen Joy-Stick-Fingerübungen. Es kommt der Gedanke, man könne selber Programme schreiben. Du unterbrichst das laufende Spielprogramm, indem Du auf die Taste

RUN/STOP

drückst und gleichzeitig etwas stärker auf die

RESTORE-Taste

schlägst. Anschließend schreibst du den Befehl

new **new**

Durch "new" werden Programme, die augenblicklich eingeladen sind, aus dem Rechnerspeicher hinausgeblasen. Nicht alle Spielprogramme lassen sich jedoch mit RUN/STOP-RESTORE unterbrechen. In diesen Fällen bleibt nur die Möglichkeit, den Rechner durch eine kurze Unterbrechung der
off Stromzufuhr in den Urzustand zu versetzen.

1. Kapitel

Wir lernen in diesem Kapitel den print-Befehl kennen, führen damit Rechenoperationen aus und begegnen einer seltsamen Formel (? chr$(147)), mit der wir den Bildschirm säubern. Wir lernen, Sätze auf den Bildschirm zu plazieren.

Wer zum erstenmal vor dem Rechner sitzt, wird sich auf Kommunikationsschwierigkeiten einstellen. Rechner und Programmierer sprechen verschiedene Sprachen und können sich nicht problemlos verständigen. Auf die an sich korrekte Frage:

"Wieviel ist 2 mal 2 ?"

antwortet der Rechner mit der Bemerkung:

"Syntax Error"

was frei übersetzt bedeutet, daß er die Anweisung nicht verstanden hat. Der Rechner wird immer erwarten, daß die an ihn gerichteten Anliegen **in seiner Sprache** niedergeschrieben werden. Diese Sprache heißt **BASIC** und ist eine Fremdsprache, deren wichtigste Wörter nach herrschenden Regeln gelernt werden müssen. Zur Erleichterung des Einstiegs führen wir nachfolgend eine Deutsch-Basic-Vokabelliste mit den geläufigsten Gesprächsinhalten auf.

DEUTSCH	BASIC
Geh sofort rüber nach Zeile x!	goto x
Schreib auf den Monitor	print (oder: ?)
Schreib auf Diskette!	print#1
Stell 'ne Frage!	input
Lösch alles,was im Speicher steht!	new
Lies Daten in Strings ein!	read
Mach hier Daten zum Lesen fertig!	data
Geh ins Unterprogramm!	gosub
Raus aus dem Unterprogramm!	return
Hier hast Du nichts zu suchen! - Hau ab in die nächste Zeile!	rem
Lade mir ein Programm in den Rechner!	load "",8
Speichere ein Programm auf Diskette!	save "",8
Mach den Bildschirm sauber!	chr$(147)
Sag mir eine Zahl zwischen 1 und x!	rnd(1)*x+1
Sag mir, wie spät es ist!	? ti$
Halt den Mund und warte auf weitere Anweisungen!	stop
So, kannst jetzt weitermachen (nur nach vorherigem "stop")!	cont
Mach einen Diskettenkasten auf!	open
Mach den Diskettenkasten wieder zu!	close
Sag mir, aus wieviel Buchstaben ein Wort besteht!	len("..")
Mache mir eine Feldvariable!	dim
Stecke das, was rechts vom Gleichheitszeichen steht, rüber nach links!	i=i+1
Programm ist zu Ende. Kannst die Klappe halten!	end

Es ist sinnvoll, sich diese Vokabelliste 5 bis 10 Minuten intensiv anzuschauen. Die Einarbeitung in unsere neue Materie wird erleichtert, wenn die wichtigsten Begriffe schon einmal gehört wurden.
Wir gehen zu unserer Rechenaufgabe vom Anfang

zurück und können sie nun korrekt übersetzen.
Die deutsche Frage: "Wieviel ist 2 mal 2 ?"
heißt in Basic:

print print 2*2

Das vor der Rechenanweisung stehende "print"
kannst Du durch das schlichtere "?" ersetzen:

? ? 2*2

Merke: jedesmal, wenn Du etwas auf dem
Bildschirm erscheinen lassen willst, gebrauchst
Du vorerst den print-Befehl!
Sämtliche Rechenarten werden auf diese Weise
bewältigt:

+	? 17+4	- Addieren
-	? 12-5	- Substraieren
*	? 2*2	- Multiplizieren
/	? 100/20	- Dividieren

Beachte, daß Dezimalstellen wie in

2.18 ? 2.18*2

durch **einen Punkt abgetrennt werden!**

Wenn Du mit dem Print-Befehl nicht
Rechenergebnisse, sondern Wörter oder Sätze auf
den Schirm projizieren willst, müssen diese
zwischen Anführungsstrichen geschrieben werden.
Auf ein schlichtes

? Fritz hat geerntet

reagiert der Rechner mit der uns bekannten
verständnislosen Antwort. Erst wenn wir die

nicht näher bezeichnete Ernte von Fritz in korrektem Basic schreiben:

```
? "..."
```

```
? "Fritz hat geerntet"
```

wird der Satz unbeanstandet ausgegeben.

Ebenso:

```
? "Politiker sind dumm"
```

Du kannst auch mehrere Befehle hintereinander aufschreiben, mußt sie dann aber durch einen Doppelpunkt trennen:

```
? chr$(147)
```

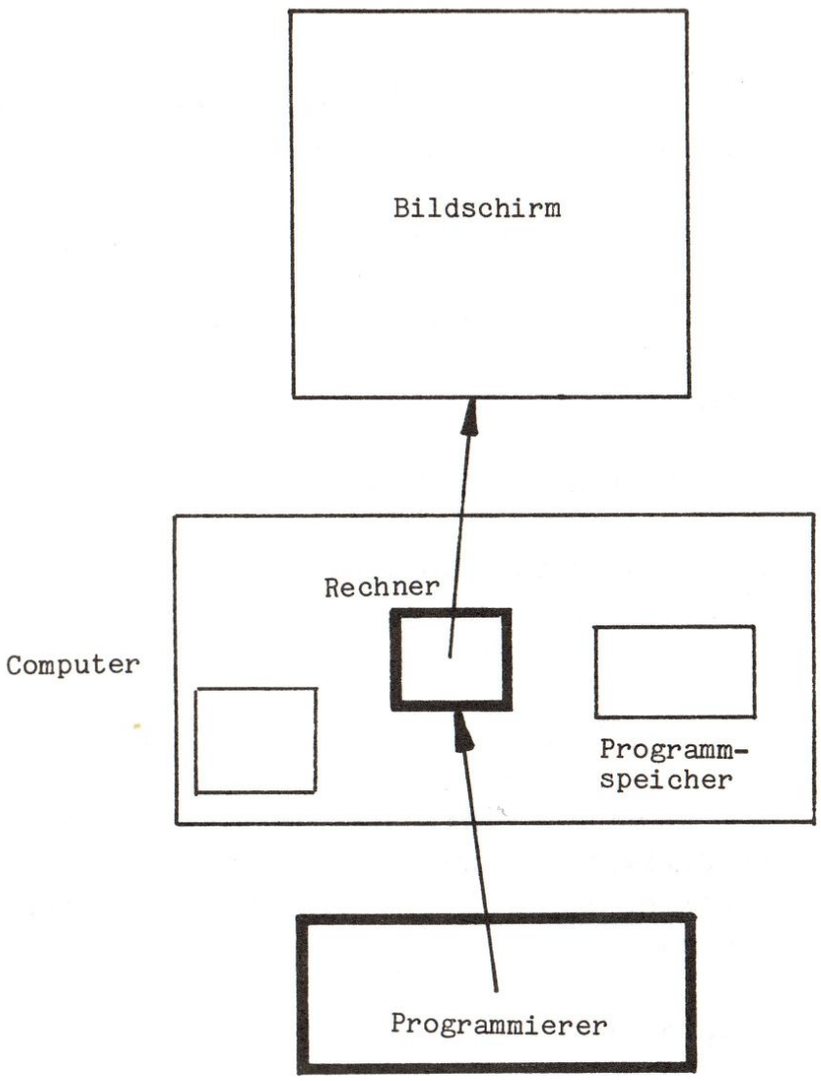
```
? chr$(147):? "Politiker sind dumm"
```

Der Rechner führt die Programmanweisung **"? chr$(147)"** aus und löscht damit den Bildschirm, bevor er zur Print-Anweisung übergeht. Besser gar noch:

```
? chr$(147):?:?:?:?:?:? "          Politiker sind dumm!!"
```

wodurch unser letzter Satz formschöner in die Bildschirmmitte rückt. Beachte: "?" (oder "print") ohne jeglichen Zusatz schreibt eine Leerzeile. Fünfmaliges, durch Doppelpunkte getrenntes nacktes "?" ergibt 5 Leerzeilen und unser Satz erscheint in der sechsten Bildschirmzeile.

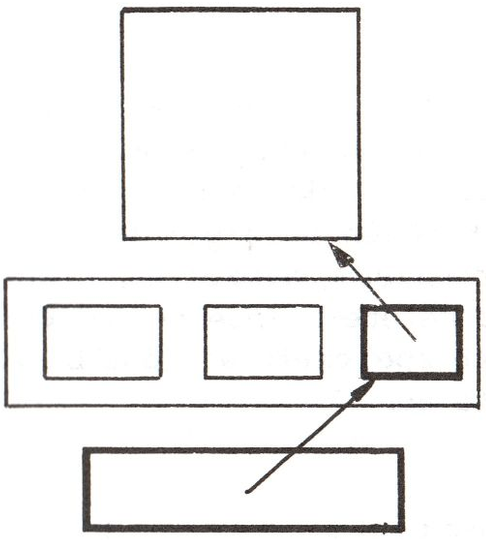
Was wir gerade gemacht haben, nennt man **Direkt-Anweisungen.** Wir schreiben eine Anweisung, geben RETURN und der Rechner führt sie aus:



Zeichnung 1.

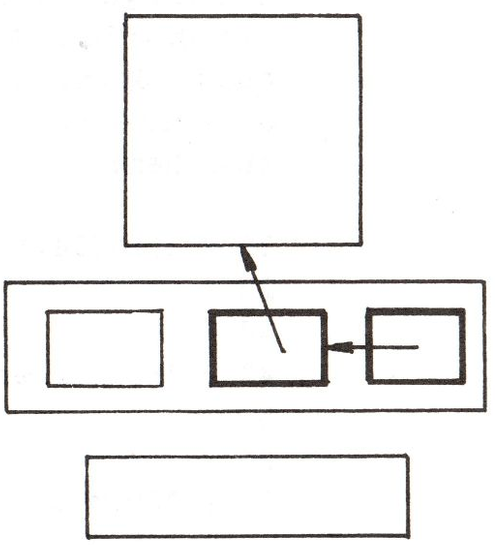
Direktanweisungen: Der Programmierer richtet Befehle an den Rechner und schließt sie mit RETURN ab. Die Ergebnisse erscheinen sofort auf dem Bildschirm.

Viel eleganter ist es aber, die Anweisungen von einem Programm aus zu geben. Wenn wir dieses mit "run" starten, "sprechen" nur noch Programm und Rechner miteinander und führen gemeinsam die Anweisungen aus, die wir zuvor in dem Programm niedergelegt haben:



Zeichnung 2.

Programmierphase: Der Programmierer schreibt seine Befehle zunächst in ein Programm. Auf dem Bildschirm erscheinen zwar die Programmzeilen (siehe auch Kapitel 2), die Befehle werden aber noch nicht ausgeführt.



Zeichnung 3.

Arbeitsphase: Der Programmierer hat sein Programm gestartet, und Programmspeicher und Rechner im engeren Sinne "reden" miteinander und führen das Programm aus. Eventuelle Ergebnisse erscheinen auf dem Bildschirm oder dem Drucker.

2. Kapitel

Wir lernen in diesem Kapitel, wie Programme aufgebaut werden und begegnen den Programmzahlen. Mit der "goto"-Anweisung springen wir von einem Punkt des Programms zu einem beliebigen anderen. Wir fügen fertigen Programmen weitere Befehle hinzu und löschen Programmzeilen. Fertige Programme lernen wir danach mit Delete-, Insert- und Cursortasten zu verändern. Schließlich speichern wir das Ergebnis unserer Arbeit auf der Diskette ab.

Ein Programm besteht aus vielen Einzelanweisungen. Dem Rechner muß deshalb eine Richtlinie an die Hand gegeben werden, aus der ihm ersichtlich wird, in welcher Reihenfolge die Anweisungen auszuführen sind. Du legst diese Reihenfolge fest, indem Du vor Deine Anweisungen eine **Zahl** schreibst und jede geschriebene Programmzeile mit einem RETURN abschließt. Je niedriger die Zahl, um so früher wird die zugehörige Anweisung später vom Rechner ausgeführt. 10 wird vor 20, 20 vor 30 usw. abgearbeitet. Ein kurzes Programm könnte so aussehen:

```
10 ? chr$(147)
20 ?
30 ?
40 ? "Politiker sind dumm"
50 ?
60 ?
70 ? "Dumm und faul!!"
```

Sobald Du das Programm mit "run" startest, geht der Rechner nach 10, wo er vom Programm die Anweisung bekommt, den Bildschirm zu löschen. Dann, in 20 und 30, wird die Erzeugung von zwei Leerzeilen veranlaßt, bevor der Rechner aus 40

den ersten Satz in die dritte Bildschirmzeile bringt. Nach weiteren zwei Leerzeilen aus 50 und 60 erscheint durch 70 der zweite Satz in der sechsten Bildschirmzeile.

In diesen strengen Ablauf der Programmbearbeitung kannst Du mit der **goto**-Anweisung verändernd eingreifen. Mit goto ist es möglich, den Rechner von einer Programmzeile aus in eine beliebige andere zu schicken. Schreibe als zusätzliche Zeile

```
80 goto 20
```

Nach dem erneuten Start des Programms werden unsere beiden Beispielsätze unablässig auf den Bildschirm geholt. Sobald das Programm bis 70 abgearbeitet ist, bekommt der Rechner in 80 die Anweisung, zurück nach 20 zu gehen. Von dort aus werden von neuem aufsteigend alle Programmzeilen bearbeitet, der Rechner gelangt zum zweitenmal nach 80, springt wieder zurück usw. Wenn Du die Programmbearbeitung nicht durch Betätigen der stop-Taste unterbrichst, schreibt der Rechner unverdrossen die immer gleiche Satzfolge... bis der Strom ausfällt, der Rechner seinen Geist aufgibt, das Haus abgerissen oder die Stadt durch einen Atomangriff zerstört wird.

Genauso wichtig wie ein Programm zu schreiben ist es, es nachher zu verändern, zu korrigieren und zu vervollständigen. In der Computersprache werden diese Änderungen als "Editieren" bezeichnet. Es gibt verschiedene Möglichkeiten des Editierens:

1. Einfügen zusätzlicher Programmzeilen:
Du fügst eine bisher noch nicht benutzte Zeilennummer mit einer zusätzlichen Anweisung ein, etwa

```
25 goto 60
```

Würden wir das Programm auflisten, sähen wir Zeile 25 eingefügt zwischen die Zeilen 20 und 30. Du siehst an diesem Beispiel auch, daß Programmabschnitte mit der **goto**-Anweisung übersprungen werden. Ein Programmstart nach Einfügen von Zeile 25, und Du siehst, daß auf dem Bildschirm nur noch von Faulheit und Dummheit die Rede ist.

2. Löschen von Programmzeilen:
Programmzeilen löschst Du, indem Du die zugehörige Zeilennummer tippst und kommentarlos mit RETURN quittierst. Du schreibst:

```
25
```

und gibst RETURN. In der Auflistung des Programms ist Zeile 25 nicht mehr vorhanden.

3. Änderungen innerhalb einer Programmzeile:
Eine Programmzeile neu zu schreiben, nur weil dort ein Rechtschreibefehler gemacht wurde, ist unwirtschaftlich. Um kleine "Macken" auszubessern, bedienst Du Dich der **Cursor-Tasten** unterhalb der RETURN-Taste. Du kannst zweierlei Korrekturen ausführen:

a. Einfügen von Buchstaben:
Du gehst mit dem Cursor auf den Buchstaben, der nach rechts verschoben werden soll, drückst auf SHIFT und danach sooft auf INST/DEL (oben rechts auf der Tastatur), bis genügend Platz geschaffen ist.

save "bulle",8; verify "bulle",8

b. Löschen von Buchstaben:
Du gehst mit dem Cursor auf den Buchstaben **hinter** demjenigen, der gelöscht werden soll und drückst auf INST/DEL.

Programmieren ist für den Anfänger, der mit der Tastatur noch nicht vertraut ist, eine langwierige und zeitraubende Angelegenheit. Um so schlimmer ist es daher, wenn sich das Ergebnis dieser Arbeit durch Dummheit (unvorsichtige Unterbrechung der Stromzufuhr) oder höhere Gewalt (spielende Kinder in der Nähe der Steckdose; frei laufende Nagetiere, die die Stromleitungen anknabbern; Blitz- oder Anschlag auf das städtische Elektrizitätswerk) ins Nichts verflüchtigt. Schon jetzt speicherst Du Dein kleines Programm auf der Diskette und schreibst:

save "@0:(Programmname)",8

Wenn Dein Programm "bulle" heißen soll, hieße die ausgeschriebene Anweisung:

save "@0:bulle",8

Um sicher zu gehen, daß der Inhalt des Rechnerspeichers heil auf der Diskette angekommen ist, schreibst Du außerdem:

verify "bulle",8

"verify" bittet den Rechner nachzuprüfen, ob das gerade unter dem Namen "bulle" auf der Diskette gespeicherte Programm völlig identisch ist mit dem Programm, das im Rechnerspeicher steht.
Eleganter als "save" und "verify" in der Direktanweisung zu schreiben ist es, sie ins

Programm einzubauen:

```
59999 end
60000 save "@0:bulle",8
60001 verify "bulle",8
```

Ein schlichtes, im Direktmodus geschriebenes

goto 60000

setzt die Sicherungs- und Prüfroutine in Gang und garantiert außerdem, daß das Programm nicht versehentlich unter einem falschen Namen gespeichert wird.

Sobald das Programm gesichert ist, simulierst Du einen Stromausfall. Nach dem Wiedereinschalten der Geräte schreibst Du

load "bulle",8

und startest mit "run": Der Rechner läuft wieder auf seiner unendlichen 20/80-Runde...

Du unterbrichst den Rechner in seiner Arbeit und fegst das Programm mit dem schon bekannten Befehl

new

aus dem Speicher. Die Zeit des Spiels ist vorbei. In den nächsten Kapiteln wird ernsthaft gearbeitet.

Du hast in diesem Kapitel gelernt:

- daß mit Programmzahlen dem Rechner gesagt wird, in welcher Reihenfolge die Anweisungen auszuführen sind.

- daß mit "goto + (Zeilennummer)" der Rechner innerhalb des Programmes beliebig springen kann.
- daß Programmzeilen an beliebiger Stelle eingefügt werden können, wenn die Zeilennummer noch nicht besetzt ist.
- daß Programmzeilen gelöscht werden, indem die Zeilennummer mit einem kommentarlosen RETURN quittiert wird.
- daß zur Feinkorrektur die Tasten für die Cursorsteuerung zusammen mit der INST/DEL-Taste (mit oder ohne SHIFT) benötigt werden.
- daß Programme mit dem Befehl save "(Programmname)",8 auf der Diskette gespeichert werden und mit verify der Speichervorgang noch einmal kontrolliert wird.

Zehnfingersystem

3. Kapitel

Nachfolgend eine Erklärung des Zehn-Finger-Systems! Wer ein guter Programmierer werden will, ist gut beraten, dieses Schema zu erlernen. Nichts ist hinderlicher als Programmanweisungen mühsam mit zwei Fingern zusammenzusuchen und nichts anstrengender und ermüdender für die Augen, mit dem Blick ständig zwischen Bildschirm und Tastatur zu wandern. Mit weniger als zehn Fingern zu schreiben sollte daher auf Dauer tunlichst vermieden werden.

Linke Hand

kleiner Finger:	q,a,y
Ringfinger:	w,s,x
Mittelfinger:	e,d,c
Zeigefinger:	r,f,v,t,g,b
Daumen:	Leertaste

Rechte Hand

Daumen:	Leertaste
Zeigefinger:	z,h,n,u,j,m
Mittelfinger:	i,k
Ringfinger:	o,l.,
kleiner Finger:	p,ö,.,ß,ü,ä,-

? chr$(14); ? chr$(142)

4. Kapitel

Wir beginnen in diesem Kapitel mit den Vorbereitungen zur Programmierung von Vokabeltrainingsprogrammen und lehren unseren Rechner die ersten Vokabeln: er soll mit der "read"-Funktion "data-Zeilen" auslesen. Wir selber lernen, wie wir diese Vokabeln in sogenannte "Stringvariablen" zwischenspeichern. Schließlich begegnen wir in der "tab"-Funktion einem nützlichen Hilfsmittel, um unsere Bildschirmausgaben formschöner zu gestalten.

Wenn Du gleichzeitig auf die Commodoretaste am unteren linken Tastaturrand und auf die SHIFT-Taste drückst, siehst Du, daß Du zwischen zwei verschiedenen **Zeichensätzen** wählen kannst. Beim "Graphikmodus", der eingestellt ist, wenn der Rechner eingeschaltet wird, erscheinen Großbuchstaben auf dem Bildschirm. Beim "Schreibmodus", nach Umschalten mit CO+SHIFT, erscheinen Kleinbuchstaben, während für Großbuchstaben ähnlich wie auf einer Schreibmaschine gleichzeitig mit der Buchstabentaste die SHIFT-Taste gedrückt werden muß. Um von einem Programm aus die Umstellung von Graphikmodus nach Schriftmodus zu erreichen, mußt Du schreiben:

```
? chr$(14)                    ? chr$(14)
```

Von Schriftmodus nach Graphikmodus wird mit:

```
? chr$(142)                   ? chr$(142)
```

umgestellt. Du beginnst Dein nächstes Programm mit den Zeilen:

data-Zeilen

```
1 ? chr$(14)
59999 end
60000 save "@0:generator",8
60001 verify "generator",8
```

Wenn der Rechner Dein Vokabelwissen prüfen soll, muß er zunächst sich selber dieses Wissen aneignen. Vokabeln sind **Daten**, die Du ihm in geeigneter Form zur Verfügung stellen mußt. In Basicprogrammen werden Daten hinter eine **data**-Anweisung geschrieben. Steht mehr als eine Vokabel in einer Zeile, müssen sie durch ein Komma getrennt sein. Beachte, daß eine data-Zeile nicht länger sein darf als zwei Bildschirmzeilen!!
Du schreibst folgenden Datenspeicherabschnitt:

```
10010 data to kiss, kuessen
10020 data to besiege,belagern
10030 data dissident,andersdenkend
10040 data demonstrator,"Demonstrant"
10050 data to escape,entkommen
10060 data peace mouvement,"Friedensbewegung"
10070 data popular uprising,"Volksaufstand"
10080 data stone-throwing,steinewerfend
10090 data to struggle,kaempfen
10100 data to undermine,unterwandern
10110 data to survive,ueberleben
10120 data crowd,"Menschenmenge"
10130 data solidarity,"Solidaritaet"
10140 data injustice,"Ungerechtigkeit"
10150 data dangerous,gefaehrlich
```

Merke: Wörter mit großen Anfangsbuchstaben (Demonstrant, Friedensbewegung etc) müssen in data-Zeilen zwischen Anführungszeichen geschrieben stehen!

Mit der Niederschrift dieser Daten in data-Zeilen ist allein nichts gewonnen. Zwar können sie mit "list" wieder auf den Bildschirm geholt werden, sie sind aber noch nicht genügend aufbereitet, als daß der Rechner sie einzeln oder nacheinander auf den Bildschirm projizieren könnte. Zuvor muß der Rechner selbst diese Vokabeln **lesen.** Er macht dies mit **read,** wenn wir ihm vorher noch sagen, unter welchem Codenamen er sich den so gewonnenen Lesestoff merkt. Wenn Du schreibst:

```
10 read a$
20 print a$
```

und "run" gibst, liest der Rechner das erste Wort aus unserer Sammlung von data-Zeilen, also "to besiege", gibt diesem Wort den Codenamen a$ und blendet dieses a$ in Zeile 20 auf den Bildschirm ein.

Merke Dir, daß die Codenamen, unter denen der Rechner sich Wörter merkt:
- aus einem oder zwei Buchstaben bestehen:
  (Beispiele: a$,b$,ab$,b3$,gh$,fx$,t4$)
- am Ende immer ein Dollarzeichen ($) haben
- im Computerjargon als **Stringvariable** bezeichnet werden

Zurück zu den Vokabeldaten. Du fügst als dritte Programmzeile

```
30 goto 10
```

ein, startest das Programm erneut und siehst, daß der Rechner nacheinander alle Wörter der data-Zeilen auf den Bildschirm, sich aber nach dem letzten Wort mit einer Fehlermeldung verabschiedet:

tab("Spalte")

out of data error in 10

und damit sagen will, daß der read-Funktion aus Zeile 10 die Lektüre ausgegangen sei. Wenn Du jetzt im Direktmodus nach dem Inhalt von a$ fragst:

```
? a$
```

so siehst Du, daß in der Stringvariablen a$ "gefährlich" gespeichert ist. Zwar wurde der Inhalt der data-Zeilen auf den Schirm gebracht, doch der Rechner hat sie immer noch nicht gelernt. Das ändert sich auch nicht, wenn Du die Wörter formschöner auf den Bildschirm bringst, etwa durch

```
10 read a$,b$
20 print a$ tab(20) b$
30 goto 10
```

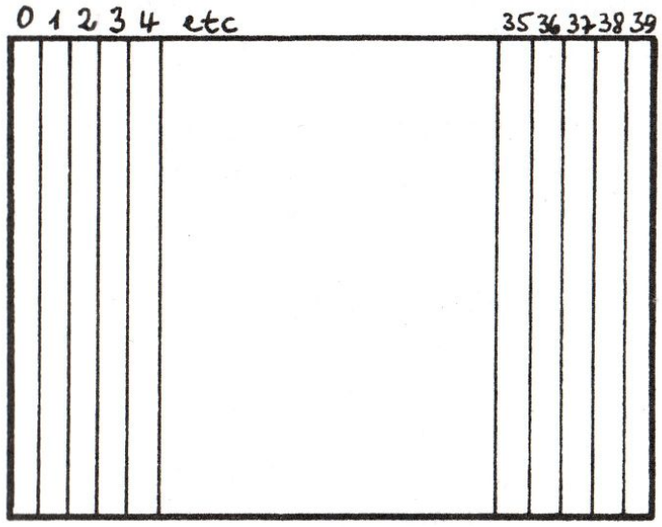
tab( )

Hier werden in Zeile 10 die ersten beiden Wörter aus unserer Vokabelliste gelesen und das erste als a$, das zweite als b$ zwischengespeichert. In 20 wird a$ an den linken Bildschirmrand geschrieben, während durch **tab(20)** die nachfolgende Stringvariable b$ mit dem ersten Inhalt "belagern" in die 20. Bildschirmspalte plaziert wird.
Auch jetzt meldet sich der Rechner nach Abarbeiten der data-Zeilen immer noch mit der Fehlermeldung "out of data error in 10". Im Direktmodus erfragtes a$ ergibt nun "dangerous", ? b$ ergibt "gefährlich". Der Rechner hat zwei Wörter gelernt, viel zu wenig.
Du beendest das Kapitel, indem Du mit

```
goto 60000
```

das neue Programm auf der Diskette sicherst.

Wir können noch nicht zufrieden sein mit den Ergebnissen unserer Arbeit und müssen auf das nächste Kapitel warten, um unseren Rechner zu einem folgsamen Schüler zu machen.



Zeichung 4:
Spaltenaufteilung des Bildschirms

Du hast in diesem Kapitel gelernt:

- daß Daten dem Rechner in data-Zeilen zubereitet werden müssen und durch Kommata getrennt werden, daß Wörter mit großen Anfangsbuchstaben zwischen Anführungsstrichen gehören und daß eine data-Zeile höchstens zwei Bildschirmzeilen lang sein darf.
- daß Daten aus data-Zeilen mit der read-Funktion im Rechnerspeicher unter einem Codenamen zwischengespeichert werden.
- daß der Codename für Wörter im Computerjargon "Stringvariable" genannt wird
- daß eine Stringvariable aus einem Buchstaben oder aus einer beliebigen Kombination von zwei Buchstaben besteht und am Ende ein Dollarzeichen trägt.
- daß der Bildschirm 40 Spalten hat und mit der tab-Funktion der Wortanfang in eine beliebige Spalte projiziert werden kann.

i=i+1

5. Kapitel

Wir lernen in diesem Kapitel, zu zählen, wie oft ein bestimmter Programmabschnitt vom Rechner durchlaufen wurde und warum i=i+1 ist. Wir begegnen dem Begriff der "Feldvariablen", "dimensionieren" uns 20 verschiedene a$'s und speichern dorthin unsere Vokabeln aus den data-Zeilen.

Bevor wir in unserem Programm fortfahren, müssen wir noch erklären, wie es möglich ist zu zählen, wie oft ein bestimmter Programmabschnitt durchlaufen wird. In unserem kleinen Programm

```
1  ? chr$(14)
10 read a$,b$
20 ? a$ tab(20) b$
30 goto 10
+ data-Zeilen bei 10010ff
```

springt der Rechner immer wieder von 30 nach 10 zurück und durchläuft die Zeilen 10 bis 30 solange, bis in 10 der Lesestoff ausgeht. Wir fügen

i=i+1

```
27 i=i+1
```

ein und wundern uns zunächst über den ungewöhnlichen mathematischen Ausdruck. Sicher haben wir einmal gelernt, daß entweder i=i oder i+1=i+1 ist, der Ausdruck aus Zeile 27 aber sicherlich falsch ist. Es handelt sich hier jedoch auch nicht um eine mathematische Gleichung. Schon das Gleichheitszeichen (=) heißt in Basic nicht "gleich", sondern heißt, daß das, was rechts vom Gleichheitszeichen steht, nach links gepackt werden soll.
Um diesen auf den ersten Blick unverständlichen Vorgang einsichtig zu machen, greifen wir auf das Eier-Beispiel aus der Grundschule zurück.

Erinnern wir uns, daß i für den Rechner eine Kiste ist, in der eine Anzahl Eier abgelegt sind. Bei Programmbeginn sind in der Kiste "i" keine Eier enthalten. Sobald der Rechner in Zeile 27 angelangt, öffnet er Kiste "i", nimmt den Inhalt heraus und fügt ihm eine Einheit hinzu (i=i+1), so daß wir insgesamt ein Ei haben. Er geht dann hinüber auf die andere Seite des Gleichheitszeichens und verstaut die Eier (oder das Ei) wieder in Kiste "i". Nachdem der Rechner von 30 zurück nach 10 geschickt wurde und zum zweiten Mal bei 27 vorbeikommt, findet er bereits ein Ei vor, packt ein weiteres hinzu und packt nun 2 Eier in die Kiste "i". Wenn wir das Programm starten, die Vokabeln auf den Schirm geblendet wurden und schließlich die Fehlermeldung kommt: out of data error in 10, so können wir durch ein schlichtes

```
? i
```

erfragen, wie oft der Rechner an Zeile 27 vorbeigekommen ist, das heißt, wie oft der Rechner mit Erfolg Daten aus den data-Zeilen in die a$()-Variablen eingelesen hat. Der Wert, der auf "? i" ausgegeben wird, entspricht der Anzahl der Vokabelpaare, die der Rechner eingelesen und ausgedruckt hat.
Wenn wir unser Programm noch einmal starten, können wir nach der Fehlermeldung i, a$ und b$ abfragen und erhalten die Anzahl der gelesenen Vokabelpaare sowie die letzten beiden Wörter, die vom Rechner gelesen wurden. Wenn der Rechner uns späterhin Vokabeln abfragen soll, muß er sie vorher selber lernen, wovon aber im Augenblick noch keine Rede sein kann. Er kennt nur "dangerous" und "gefährlich". Wir müssen ihm daher ein Mittel an die Hand geben, damit er sich mehr als nur ein Vokabelpaar merkt. Dies gelingt uns mit einer sogenannten **Feldvariablen,**

Dimensionierung einer Feldvariablen: dim a$(20)

ein Ding, das sich komplizierter anhört, als es ist. Sehen wir selbst...
Stellen wir uns vor, daß wir dem Rechner eine begrenzte Anzahl von verschiedenen a$'s und b$'s zur Verfügung stellen wollen, unter denen er alle in den data-Zeilen enthaltenen Wörter intern abspeichern kann. Diese verschiedenen a$'s und b$'s müßten zur Unterscheidung untereinander gekennzeichnet werden, etwa indem wir ihnen eine **Kennziffer** mitgeben, der besseren Kenntlichmachung wegen zwischen Klammern geschrieben. Wir bekommen dann ein a$(1), ein a$(2), a$(3), a$(4) usw. Die gleiche Numerierung gilt für b$'s. Bevor wir anfangen können, in diesen neuen Stringvariablen (denn a$(1) ist eine Stingvariable wie a$) unsere Wörter abzulegen, müssen wir dem Rechner sagen, wieviele a$'s er bereithalten soll, wir müssen **die Feldvariable a$ dimensionieren:**

dim

```
5 dim a$(20),b$(20)
```

"dim" steht für Dimensionieren und heißt, daß im Rechner die Stringvariablen a$(1), a$(2), a$(3)..... bis a$(20) (und ähnlich für b$()) darauf warten, beschriftet zu werten.
Nun kommt auch der Einsatz unseres Zählers aus dem letzten Abschnitt! Wir schreiben Zeile 20 um, definieren in 6, daß i=1 sei, und danach sieht unser Programm folgendermaßen aus:

```
1 ? chr$(14)
5 dim a$(20),b$(20)
6 i=1
10 read a$(i),b$(i)
20 ? a$(i) tab(20) b$(i)
27 i=i+1
30 goto 10
```

Nachdem wir die Feldvariablen a$() und b$() in 5 dimensioniert und in 6 für i den Wert 1 festgelegt haben, wird unser erstes Vokabelpaar in 10 in die Variablen a$(1) und b$(1) eingelesen (i ist ja 1!) und in 20 auf den Schirm projiziert. In 27 wird der Wert von i um 1 erhöht und hat danach den Wert 2, und aus 30 geht der Rechner zurück nach 10. Dort liest er das nächste Vokabelpaar in a$(2) und b$(2) ein (i ist nun 2!) usw. Der Rechner geht in dem Programm auf und ab, erhöht bei jedem Durchgang i um 1, speichert die gerade gelesenen Vokabeln in die nächsten a$() und b$() und steigt erst aus, wenn ihm der Lesestoff ausgeht und die Fehlermeldung für Zeile 10 kommt.
Wenn wir jetzt nachprüfen, was der Rechner sich gemerkt hat, so können wir mit unserem Schüler zufrieden sein. Auf "? a$(9)" wird "to struggle" ausgegeben, auf "? b$(7)" "Volksaufstand", auf "? b$(5)" "entkommen" usw. In 15 verschiedenen a$'s und b$'s wurden unsere Vokabeln aus den data-Zeilen abgelegt. Mit diesem Wissen unseres Computers könnten wir jetzt schon beginnen, Trainingssequenzen zum Vokabellernen auszuarbeiten, doch bevor wir zum ersten Ziel dieses Übungsbuches gelangen, sind noch einige kleine Unebenheiten im Programmablauf zu beseitigen.

Du hast in diesem Kapitel gelernt:

- daß i gleich i+1 sein kann und daß der Ausdruck i=i+1 bedeutet, daß alles, was rechts vom Gleichheitszeichen steht, hinüber nach links geschickt und dort zwischengespeichert wird.
- daß wir mit Feldvariablen mehrere a$'s und b$'s als Codenamen für unsere Vokabeln bereithalten können.
- daß die maximale Anzahl von Strings einer Feldvariablen am Programmanfang mit der dim-Funktion festgelegt wird.

if a$(i)="r" then 100

6. Kapitel

Wir lernen in diesem Kapitel eine sogenannte "Programmverzweigung" kennen und verhindern damit, daß dem Rechner der Lesestoff ausgeht. Schließlich führen wir die "for-next-Schleife" ein, eines der wichtigsten Instrumente im Basic, das ähnliche Funktionen hat wie der Zähler des letzten Kapitels.

Unser letztes Programm wird durch eine Fehlermeldung des Computers abgebrochen. Dies ist unter keinen Umständen befriedigend, denn nur wir, auf keinen Fall aber der Rechner, haben zu bestimmen, wann ein Programm endet. Oberstes Ziel aller Programmierkunst ist es daher auch immer zu verhindern, daß der Rechner mit einer Fehlermeldung seinen Dienst quittiert und auf die Kommandoebene zurückschaltet. Im vorliegenden Fall mußt Du daher vermeiden, daß dem Rechner der Lesestoff ausgeht und ihn rechtzeitig aus seiner 10/30-Schleife hinausschicken, wenn die data-Zeilen abgearbeitet sind. Du fügst in Dein Programm

```
10999 data r,r
```

sowie

```
29  if a$(i)="r" then 100
100 end
```

ein. Ein neuer Programmstart führt die Auflistung unserer Vokabeln anstandslos aus, ohne am Ende mit einer Fehlermeldung aufzuwarten. Was ist geschehen?

if (Bedingung erfüllt) then (Konsequenz)

if... then...

Wir sind zum ersten Mal einer der wichtigsten Strukturen des Basic begegnet: der **bedingten Programmverzweigung "if...then"**!! Ausführlicher geschrieben heißt "if...then...":

if (Bedingung erfüllt) then (Konsequenz)

In unserem Fall prüft der Rechner bei jeder 29er-Passage, ob a$(i) (die zuletzt eingelesene a$()-Vokabel) identisch ist mit r. Bei den ersten Passagen ist dies nicht der Fall, a$(1) ist "to besiege", a$(2) lautet "dissident", a$(3) ist "demonstrator" usw. Die Bedingung -a$(i)="r"- ist nicht erfüllt, und der Rechner geht unverrichteter Dinge weiter in die nächste Programmzeile. Erst wenn in a$(i) das Wort "r" und somit die letzte unserer data-Zeilen eingelesen wurde, ist die Bedingung aus Zeile 25 erfüllt, und erst in diesem Augenblick führt der Rechner die **Konsequenz** aus, die in der Formel "if...then..." hinter "then" steht: er geht nach 100, denn "then 100" ist eine Kurzformel für "then goto 100". In 100 wird dem Rechner mit **end** gesagt, er solle seine Aktivitäten einstellen, auf die Kommandoebene zurückkehren und mit "ready" bekanntgeben, daß er auf die nächsten Befehle warte.

Wir schreiben unser Programm zur besseren Übersicht noch einmal in geordneteren 10er-Schritten:

```
1 ? chr$(14)
2 ? chr$(147)
10 dim a$(10),b$(10)
20 i=1
30 read a$(i),b$(i)
40 ? a$(i) tab(20) b$(i)
50 if a$(i)="r" then 100
60 i=i+1
```

```
70 goto 30
100 end
+ Datazeilen bei 10010ff
```

Der Rechner geht auf einer Schleife zwischen 30 und 70, bis die data-Zeilen abgearbeitet sind. Unser Zähler i=i+1 steht in 60.
In Basic gibt es nun eine sehr viel elegantere Methode, einen Zähler zu bauen, wenngleich dies nur mit einer zu Anfang eher unverständlichen Formel zu erreichen ist:

for-next-Schleife

```
for i=1 to 100
```

nebst dazugehörigem

```
next
```

Ausbau des Zählers i=i+1 und Einbau der for-next-Schleife ergeben folgenden Programmaufbau:

```
10 dim a$(20),b$(20)
20 for i=1 to 100
30 read a$(i),b$(i)
40 ? a$(i) tab(20) b$(i)
50 if a$(i)="r" then 100
60 next
100 end
```

In Zeile 20 wird dem Rechner mit "for i=1 to 100" gesagt, daß er bei jedem Durchgang durch diese Programmzeile den Wert i von 1 bis 100 jeweils um 1 erhöhen soll. Der erste Wert von i ist 1, der höchste mögliche Wert ist 100. Durch "next" aus Zeile 60 springt der Rechner zum "for i=1 to 100" aus 20 zurück.

An zwei Beispielen sollen ähnliche Funktionsweisen der for-next-Schleife verdeutlicht werden. Schreiben wir:

```
10 for i=1 to 10
20 ? "Molli"
30 next
40 ?:? "fertig!"
```

so wird die 10/30-Schleife zehnmal durchlaufen, i bei jedem Durchgang um 1 erhöht und "Molli" zehnmal auf den Schirm gebracht. Kehrt der Rechner zum 11. Mal über next nach 10 zurück, wird i=11, die Bedingung aus 10 (for i=1 to **10**) ist nicht erfüllt, der Rechner springt in die Programmzeile, die auf "next" folgt und sagt, daß er fertig ist.

In dem Programm:

```
10 for i=1 to 15
20 a=a+1
30 ? a
40 next
```

wird die 10/40-Schleife 15 mal durchlaufen, die Variable a wird in 20 bei jedem Durchgang um 1 erhöht und der jeweilige Wert von a in 30 auf den Schirm gedruckt. Wir erhalten die Zahlen von 1 bis 15.
Varianten:
30 ? a; (Die Zahlen erscheinen nebeneinander)
30 ? a, (Die Zahlen haben 8 Spalten Abstand)

Du hast in diesem Kapitel gelernt:

- daß Programmverzweigungen mit "if...then..." konstruiert werden.
- daß Programmzähler elegant mit der "for-next-Schleife" konstruiert werden können

Sequentielle Files

7. Kapitel

In diesem Kapitel machen wir uns mit einem sogenannten "sequentiellen File" vertraut, eine besondere Art, Daten auf der Diskette zu speichern. Wir lernen, wie wir Daten in ein sequentielles File speichern und aus diesem später wieder lesen. Wir erstellen uns auf diese Art ein Generatorprogramm für die Wortlisten unseres Vokabeltrainingsprogramms.

Wir kennen aus Kapitel 2 schon die Möglichkeit, unsere Programme mit save "...",8 auf die Diskette zu schreiben. In diesem Kapitel wollen wir unsere Daten (die Vokabeln) auf eine andere Art niederschreiben, so daß sie später von jedem beliebigen Programm aus in den Rechnerspeicher eingelesen werden können. Wir sagen, daß wir unsere Daten "in ein sequentielles File" schreiben, ohne im Augenblick im einzelnen darauf einzugehen, was dieser Begriff bedeutet. Damit der Rechner später die niedergeschriebenen Daten wiederfinden kann, müssen wir unserem Datenpaket einen Namen geben. Wir benennen einen Abschnitt der Diskette mit einem Namen, schreiben in diesen "Kasten" unsere Daten und verschließen ihn anschließend wieder. Sobald wir später die Daten benötigen, öffnen wir den "Kasten" wieder, holen die Daten heraus und können sie in unseren Programmen weiterverarbeiten.
Wir setzen unser Programm aus dem letzten Kapitel fort, indem wir statt "end" in 100 schreiben:

```
100 f$="vel"
```

"vel" soll der Name des ersten Vokabel-"Kastens" sein. "ve" steht für "Vokabeln Englisch", "1" steht für Vokabelliste Nummer 1. "vel" wird in der Stringvariablen f$ gespeichert. Es folgt:

open

10110 open 2,8,2,"*0:"+f$+"s,w"

Es ist dies eine Standardformel, die dem Anfänger wahrscheinlich wie eine obskure Zauberformel erscheint, die dem Rechner aber sagt, er solle auf der Diskette einen "Kasten" mit dem Namen "vel" eröffnen, um Daten dort abzulegen. Wir gehen auf die einzelnen Bestandteile der "Zauberformel" nicht ein. Du mußt sie auswendig lernen, ohne Fragen zu stellen.

Nach 110 fahren wir fort mit:

```
120 r=i
130 for i=1 to r
140 print#2,a$(i)
150 print#2,b$(i)
160 next
200 close2
210 clr
```

Beachte Zeile 120! Dort sagen wir, daß der Wert, der gerade in i steht, in r zwischengespeichert wird. In i steht in dem Augenblick, in dem der Rechner in 120 ankommt, die Anzahl der Vokabelpaare einschließlich des Markers "r".
Der neue Wert r ist der obere der beiden Grenzwerte der neuen for-next-Schleife 130/160 (for i=1 to r), die nur so oft durchlaufen wird, wie Vokabelpaare in den Feldvariablen a$() und b$() notiert sind.
In 140 und 150 werden die Vokabelpaare in das sequentielle File geschrieben. Beachte, daß sich die Schreibweise für die Datenausgabe an die Diskette kaum von der Ausgabe auf den Bildschirm mit "print" unterscheidet. Im Unterschied zum Bildschirmausdruck wird zum Beschriften der Diskette dem "print" lediglich "#2" angefügt.

print#2; close2

print#2

Beachte, daß "print#2" ausgeschrieben werden muß (niemals "? #2"!!) und von der folgenden Stringvariablen **durch ein Komma getrennt** wird!

Nachdem die for-next-Schleife r-mal durchlaufen wurde, alle Vokabelpaare und der Erkennungsbuchstabe "r" unserer Programmverzweigung niedergeschrieben wurden, geht der Rechner weiter nach 200, von wo aus er den "Kasten" vel wieder verschließt. Dieser Befehl

close

close ist sehr wichtig, damit die eben abgespeicherten Daten nicht abhanden kommen können. Erst nachdem der Rechner das sequentielle File geclosed hat, erlöscht das rote Lämpchen unserer Diskettenstation. Solltest Du einmal mit RUN/STOP das Programm abbrechen, während gerade ein sequentielles File bearbeitet wird, und das rote Lämpchen weiterleuchten, mußt Du es mit der Direkteingabe

close2

wieder zum Erlöschen bringen.

clr

In 210 schließlich endet Deine jetzige Programmerweiterung mit dem Befehl **clr**, durch den **sämtliche Variablen im Rechnerspeicher auf Null gesetzt werden!** Wenn Du nach dem Programmende mit "? a$(1)" das erste englische Wort Deiner Vokabelliste erfragen willst, antwortet der Rechner mit einer unwissenden Leerzeile. Auf "? i" folgt ebensowenig ein Zahlenwert, der größer als 0 ist wie auf "? r". Alle numerischen Variablen (hier i und r) haben jetzt den Wert 0, alle Stringvariablen haben 0 Buchstaben und ergeben im Ausdruck eine Leerzeile. Die erneute Ignoranz des Rechners liegt in dieser Phase der Programmbearbeitung durchaus in Deinem Interesse, denn wir wollen nachprüfen, ob die eben auf Diskette

geschriebenen Daten tatsächlich dort angekommen sind, und Du in der Lage bist, sie von dort wieder in den Rechnerspeicher zu laden. Du fährst fort:

```
220 f$="vel"
230 dim a$(20),b$(20)
230 open 2,8,2,f$+"s,r"
240 for i=1 to 100
250 input#2,a$(i)
260 input#2,b$(i)
270 ? a$(i) tab(20) b$(i)
280 if a$(i)="r" then 300
290 next
300 close2
```

In 220 definierst Du f$ neu, weil durch das voraufgegangene clr auch f$ auf Null gesetzt wurde. Aus dem gleichen Grund werden auch die Feldvariablen a$() und b$() neu dimensioniert. Du öffnest in 230 Dein sequentielles File (also Deinen Vokabeldatenkasten) durch eine zweite Zauberformel, die wie die erste ohne Zwischenfragen auswendig gelernt wird. In der for-next-Schleife 240/290 werden die Daten aus dem eröffneten sequentiellen File von der Diskette in den Rechnerspeicher gelesen. Der dazu notwendige Befehl ist **input#2,**. 280 prüft bei jedem Durchgang, ob die Bedingung a$(i)="r" erfüllt ist und geht im positiven Fall, nachdem alle Daten eingelesen wurden, weiter nach 300. Dort wird das sequentielles File mit "close2" ordnungsgemäß verschlossen.

input#2

Voilà, Dein erstes sinnvolles Programm ist fertig! Du hast ein **Generatorprogramm** geschrieben, mit dem Deine Vokabeln so auf die Diskette geschrieben werden, daß sie später von anderen Programmen wiederverwertet werden können. Hier das Programm noch einmal in der kompletten Übersicht:

```
1 ? chr$(14):? chr$(147)
5 f$="vel"
10 dim a$(20),b$(20),c$(20),d$(20)
20 for i=1 to 100
30 read a$(i),b$(i)
40 ? a$(i) tab(20) b$(i)
50 if a$(i)="r" then 100
60 next
100 stop
110 r=i
120 open 2,8,2,"@0:"+f$+"s,w"
130 for i=1 to r
140 print#2,a$(i)
150 print#2,b$(i)
160 next
170 close2
180 ? chr$(147)
200 open 2,8,2,f$+"s,r"
210 for i=1 to 100
220 input#2,c$(i)
230 input#2,d$(i)
240 ? c$(i) tab(20) d$(i)
250 if c$(i)="r" then 300
260 next
300 close2
10000 rem *** "Hier Vokabelpaare in data-Zeilen
durch Komma getrennt niederschreiben
10999 data r,r
59999 end
60000 save " 0:del",8
60001 verify "del",8
```

Wir haben "clr" wieder entfernt, womit die Neudefinition von f$ und die erneute Dimensionierung von a$() und b$() entfällt, und speichern stattdessen in der Lese- und Kontrollphase (Programmnummern 200 ff) die Vokabeln in die Feldvariablen c$() und d$() ein, die in 10 zusätzlich dimensioniert werden. Nachdem wir das Programm durch ein "run 60000"

unter dem Namen "del" (Daten Englisch 1)gesichert haben, löschen wir alle data-Zeilen außer 10999, verändern den Programmnamen in 60000 und 60001 wieder in "generator" und speichern auch dieses Programm durch "goto 60000" auf Diskette.

Wenn Du neue Vokabeln eingeben willst, gehst Du in Zukunft folgendermaßen vor:

1.
Du lädst "generator" ein und veränderst die Kennzahl für das sequentielle File in Zeile 5. Aus "vel" wird so "ve2","..3","..4" usw.
Du veränderst den Filenamen "generator" in "de2", "de3", "de4" entsprechend der Kennzahl der Vokabellektion.

2.
Du schreibst nach 10000 Deine Vokabelpaare in
rem data-Zeilen. **rem** heißt hier, daß der Rechner diese Zeile nicht bearbeitet; der Inhalt einer Rem-Zeile ist nur eine Gedächtnisstütze für den Programmierer, damit er sich später in seinen Programmen zurechtfindet.
Denke daran, daß Du im Augenblick nicht mehr als 20 Vokabelpaare eingeben kannst. Willst Du mehr als 20 eingeben, mußt Du die Dimensionierung der Feldvariablen in 10 erhöhen, etwa dim a$(30), dim a$(49) etc, immer entsprechend der maximal zu speichernden Vokabelzahl.

3.
Nach Eingabe der Vokabeln startest Du das Programm und prüfst, ob nach dem Programmstop
stop aus 100 die beiden "r" korrekt in einer Zeile stehen. Ist dies nicht der Fall, wurde in den data-Zeilen ein Fehler gemacht. Beliebter Fehler:
- Nur ein Wort eingegeben.

- Komma zwischen englischem und deutschem Wort vergessen.
Ist alles korrekt, schreibst Du im Direktmodus

cont cont

und gibst RETURN, womit der Rechner im Programm fortfährt und die Daten auf die Diskette schreibt.

4.
Du schreibst "goto 60000" und sicherst Deine Datenbank. Wenn später Rechtschreibefehler in den Daten festgestellt werden, rufst Du die entsprechende Datenbank wieder auf (load "del",8; load "de2",8 etc je nach Kapitel), korrigierst die Fehler in den data-Zeilen, gibst RUN und speicherst nach der Niederschrift der Daten in das sequentielle File auch die Datenbank wieder durch "goto 60000" ab.

Du hast in diesem Kapitel gelernt:

- daß es eine besondere Art gibt, Daten auf der Diskette für die Verwendung in späteren Programmen zuzubereiten: das sequentielle File.
- daß ein sequentielles File durch die Formel open 2,8,2,"@0:"+f$+"s,w" für die Niederschrift von Daten vorbereitet wird.
- daß Daten durch "print#2" in ein sequentielles File geschrieben werden.
- daß ein sequentielles File durch die Formel open 2,8,2,f$+"s,r" zum Lesen der Daten eröffnet wird.
- daß Daten durch "input#2" aus einem sequentiellen File gelesen werden.
- daß sequentielle Files durch "close2" geschlossen werden.
- daß hinter "rem" Gedächtnisstützen für den Programmierer stehen, die vom Rechner nicht beachtet werden.
- daß der Programmablauf mit "stop" unterbrochen und mit der Direkteingabe "cont" wieder aufgenommen wird.

8. Kapitel

Wir lernen in diesem Kapitel die Anweisung "input" kennen, mit der wir den Rechner eine Frage stellen lassen. Wir begegnen der Funktion "get", die uns die Konstruktion einer "Endlosschleife" gestattet und die den Programmablauf solange aufhält, bis eine beliebige Taste gedrückt wird. Schließlich prüfen wir das Vokabelwissen unserer Schüler und leiten gegen Ende des Programms zum "Endmenu" über.

Wir beginnen unser Hauptprogramm mit 10000, weil wir die niedrigen Zeilennummern später noch für andere Zwecke benötigen. Als erstes vermerken wir in einer rem-Zeile, daß es sich um das Hauptprogramm handelt. Danach löschen wir den Schirm und lassen den Rechner nach der Nummer des Kapitels fragen, das bearbeitet werden soll. Es versteht sich, daß wir zuvor mit Hilfe des Generatorprogramms ein Kapitel mit einer entsprechenden Nummer hergestellt haben:

```
10000 rem *******  "Hauptprogramm" *********
10010 ? chr$(147):? chr$(14)
10020 input "Bitte Kapitelnummer eingeben";a$
10030 f$="ve"+a$
```

sowie

```
59999 end
60000 save "@0:vokabel",8
60001 verify "vokabel",8
```

input

Durch 10020 bringen wir die Wörter, die zwischen den Anführungszeichen stehen, auf den Schirm wie durch eine print-Anweisung. **input** bedeutet dem Rechner aber weiter, daß er auf eine Antwort des Benutzers zu warten habe, die durch RETURN bestätigt werden muß und dann in die Variablen gespeichert wird, die von dem letzten

Anführungszeichen durch ein Semikolon abgetrennt ist. Geben wir auf die Frage nach der Kapitelnummer eine 1 ein, so wird "1" in a$ gespeichert. Da a$ eine Stringvariable ist, wird 1 wie ein Buchstabe und nicht wie eine Zahl behandelt. Buchstaben aber kann man miteinander zu neuen Wörtern verbinden, wie in 10030 geschehen. Dort wird in f$ abgelegt, was auf der rechten Seite des Gleichheitszeichens durch das Pluszeichen verschmolzen wird.
Durch das Pluszeichen können beliebig viele Wörter miteinander verbunden werden. Wenn wir die drei Variablen a$, b$ und c$ definieren als:

```
a$="Mollis"
b$="+"
c$="Muslis"
```

so lassen sich diese drei Stringvariablen in einer vierten zusammenfassen und auf den Schirm bringen:

```
d$=a$+b$+c$
? d$
```

Antworten die Schüler auf die Frage aus 10020 mit 1, so wird f$ zu "ve"+"1", zu "vel". Dies ist der Name, unter dem wir unsere erste Vokabelliste als sequentielles File auf die Diskette geschrieben haben und unter dem wir jetzt unsere Daten wiederfinden:

```
10040 open 2,8,2,f$+"s,r"
10050 dim a$(20),b$(20)
10060 for i=1 to 100
10070 input#2,a$(i)
10080 input#2,b$(i)
10090 if a$(i)="r" then 10200
10100 next
10200 close2
```

```
10210 r=i-1
10220 ? chr$(147)
10230 for i=1 to r
10240 ? a$(i) tab(20) b$(i)
10250 next
```

Das Programm ist von seinem Aufbau schon aus den letzten Kapiteln bekannt. In 10210 wird die Anzahl der eingelesenen Vokabelpaare in r gespeichert (i-1, weil das letzte Wortpaar der Marker "r","r" war). Der Schirm wird gelöscht und die 10230/10250-Schleife blendet die Vokabeln ein. Nachdem sich unser Schüler auf diese Weise den Inhalt des von ihm gewünschten Kapitels auf den Schirm geholt hat, wollen wir ihm sagen, er könne durch das Betätigen irgendeiner Rechnertaste zur Prüfung übergehen. Wir lassen zu diesem Zweck zwei Leerzeilen ausdrucken und projizieren unsere Anweisung:

```
10260 ?:?
10270 ? "Fortsetzen mit beliebiger Taste"
```

Es folgt die "Endlosschleife":

```
10280 get a$:if a$="" then 10280
```

get

Durch **get** sagen wir dem Rechner, er solle unablässig die Tastatur daraufhin abfragen, ob dort irgendeine Taste gedrückt wird. Solange unsere Schüler vor dem Schirm sitzen und Vokabeln lernen, drücken sie keine Taste, und a$ -das der Rechner ja ohne Unterlaß gettet- ist gleich Null oder, zwischen Anführungszeichen geschrieben: a$=""! Damit ist die Bedingung aus 10280 erfüllt und der Rechner geht wie angeordnet zurück an den Anfang von 10280, gettet wieder a$ usw. Erst in dem Augenblick, in dem eine Taste gedrückt wird, nimmt a$ einen Wert an, der von "" verschieden ist. Die

Prüfung der Antwort

Bedingung aus 10280 ist nicht mehr erfüllt und der Rechner geht weiter im Programm. Dort findet er nun vor:

```
10300 for i=1 to r
10310 ? chr$(147)
10320 ? b$(i)
10330 input an$
10340 if an$=a$(i) then 10380
10350 ? "Falsch!! - Richtig wäre gewesen:"
10360 ? a$(i)
10361 for k=1 to 3000:next
10370 goto 10390
10380 ? "WOW!! - Total richtig!! - Supergut!!"
10390 for k=1 to 1000:next
10400 next
```

Was hier innerhalb der 10300/10400-Schleife geschieht, ist die eigentliche Prüfungssequenz! Die Schleife wird r-mal durchlaufen, solange nämlich bis alle b$(), die eingespeicherten deutschen Wörter, einmal als Frage auf den Schirm eingeblendet wurden. Ebenfalls r-mal gibt der Rechner in 10330 mit einem Fragezeichen zu erkennen, daß er auf eine Antwort wartet, die in an$ abgelegt wird.
In der folgenden Zeile spaltet sich das Programm je nach den Leistungen des Schülers auf. Entweder an$ ist identisch mit a$(i), der englischen Entsprechung von b$(i), und der Rechner läßt auf die erfüllte Bedingung von 10340 die Konsequenz "goto 10380" folgen, wo ein zufriedener Kommentar auf dem Schirm erscheint. War an$ hingegen verschieden von a$(i) und die Antwort daher falsch, kommt vom Rechner aus 10350 eine Nachricht über die fehlerhafte Leistung, während aus 10360 die richtige Antwort eingeblendet wird. Zeile 10370 schließlich überspringt den begeisterten Kommentar aus 10380 und leitet direkt zu 10390 über. Hier

steht, wie auch in 10361, eine sogenannte "Warteschleife", die dafür sorgt, daß der Rechner sich eine Zeitlang beschäftigt und die Schüler sich den Kommentar ansehen können. In "for k=1 to 1000:next" wird dem Rechner nichts anderes gesagt, als still für sich von 1 bis 1000 zu zählen. Dafür benötigt er ungefähr 1 Sekunde (Bitte unbedingt als Schleifenvariable k wählen; würdest Du i wählen, würde der Rechner sich die 10300/10400-Schleife durcheinanderbringen!!). Nachdem er bei 4000 angekommen ist, geht der Rechner über 10400 zurück nach 10300, solange, bis die Schleife 10300/10400 r-mal durchlaufen wurde und alle deutschen Wörter aus b$() einmal zur Übersetzung ins Englische angeboten hat.

Damit der Rechner nach 10400 nicht ins Leere stürzt, bauen wir eine Überleitung ein, damit die Schüler auf Wunsch entweder das gleiche Kapitel noch einmal oder aber ein anderes Kapitel bearbeiten können:

```
20000 rem ***** "Endmenu" *****
20001 ? chr$(147)
20010 ?:?:?:? "            Ende der Prüfung"
20020 ?:?:?:?:?
20030 ? "          <0> Ende"
20035 ? "          <1> Das gleiche Kapitel"
20040 ? "          <2> Ein anderes Kapitel"
20050 get a$
20060 if a$="0" then ? chr$(147):end
20065 if a$="1" then 10220
20070 if a$="2" then run
20080 goto 20050
```

Der Schirm wird gelöscht, in der 4.Zeile erscheint etwas eingerückt die Bemerkung über das Ende der Prüfung. Nach 5 Leerzeilen werden die Wahlmöglichkeiten aufgeführt und in 20050

wartet der Rechner auf eine Antwort. Solange wir keine Taste drücken, läuft er unablässig von 20050 bis 20080 und von dort zurück nach 20050. Erst wenn wir eine 0, eine 1 oder eine 2 eingeben, wird er aus dem 20050/20080-Kreis entlassen: - Bei "0" wird der Bildschirm gelöscht und "ready" gibt bekannt, daß der Rechner auf die Kommandoebene zurückgekehrt ist.
- Bei "1" geht der Rechner nach 10220 zurück und blendet die Vokabelpaare des gleichen Kapitels noch einmal zur Übersicht ein.
- Bei "2" startet der Rechner das Programm erneut und bittet wieder um die Eingabe der Kapitelnummer.

Wir haben unser erstes Vokabeltrainingsprogramm geschrieben und sollten mit uns zufrieden sein. Dennoch ist unser Werk ein Schrottprogramm, das so oder ähnlich zwar im Handel für nicht wenig Geld angeboten wird, mit dem unsere Schüler aber auf Dauer nicht vernünftig lernen können. Im nächsten Kapitel die ersten Verbesserungen...

Wir haben gelernt:

- daß Fragen an den Benutzer mit "input" auf den Bildschirm projiziert werden und die Antwort entweder in eine Stringvariable oder in eine numerische Variable zwischengespeichert wird.
- daß mehrere Strings durch ein Pluszeichen zu einem neuen String vereint werden können.
- daß mit "get" die Tastatur vom Rechner daraufhin abgefragt wird, ob dort irgendeine Taste gedrückt wird. Der Tastenbuchstabe wird in einer Stringvariable gespeichert. Wenn keine Taste gedrückt wird, ist der Inhalt der Stringvariablen gleich 0: ""
- daß wir den Rechner mit "for i=1 to 4000:next" etwa 4 Sekunden beschäftigen können.

9. Kapitel

Wir gebrauchen in diesem Kapitel noch einmal einen Programmzähler, um festzustellen, wieviel richtige Antworten der Schüler gegeben hat. Wir errechnen den Prozentsatz der richtigen Antworten und die benötigte Prüfungszeit und blenden diese Werte nach der Prüfung ein.

Es ist weder Programmier- noch Überwachungsmanie, die uns dazu führt, in unser Programm eine Uhr und einen Zähler der richtigen Antworten einzubauen, mit denen wir die Leistungen unserer Schüler überprüfen. Es ist in der Tat sehr wichtig, nach jeder Arbeit eine Rückmeldung über das zu erhalten, was gerade geleistet wurde. Da es außerdem nur sehr selten vorkommt, daß die Schüler von einer Prüfung zur anderen schlechter werden, ist es nur motivierend, am Ende jeder Prüfung einen Überblick über den Prüfungsablauf zu erhalten. Wir fügen deshalb in unser Programm zusätzlich ein:

```
10385 v=v+1
```

und zählen damit, wie oft der Rechner an der Begeisterungsbekundung des Programms vorbeikommt und eine Frage richtig beantwortet wurde.
Außerdem schreiben wir:

```
10290 ti$="000000"
```

und setzen damit die rechnerinterne Uhr auf Null. ti$ ist die vom Rechner festgesetzte Variable, in der 60 Mal pro Sekunde die Zeit seit Einschalten der Stromzufuhr oder seit der letzten Null-Setzung gespeichert wird. Die Zeit wird in der Form "000000" ausgegeben, so daß 13 Stunden, 7 Minuten und 1 Sekunde "130701"

```
? ti$
```

ergeben. Wenn wir die Uhr vor der Prüfung auf Null setzen, können wir sie nach der Prüfung ablesen lassen und in t$ zwischenspeichern:

```
10600 rem "Auswertung des Prüfungsergebnisses"
10610 t$=ti$
```

Das Programm wird wie folgt fortgesetzt:

```
10620 ? chr$(147)
10625 p=v/r*100
10630 ? "            Prüfungsergebnis:"
10640 ?:?:?:?:?
10650 ? "Prüfungszeit:          ";t$
10660 ?:? "Richtige Antworten:     ";v
10670 ?:? "Richtige Antworten in % ";p
10680 ?:?:?
10690 ? "Fortsetzung mit beliebiger Taste"
10700 get a$:if a$="" then 10700
10710 v=0
```

Nachdem der Schirm gelöscht wurde, wird in 10625 der Prozentsatz der richtigen Antworten ausgerechnet und in der Variablen p gespeichert. Der Prozentsatz errechnet sich aus dem Bruch **Richtige Antworten durch Gesamtzahl der Fragen multipliziert mit 100.**
Beachte die doppelten, durch einen Doppelpunkt getrennten Frage-(print-)Zeichen in 10660 und 10670 zur Erzeugung einer Leerzeile zwischen den einzelnen Ergebnissen. Beachte ferner das

; **Semikolon** vor v bzw. p in 10660 und 10670, durch das die Werte von v bzw. p unmittelbar an den Inhalt der Anführungsstriche noch in die gleiche Bildschirmzeile geschrieben wird.
Die beiden letzten Zeilen sind uns bereits bekannt. Wir blenden auf den unteren Bildschirm die Anweisung ein, mit einer beliebigen Taste das Programm fortzusetzen und getten a$ in einer Ein-Zeilen-Schleife. Das Drücken

irgendeiner Taste entläßt den Rechner aus 10700, und v wird wieder auf Null gesetzt.

Wir haben gelernt:

- daß im Rechner eine Uhr läuft, die 60 Mal pro Sekunde die Zeit mißt. Mit ? ti$ kann so jederzeit die Zeit gemessen werden, die seit dem Einschalten des Gerätes oder seit der letzten Null-Setzung verstrichen ist.
- daß die Uhr mit ti$="000000" auf Null gesetzt wird.
- daß die Zeit in der Form "000000" ausgegeben wird, wobei die erste Ziffernpaar die Stunden, das zweite die Minuten und das dritte die Sekunden anzeigt.
- daß sich der Prozentsatz der richtigen Antworten mit der Formel v/r*100 errechnet

i=rnd(1)*r+1

10. Kapitel

Wir lernen in diesem Kapitel, wie wir den Rechner Zufallszahlen (vgl. Lottozahlen) bilden lassen. Zufallszahlen helfen uns danach, die Vokabeln eines Kapitels in immer anderer Reihenfolge zum Übersetzen anzubieten. Dummes Auswendiglernen einer Fragensequenz wird den Schülern damit unmöglich gemacht. Wir lernen schließlich, in einer Feldvariablen zu speichern, ob eine Frage bereits gestellt wurde oder nicht.

Wer probeweise ein Kapitel mehrmals bearbeitet hat, mußte feststellen, daß die Fragen in immer der gleichen Reihenfolge eingeblendet werden. Effektives Lernen wird dadurch erschwert, denn die Schüler wissen am Ende, daß "dissident" nach "to besiege" gefragt wird und denken oft gar nicht mehr an den Zusammenhang mit dem deutschen Wort. Um diesem Mißstand abzuhelfen, greifen wir auf den rechnerinternen "Zufallsgenerator" zurück.

Zufallszahlen werden mit der Formel (bitte sofort und endgültig auswendig lernen!!):

rnd

(Zufallszahl) = rnd(1) * r + 1

gebildet, wobei eine Zahl zwischen 1 und r gezogen wird:

```
i=rnd(1)*100+1
? i
```

ergibt eine Zahl zwischen 1 und 100.
Das Ergebnis ist selten eine ganze Zahl, sondern hat zumeist Stellen hinter dem Komma. Um hieraus eine ganze Zahl zu erhalten, müssen wir uns der

int()

Funktion **int(...)** bedienen:

```
i=rnd(1)*r+1
? i
ii=int(i)
? ii
```

Durch int() werden alle Stellen hinter dem Komma abgestrichen. Wir schreiben unsere Formel der Zufallszahl neu und erhalten:

```
(Zufallszahl)=int(r(1)*r+1)
```

Wenn rnd(1)*r+1 7.445 ergeben sollte, so erhalten wir im gleichen Fall durch int(rnd(1)*r+1) den Wert 7.
(Schreibe jetzt auch: 10626 p=int(p))

Wir kehren zu unserem Programm zurück und ändern bzw. fügen ein:

```
10001 dim a(20)
10300 for j=1 to r
10315 i=int(rnd(1)*r+1)
10316 if a(i)=1 then 10315
10317 a(i)=1
10710 v=0:for i=1 to r:a(i)=0:next
```

und sehen, daß nach dem Neustart des Programms die Fragen stets in unterschiedlicher Reihenfolge auf den Bildschirm gebracht werden. In 10001 haben wir zuerst eine sogenannte

a() **Gedächtnis-Feldvariable** mit dem Namen a dimensioniert. Die Variable aus 10300 haben wir in j umgetauft, weil wir i für die Zwischenspeicherung der Zufallszahl benötigen. Danach folgen die Programmzeilen 10315 bis 10317, die einer genaueren Erklärung bedürfen.

Indem wir die Feldvariable a(20) dimensionieren, setzen wir sämtliche dadurch geschaffene

Variablen auf Null. a(1) ist 0, so wie auch a(2),a(3),a(4) bis a(20) gleich Null sind. Wenn der Zufallsgenerator in 10315 den ersten Wert für i gezogen hat, prüft der Rechner in 10316, ob a(i) gleich 1 ist. Wir sahen gerade, daß noch alle a()-Werte gleich Null sind, die Bedingung von 10316 ist nicht erfüllt, und der Rechner geht weiter nach 10317, wo a(i) gleich 1 gesetzt wird. Die zu i gehörige Frage wird gestellt, auf die Antwort gewartet, der Rechner kehrt in seiner Schleife zurück und zieht die zweite Zufallszahl. Sollte jetzt (oder später) die zweite Zahl identisch sein mit der, die im ersten Durchlauf gezogen wurde, so ist die Bedingung aus 10316 erfüllt. Der Rechner wird zurück nach 10315 geschickt, um eine neue Zufallszahl zu ziehen. In 10316 also wird ihm gesagt, daß er eine Zahl suchen soll, die bisher noch nicht erschienen ist und daher einen a(i)-Wert von Null hat. Auf diese Weise wird der Rechner nur dann in das weitere Programm entlassen, wenn er eine Zahl zieht, die bisher noch nicht gezogen wurde. Nachdem er alle Zahlen gezogen hat, ist auch die Schleife "for j=1 to r" erschöpft, und der Rechner geht zur Schlußbewertung weiter. Jede Frage ist nur einmal erschienen. Die Reihenfolge der Fragen ist bei jeder Prüfung verschieden.
Beachte 10710, wo zusätzlich zu v=0 nach jeder Prüfung noch gesagt wird, alle Werte von a() auf Null zu setzen:

```
for i=1 to r:a(i)=0:next
```

Unser Programm ist jetzt schon um einiges leistungsfähiger als noch nach Kapitel 8. Dennoch besteht kein Grund zur Zufriedenheit. Die Bemerkungen des Rechners nach jeder Frage - ob begeistert oder unzufrieden- sind nur am Anfang lustig, auf die Dauer aber nur nervig und gehören aus dem Programm genommen. Wie wir

später noch sehen werden, können in einer Stunde bis zu 100 Vokabeln gelernt werden. Das heißt, daß in einer Stunde etwa 400 Antworten gegeben werden. 400 Mal aber "WOW, das war wieder einmal supergut!!" zu hören, ist selbst für Lernfrustrierte zuviel. Der Rechner hat daher in Zukunft seine Arbeit kommentarlos zu verrichten.

Ein weiteres Ärgernis ist die wenig elegante Art, in der Fragen und Antworten auf dem Schirm lokalisiert sind. Wir werden in den nächsten Kapiteln daher versuchen, die Ausgaben auf dem Bildschirm formschöner in die Schirmmitte zu rücken und lernen dabei die Welt der gosub's kennen...

Lotto

Zum Schluß noch ein kleines Übungsprogramm mit eventuell lukrativer Anwendung der rnd-Funktion:

```
10 ? chr$(147):? chr$(14):dim a(49)
20 for k=1 to 7
30 i=int(rnd(1)*49+1)
35 if a(i)=1 then 30
36 a(i)=1
40 if k=7 then 70
50 ? k;".Zahl:    ";i
60 goto 100
70 ?:? "Zusatzzahl: ";i
100 next
```

Wir haben gelernt:

- daß Zufallszahlen zwischen 1 und r mit der Formel "rnd(1)*r+1" erzeugt werden.
- daß mit der Funktion "int()" Stellen hinter dem Komma abgeschnitten werden.
- daß wir in einer "Gedächtnis-Feldvariablen" speichern können, welche Vokabeln schon gefragt wurden.
- wie der Rechner auf dem Weg zum Lottogewinn helfen kann...

Koordinatennetz

11. Kapitel

Wir lernen in diesem Kapitel, wie wir Wörter gezielt auf den Schirm projizieren können und machen dadurch unser Trainingsprogramm stilschöner. Wir begegnen zum ersten Mal dem Befehl **poke**, einem schwer verständlichem Instrument mit noch ungeahnten Möglichkeiten. Wir löschen Bildschirmzeilen durch einen eleganten Trick und werden in die Geheimnisse der Unterprogramme eingeweiht. Wir lassen Wörter über den Schirm kriechen und hüpfen.

Die Fragen, die das Trainingsprogramm dem Schüler stellt, werden immer an den linken Schirmrand geschrieben und machen dadurch einen wenig gefälligen Eindruck. Ziel dieses Kapitels ist es, den Rechner dazu zu bewegen, Fragen und Antworten mehr in die Bildschirmmitte zu projizieren. Wir müssen uns zunächst erinnern, daß der Bildschirmbereich aus 24 Zeilen besteht. Jede Zeile hat 40 Spalten. Der Rechner zählt die Zeilen von 0 bis 23, die Spalten von 0 bis 39. Dadurch entseht folgendes Koordinatennetz:

(Die Zeichnung ist leider abhanden gekommen...)

Wir sehen, daß Zeilen und Spalten von der **linken oberen Bildschirmecke** nach unten bzw. rechts gezählt werden. Wir bezeichnen in Zukunft die Zeilen mit der Variablen y, die Spalten mit x und können jeden der 960 Punkte (24*60) auf dem Bildschirm durch x und y bestimmen. Den punktierten Bildschirmpunkt unserer Zeichnung erhalten wir, indem wir x als 10 und y als 6 definieren; der gestrichelte ist durch x=18 und y=10 definiert, das freie Kästchen entspricht x=31 und y=8. Wir schreiben das folgende kleine Demonstrationsprogramm:

```
10 ? chr$(147)
20 input "x-Wert";x
25 if x>39 then 20
30 input "y-Wert";y
35 if y>23 then 30
40 input "Wort";a$
50 poke 211,x:poke 214,y:sys 58640
60 ? a$
70 get a$:if a$="" then 70
80 goto 10
```

poke

Nach dem Programmstart fragt der Rechner zuerst nach dem x-, dann dem y-Wert und prüft in 25 bzw. 35, ob die eingegebenen Werte die Grenzen des Bildschirms nicht sprengen. Nachdem auch ein beliebiges Wort eingegeben und mit RETURN bestätigt wurde, erscheint unser Wort an der Stelle des Bildschirms, die wir durch unsere x- und y-Koordinaten definiert hatten. Durch das Betätigen einer beliebigen Taste geht das Programm zurück nach 10 und fragt nach neuen Werten.
Kern- und Glanzstück des kleinen Programms ist Zeile 50, die wieder als Zauberformel betrachtet und ohne weitere Zwischenfragen auswendig gelernt wird: diese Formel -vor ein print oder ein input gestellt- bringt die folgende Bildschirmausgabe in die gewünschte Position.

Ein weiteres Übungsprogramm:

```
10 ? chr$(147)
20 input "Wort";a$
30 ? chr$(147)
40 for x=1 to 20
50 poke 211,x:poke 214,10:sys 58640:? a$
60 for i=1 to 200:next
70 poke 781,10:sys 59903
80 next
90 goto 10
```

In 20 wird nach einem beliebigen Wort gefragt und danach der Bildschirm gelöscht. Es folgt eine for-next-Schleife (40/80), in der ein x-Wert von 1 bis 20 hochgezählt wird. Dieser x-Wert dient in 50 zur Plazierung des eingegebenen Wortes auf den Bildschirm in die x. Spalte und die 10. Zeile (poke 214,**10!!**). Dort bleibt das Wort solange stehen, bis der Rechner in 60 still für sich bis 200 gezählt hat. In 70 schließlich begegnet er einer weiteren Zauberformel (auswendig lernen!!), die ihm sagt, er solle die 10. Schirmzeile wieder löschen (poke 781,**10**:sys 59903!!). Sofort hiernach gelangt der Rechner nach 80, kehrt nach 40 zurück, erhöht den letzten x-Wert um 1 und plaziert so das Wort nun um eine Spalte nach rechts versetzt. Es entsteht der Eindruck, als krieche ein Wort über den Bildschirm.
Ein letztes Übungsprogramm, um die Vorteile unserer neuen Formel zu demonstrieren:

```
10 a$="Molli"
20 x=int(rnd(1)*30+1)
30 y=int(rnd(1)*23+1)
40 poke 211,x:poke 214,y:sys 58640:? a$
50 for i=1 to 200:next
60 poke 781,y:sys 59903
70 goto 20
```

In 20 und 30 werden zwei Zufallszahlen gebildet, für x eine Zahl zwischen 1 und 30, für y eine Zahl zwischen 1 und 23. Durch diese beiden Werte wird ein Punkt auf unserem Bildschirm definiert, auf den der erste Buchstabe von "Molli" geschrieben wird. Dort verharrt der "Molli", bis der Rechner in 50 bis 200 gezählt hat. In 60 wird die Zeile mit dem Wert y wieder gelöscht, der Rechner geht nach 20 zurück, zieht sich zwei neue Werte für x und y usw. Wir erhalten einen Molli, der im Zickzack über den Schirm huscht..

Bevor wir die neuen Erkenntnisse sinnvoll in unser Trainingsprogramm einbauen, müssen wir uns noch mit der Gestaltung von Unterprogrammen beschäftigen. Unterprogramme schreiben wir in Zukunft immer dann, wenn eine bestimmte Befehlsfolge mehr als einmal im Laufe des Programms vorkommt. Anstatt die gleiche Befehlsfolge doppelt und dreifach und noch öfter zu schreiben, legen wir sie an einer bestimmten Stelle des Programms ab und springen vom Hauptprogramm immer dann in das so geschaffene Unterprogramm hinab, wenn wir es brauchen. So können wir ein Programm zum Projizieren von drei Namen auf den Bildschirm zwar wie folgt schreiben:

Variante 1:

```
10 ? chr$(147)
20 poke 211,10:poke 214,6:sys 58640
30 ? "Hinz"
40 poke 211,18:poke 214,10:sys 58640
50 ? "Kunz"
60 poke 211,31:poke 214,8:sys 58640
70 ? "Fritz"
```

doch ist die Schreibarbeit mit poke's, Zahlen und sys nur lästig. Viel eleganter ist die Variante mit den **gosub/return**-Befehlen:

gosub
return

Variante 2:

```
1 goto 10
5 poke 211,x:poke 214,y:sys 58640
6 return
10 x=10:y=6:gosub 5
20 ? "Hinz"
30 x=18:y=10:gosub 5
40 ? "Kunz"
```

```
50 x=31:y=8:gosub 5
60 ? "Fritz"
```

Ohne Zweifel ist die zweite Variante kürzer und wir erkennen sofort die Nützlichkeit von Unterprogrammen. In der Tat kam in der ersten Variante die Formel "poke 211,x:poke 214,y:sys 58640" dreimal vor. Grund genug, sie in ein Unterprogramm zu verlagern, das in der letzten Variante nur aus den Programmzeilen 5 und 6 besteht. In Zeile 5 wird die Formel untergebracht, aus Zeile 6 geht der Rechner mit **return** wieder ins Hauptprogramm zurück.
Das Unterprogramm wird zum Programmstart mit

```
1 goto 10
```

übersprungen. In 10 werden zuerst die Werte für x und y festgelegt, mit denen der Rechner durch **gosub** 5 in das Unterprogramm springt. Das "return" aus 6 sagt dem Rechner beim ersten Unterprogrammbesuch, daß er in jene Zeile des Hauptprogramms zurückkehren soll, die auf das letzte "gosub" folgt. Wenn der Rechner von Zeile 10 aus ins Unterprogramm ging, schickt ihn "return" also nach Zeile 20 zurück. Dort druckt er "Hinz" auf den Bildschirmpunkt 10/6. In 30 werden neue Werte für x und y definiert, der Rechner springt erneut ins Unterprogramm und kehrt mit "return" dieses Mal nach 40 zurück usw.

Wir können schon jetzt sich oft wiederholende Befehle unseres Trainingsprogramms in Unterprogramme bzw. Unterprogrammzeilen schreiben. Wir überspringen die Zone der Unterprogramme, die wir nun einführen mit:

```
1 goto 10000
```

und schreiben das lästige "? chr$(147)" zum Löschen des Bildschirms in ein Unterprogamm:

```
10 ? chr$(147):return
```

Wo im Programm der Schirm gelöscht werden muß, reicht in Zukunft ein schlichtes und bequem zu schreibendes: gosub 10...

Wir haben gelernt:

- daß der Bildschirm in ein Koordinatennetz aus 24 Zeilen und 40 Spalten aufgeteilt ist und der Nullpunkt dieses Koordinatennetzes in der linken oberen Bildschirmecke liegt.
- daß mit poke 211, poke 214 und sys 58640 nebst nachfolgendem "print" oder "input" ein Wort in jede gewünschte Bildschirmpositionen projiziert wird.
- daß es bei oft wiederkehrenden identischen Befehlen sinnvoll sein kann, Unterprogramme zu schreiben.
- daß Unterprogramme vom Rechner mit "gosub" angesteuert werden.
- daß Unterprogramme vom Rechner mit "return" verlassen werden.
- daß der Rechner nach der Rückkehr aus einem Unterprogramm in die Programmzeile nach dem letzten "gosub" einkehrt.

12. Kapitel

Wir verwerten die Erkenntnisse des letzten Kapitels, um Fragen und Antworten in gefälliger Form auf den Schirm zu projizieren. Wir bauen die Frage-Antwort-Sequenz in ein Unterprogramm ein und führen während der ganzen Prüfung in der ersten Zeile Buch über den aktuellen Prüfungsstand. Schließlich verändern wir die Farbe des Bildschirmhintergrundes, damit die Arbeit am Schirm die Augen der Schüler so wenig wie möglich belastet.

Wir streichen alle Zeilen zwischen 10300 und 10400 und schreiben statt dessen:

```
10300 rem ******  "Prüfung"  ******
10301 gosub 10
10302 ti$="000000"
10310 i=int(rnd(1)*r+1)
10320 if a(i)=1 then 10310
10330 a(i)=1
10340 fr$=b$(i):ub$=a$(i)
10350 f=f+1:gosub 2000:rem "Frage/Antwort"
```

und

```
20 poke 211,x:poke 214,y:sys 58640:return
90 poke 781,y:sys 59903:return
```

und

```
2000 rem ***** "Frage-Antwort-Sequenz" *****
2020 x=10:y=10:gosub 20
2030 ? fr$
2040 x=8:y=13:gosub 20
2050 input an$
2060 if an$=ub$ then 2200
2070 gosub 90
2080 x=10:y=15:gosub 20
2090 ? ub$
2100 ri=2:goto 2040
```

```
2200 for y=10 to 15:gosub 90:next
2210 return
```

Die Zeilen 10300 bis 10350 sind von ihrem Aufbau bekannt. In 10310 wird die Zufallszahl gezogen und in 10320 geprüft, ob die dazugehörige Vokabel schon erschienen ist. Wenn nicht, wird in der nächsten Zeile Buch darüber geführt, daß sie nun da war. Neu ist lediglich 10350, wo das deutsche Wort nach fr$ (**F**Rage) und das englische Wort nach ub$ (**UB**ersetzung) hinübergespeichert wird. Mit den so jeweils neu definierten Strings fr$ und ub$ geht der Rechner in das Unterprogramm nach 2000, nachdem er zuvor mit f=f+1 die Anzahl der gestellten Fragen vermerkt hat.
In 2020 werden die Koordinaten 10/10 für die Frage festgelegt, die in 2030 ausgedruckt wird. In 2040 wird die Koordinate 8/13 für das Fragezeichen des input's festgelegt, das den Schüler zur Niederschrift der Antwort auffordert. Wenn die Antwort mit der Übersetzung ub$ identisch ist, geht der Rechner nach 2200, löscht die Bildschirmzeilen 10 bis 15 und verläßt das Unterprogramm nach 10360.
War die Antwort falsch, geht der Rechner von 2070 aus in das Unterprogramm bei 90 und streicht Zeile 13 wieder aus. Anschließend schreibt er aus 2080/2090 die richtige Antwort auf den Schirm und geht nach 2040 zu einem erneuten "input" zurück. Dadurch wird der Schüler gezwungen, die richtige Antwort abzuschreiben. Eben diese Abschrift der nichtgewußten Antwort ist eines der Elemente, durch die das Computer-gesteuerte Vokabeltraining so effektiv ist!! Beachte, daß in 2100 die Variable ri gleich 2 gesetzt wird. Auf diese Art markieren wir, daß eine Frage fehlerhaft behandelt wurde und können später entsprechend unsere Punktwertung gestalten.

Von 10360 aus springen wir gleich in ein weiteres Unterprogramm, wo die Punkte gezählt werden:

```
10360 gosub 3000: rem **** "Punktewertung" *****
```

und

```
3000 rem ***** "Punkteauswertung" *****
3010 if ri=2 then 3030
3020 v=v+1
3030 l=int(v/f*100)
3040 x=0:y=1:gosub 20
3050 ? "   Fragen      o.k.      falsch      o.k. (%)"
3060 y=3:gosub 90:gosub 20
3070 ? tab(4) f tab(14) v tab(24) f-v tab(34) l
3080 ri=0
3100 return
```

Wir geben unseren Schülern die Möglichkeit, sich während der Arbeit mit einem Seitenblick über den Stand der Prüfung zu informieren!
In 3010 wird Zeile 3020 übersprungen, wenn ri=2, die Frage also nicht korrekt beantwortet wurde. In 3020 wird in v die Anzahl der richtigen Antworten zwischengespeichert. Aus v und f (erinnere Dich: f = die Anzahl der gestellten Fragen!) wird ein Bruch gebildet, der den Prozentsatz der richtigen Antworten anzeigt.
In 3040/3050 projiziert der Rechner die Überschrift für die aktuellen Prüfungswerte in die erste Bildschirmzeile. 3060 löscht danach die dritte Zeile, bevor im Verein mit 3070 die Werte f, v, f-v (die Anzahl der falschen Antworten) und l eingeblendet werden. Beachte, daß 3080 den Merker ri, der anzeigte, ob die letzte Antwort richtig oder falsch war, wieder auf Null setzt!! Aus 3100 geht der Rechner ins Hauptprogramm nach 10370 zurück:

```
10370 if f<r then 10310
```

wo geprüft wird, ob f -die Anzahl der schon gestellten Fragen noch kleiner ist als die Anzahl der eingespeicherten Vokabelpaare. Solange dies der Fall ist und noch Vokabeln in unserem Pool vorhanden sind, die nicht gefragt wurden, geht der Rechner nach 10310 zurück, zieht die nächste noch freie Zufallszahl, speichert die zugehörigen Vokabeln in fr$ und ub$, geht ins Frage-Antwort-Unterprogramm usw.
Wenn die Bedingung nicht mehr erfüllt ist, f also gleich r ist und alle Vokabeln geprüft wurden, geht der Rechner

weiter nach 10380:

```
10380 for i=1 to r:a(i)=0:next
10711 f=0
```

und setzt die Feldvariable a() und die Zähler f und v auf Null, damit wir auf Wunsch einen neuen Prüfungsdurchgang starten können.

****

Zum Abschluß des Kapitels eine wichtige Kleinigkeit. Wir wissen, daß wir vor allem bei längerer Arbeit vor dem Schirm unsere Augen so gut wie eben möglich schützen müssen. Das heißt, daß der Bildschirm möglichst leer und lichtschwach sein sollte. Wir schreiben daher:

```
10002 poke 53280,0:poke 53281,0:poke 646,1
```

und erhalten weiße Schrift auf schwarzem Grund. Das 53280er-poke ist dabei zuständig für die Rahmenfarbe des Schirms, das 53281er-poke für den Bereich, in den die Programminhalte projiziert werden. Das 646er-poke bestimmt die Farbe der Schrift.

Wir können auch andere Bildschirmfarben wählen, wenn wir in die Speicherstellen 53280 und 53281 andere Werte als Null hineinpoken. Nachstehend geben wir die Liste der Farben und der zugehörigen Farbnummern:

0	schwarz
1	weiß
2	rot
3	cyan
4	rosa
5	grün
6	blau
7	gelb
8	orange
9	braun
10	hellrot

11 dunkelgrau
12 mittelgrau
13 hellgrün
14 hellblau
15 hellgrau

Die gleichen Zahlen in die Speicherstelle 646 hineingepokt ergeben eine entsprechende Schriftfarbe.

Wenngleich unser Programm nun mit jedem Kapitel leistungsfähiger und unseren Schülern die Arbeit immer komfortabler wird, bleiben doch noch einige entscheidende Verbesserungen zu bewältigen. Im nächsten Kapitel werden durch eine Schlußwiederholung der Fehler die Schwachstellen der Schüler noch besser eingeübt.

Wir haben gelernt:

- wie Unterprogramme für die Prüfungssequenz und die Punktewertung geschrieben werden.
- wie die Bildschirmfarbe und die Schriftfarbe geändert werden können.

13. Kapitel

Wir lernen in diesem Kapitel, wie Informationen über Fehler gespeichert und zu einer Schlußwiederholung verarbeitet werden. Jede fehlerhaft beantwortete Frage wird den Schülern noch einmal vorgelegt. Wir bieten den Schülern darüberhinaus zur Beantwortung der Fragen einen zweiten Versuch an. Schließlich lernen wir die Funktion len() kennen, durch die wir die Anzahl der Buchstaben in einem Wort erfragen können und schreiben ein Unterprogramm, das Wörter genau in die Bildschirmmitte projiziert.

Der Rechner soll sich in Zukunft merken, bei welchen Wörtern die Schüler Fehler machten, um sie ihnen am Ende der Prüfung in einer Schlußwiederholung ein letztes Mal zur Beantwortung vorzulegen. Um dies zu erreichen, muß der Rechner sich die Vokabelnummern (in unserem Programm als i vom Zufallsgenerator gezogen) merken können. Diese Aufgabe wird gelöst, indem durch einen Fehler die Vokabelnummer in einem Element einer Feldvariablen zwischengespeichert wird. Wir ändern 10001 um in :

```
10001 dim a(20),b(100)
```

und dimensionieren somit 100 verschiedene b()'s. In unser Frage-Antwort-Unterprogramm müssen wir nun jedesmal, wenn ein Fehler gemacht wurde, die Vokabelnummer in das nächst höhere b() speichern:

```
2205 if ri=2 then fe=fe+1:b(fe)=i
```

Wenn ri=2 ist (die Antwort war falsch), wird die Variable fe (FEhler) um eins hochgesetzt. Beim ersten Fehler wird i, also die zur Vokabel gehörende Nummer, in b(1) gespeichert, beim zweiten Fehler in b(2), beim dritten in b(3) etc. Beachte, daß wir für jeden weiteren Prüfungsdurchgang fe wieder auf Null setzen müssen. Wir streichen 10390 und schreiben:

```
10710 f=0:v=0:fe=0
```

Sobald die Prüfung beendet ist, teilen wir dies dem Schüler mit, indem wir den Bildschirm löschen und in die Mitte desselben "Ende der Prüfung" einblenden. Um die Mitteilung genau in die Bildschirmmitte zu bekommen, bedienen wir uns eines Tricks, den wir in ein Unterprogramm schreiben:

```
10385 gosub 10
10390 t$="Ende der Prüfung":y=11:gosub 100
```

und

```
100 rem ******  "Zentrierung"  ******
101 le=len(t$)
102 x=(38-le)/2
103 gosub 20:? t$
104 return
```

Die Neuheit des letzten Programmabschnittes ist Zeile 101: "le=len(t$)". t$ ist bei Ankunft im Unterprogramm 100ff "Ende der Prüfung" und besteht aus 16 Buchstaben. Dies herauszufinden ist die Aufgabe von len(). In 101 wird in le der Zahlenwert 16 abgelegt. Eine Programmzeile darauf, in 102, wird von der leicht korrigierten Spaltenzahl unseres Bildschirm dieser Wert le abgezogen. 38 minus 16 ergibt dann die Zahl der freien Spalten in einer Bildschirmzeile, in der bereits "Ende der Prüfung" steht. Teilen wir diese Zahl, hier 22, durch 2, wie in 102 geschehen, so erhalten wir die Anzahl der Spalten, die vor unserer Mitteilung frei zu bleiben hat, damit diese genau zentriert auf dem Bildschirm erscheint. Von 10390 war y als 11 definiert; in 102 wird x als 8 definiert. Mit diesen beiden Werten x und y geht der Rechner von 103 ins Lokalisations-Unterprogramm nach 20, betet die Zauberformel und blendet t$ in die gewünschte Position, sobald er von 20 zurück kommt. RETURN in 104 bringt ihn wieder zurück hinter 10390. Dort steht eine Warteschleife und schließlich:

```
10400 for i=1 to 2000:next
10410 gosub 4000:rem **** "Schlusswiederholung" ***
```

Bei 4000 ff folgt:

```
4000 rem ***** "Schlusswiederholung" *****
4001 if fe=0 then return
4010 gosub 10
4020 y=2:t$="Schlusswiederholung":gosub 100
4025 g=1:h=fe
4030 for k=g to h
4040 i=b(k)
4050 fr$=b$(i):ub$=a$(i)
4060 gosub 2000
4065 ri=0
4070 next
4080 if fe=h then return
4090 g=h+1
4100 h=fe
4110 goto 4030
```

Unverständlich, nicht wahr?
Zur Erläuterung ein kleines Übungsbeispiel. Wenn wir schreiben:

```
10 for i=1 to 5
20 ? i
30 next
```

so werden die Zahlen 1 bis 5 untereinander auf den Schirm geschrieben. Wir können die beiden Werte 1 und 5 aus Zeile 10 auch als Variablen schreiben, die wir vorher definiert haben:

```
5 a=1:b=5
10 for i=a to b
20 ? i
30 next
```

Schreiben wir die Zeilen 10 bis 30 in ein Unterprogramm, so

könnten wir sagen:

```
1 ? chr$(147):goto 100
10 for i=a to b
20 ? i
30 next
40 return
100 a=1:b=5:gosub 10
110 a=6:b=10:gosub 10
120 a=10:b=15:gosub 10
130 a=16:b=20:gosub 10
```

und auf dem Schirm erscheinen untereinander die Zahlen 1 bis 20. Die 10/30-Schleife wurde insgesamt viermal durchlaufen, jedesmal jedoch mit anderen Werten a und b, die zuvor jeweils in 100 bis 130 definiert worden waren.

Kehren wir zu unserem Programm zurück. In 4010 wird der Bildschirm gelöscht und in 4020 "Schlußwiederholung" zentriert in die zweite Bildschirmzeile gebracht. In 4025 werden die Werte für die for-next-Schleife aus 4030 definiert. Dort soll k von 1 bis fe, entsprechend der Anzahl der gemachten Fehler und somit der beschrifteten b()'s, hochgezählt werden. Der Wert 1 wird in g, der Wert von fe in h abgelegt: "for k=g to h". Von b(g) bis b(h) (im Klartext: von b(1) bis b(fe)!) werden in 4040 die Vokabelnummern ausgelesen und in i abgelegt. In 4050 schließlich werden anhand von i die Wörter a$(i) und b$(i) bestimmt, die in fr$ und ub$, unseren beiden Stringvariabeln für Frage und Antwort, für das Unterprogramm 2000ff zwischengespeichert werden. 4060 geht in das Unterprogramm 2000, und ohne eine weitere Programmzeile zu schreiben, läuft die bekannte Frage-Antwort-Sequenz ab.
Nachdem das Unterprogramm so oft angesteuert wurde, wie vorher Fehler gemacht wurden, also fe-mal, geht der Rechner nach 4080 und prüft, ob fe gleich h ist. fe ist dann gleich h, wenn in der Schlußwiederholung kein weiterer Fehler gemacht worden ist, und der Rechner würde ins Hauptprogramm zurückkehren. Wenn fe aber nicht gleich h ist und demnach neue Fehler am Ende der Fehlerliste b() vermerkt wurden,

werden die Werte g und h der 4030/4070-Schleife in 4090 und 4100 neu definiert. g wird um 1 höher angesetzt als das letzte h, während in h der Wert von fe abgelegt wird. Die Schleife 4030/4070 wird (h-g)-mal durchlaufen, und nach return und somit ins Hauptprogramm gelangt der Rechner erst, wenn schließlich und kein Fehler mehr gemacht wurde und in Zeile 4080 fe=h ist.

Fassen wir zusammen:
die Fehler, die unsere Schüler während der Prüfung machen, werden in einer Feldvariablen b() als Vokabelnummer gespeichert. Nach dem Prüfungsende werden alle Fehler noch einmal zur Beantwortung vorgelegt. Auch die Fehler dieser Schlußwiederholung werden notiert und der Fehlerpool entsprechend erweitert. Der Rechner geht erst dann zur abschließenden Bewertung ins Hauptprogramm zurück, wenn keine Fehler mehr gemacht werden!!
Durch diese Art der Prüfungsanordnung erreichen wir bereits ein Höchstmaß an Wirksamkeit, durch das selbst schwierige Vokabeln in kürzester Zeit erlernt werden. Dennoch haben wir unser Drill-Repertoire nicht erschöpft, wie Kapitel 17 zeigen soll...

Wir haben gelernt:

- daß mit der Funktion "len" die Anzahl der Buchstaben in einer Stringvariablen bestimmt wird.
- die Kennziffern der Vokabeln im Falle einer falschen Antwort in der Feldvariablen b() zwischenzuspeichern.
- in einem Unterprogramm diese Kennziffern aus b() wieder auszulesen und die entsprechenden Vokabeln noch einmal zur Beantwortung vorzulegen.
- eine for-next-Schleife mit den Variabeln g und h zu konstruieren und sie mit verschiedenen, jeweils neu definierten Werten mehrmals zu benutzen.

14. Kapitel

Wir lernen in diesem Kapitel ein weiteres Beispiel für eine mit Variablen gesteuerte for-next-Schleife kennen und können dadurch mehr Vokabeln als bisher in unsere Kapitel einspeichern. Schließlich geben wir unseren Schülern einen zweiten Versuch bei der Beantwortung der Fragen. Erst bei zweimaliger falscher Antwort soll die richtige Lösung zur Abschrift eingespielt werden.

Mit der nachfolgenden Programmänderung wollen wir erreichen, daß die eingespeicherten Vokabeln gedrittelt werden (bei 21 eingeladenen Wortpaaren entstünden so 3 Drittel zu jeweils 7 Wortpaaren) und jedes Drittel bei der anfänglichen Wiederholung getrennt eingeblendet wird. Wir wollen zwischen den Dritteln vor- und zurückblättern können und in die nullte Bildschirmzeile soll die gerade eingeblendete Seite vermerkt sein.

Wir streichen die Zeilen 10220 bis 10280 und schreiben stattdessen:

```
10220 rem **** "Vokabeleinsicht" *****
10230 gosub 1000
```

und

```
1000 rem ***** "Unterprogramm Vokabeleinsicht" ***
1001 gosub 10
1010 s=1
1020 r3=int(r/3)
1030 if s=1 then g=1:h=r3
1040 if s=2 then g=r3+1:h=r3*2
1050 if s=3 then g=r3*2+1:h=r3*3
1060 for y=2 to 21:gosub 90:next
1070 x=30:y=0:gosub 20:? "Seite ";s
1080 ?
1090 for i=g to h
1100 ? a$(i) tab(20) b$(i)
1110 next
```

```
1120 t$="<P>rüfung, <n>ächste, <l>etzte Seite"
1130 y=22:gosub 100
1140 get a$
1150 if a$="p" then return
1160 if a$="n" then s=s+1:goto 1190
1170 if a$="l" then s=s-1:goto 1190
1180 goto 1140
1190 if s=4 then s=1
1200 if s=0 then s=3
1210 goto 1030
```

In 1020 wird der Wert r3 aus dem ganzzahligen Drittel von r, der Anzahl der eingespeicherten Vokabeln, gebildet. In 1030 bis 1050 werden für die drei möglichen Werte, die s annehmen kann (siehe unten), jeweils unterschiedliche Werte von g und h definiert, die angeben, welche Vokabeln auf den Schirm geholt werden. Bei einer Gesamtvokabelzahl von r=21 wird r3=7. Die Vokabeln sind damit folgendermaßen auf die drei Drittel zu verteilen:

Drittel:					
I.		II.		III.	
1 bis 7		8 bis 14		15 bis 21	

Daraus ergeben sich folgende Werte für g und h:

1 bis r3		r3+1	r3*2	r3*2+1	r3*3
g	h	g	h	g	h

In 1070 wird am Koordinatenpunkt 30/0 vermerkt, daß der Schüler sich auf Seite s befindet. Nach einer Leerzeile werden durch die 1090/1110-Schleife die Vokabeln g bis h in gewohnter Art auf den Bildschirm projiziert. In 1120/1130 erscheint in der 22. Bildschirmzeile die Anweisung an die Schüler, wie sie das Programm fortsetzen können. Durch "n"

soll er zur nächsten Vokabelseite, durch "l" zur letzten Seite, durch "p" zur Prüfung gelangen. Der Programmabschnitt 1140/1180 wird solange durchlaufen, bis eine dieser drei Tasten betätigt wird. "p" veranlaßt ein return und damit den Sprung zurück ins Hauptprogram. Bei "n" wird s um 1 erhöht, bei "l" um 1 erniedrigt und der Rechner geht weiter nach 1190. Dort und in der folgenden Zeile wird der Wert s noch einmal daraufhin geprüft, ob er zulässige Werte enthält. Für s sind nur die Werte 1,2 und 3 zulässig. Wer, obwohl er sich schon auf der dritten Vokabelseite befindet, dennoch mit "n" auf die nächste Seite will und in 1160 s auf 4 erhöht hat, bekommt für s den Wert 1 festgelegt, und die erste Seite erscheint. Ebenso darf von der ersten Seite aus nicht wirklich zurückgeblättert werden; mit dem Wert s=0 könnte der Rechner nicht arbeiten. Wenn s=0, dann s=3; wer von der ersten Seite aus zurückblättert, findet sich auf Seite 3 wieder.

Die Anzahl der Vokabeln, die wir durch unsere letzte Änderung den Schülern in einer Lektion vorstellen, sollte dennoch nicht überzogene Werte annehmen. Es ist völlig ausreichend, wenn wir die Lektionen auf 50, oder besser: 48 Vokabeln (weil durch 3 teilbar) begrenzen. Die sich dadurch nötigen Änderungen des Hauptprogramms betreffen die Dimensionierungen der Feldvariablen. Es müssen geändert werden:

```
10001 dim a(49),b(100)
10050 dim a$(49),b$(49)
```

Im Generatorprogramm ebenso:

```
10 dim a$(49),b$(49),c$(49),d$(49)
20 for i=1 to 49
240 for i=1 to 49
```

Beachte hier, daß die Dimensionierung den 49. Platz für die "r"-"r"-Weiche vorsieht.

****

Es geschieht immer wieder, daß Fehler nicht durch Unwissenheit, sondern durch Nachlässigkeit beim Maschinenschreiben entstehen. Dies gilt sowohl für den Anfänger, der mit der Tastatur noch wenig vertraut ist, wie für den Fortgeschrittenen, der so schnell über die Tasten huscht, daß ihm der eine oder andere Buchstabe entwischt. Es ist ärgerlich, wenn für solch verzeihliche Unregelmäßigkeiten vom Rechner ein Fehlerpunkt in Rechnung gestellt wird, und daher nur fair, wenn wir den Schülern die Möglichkeit eines zweiten Versuchs geben. Wir ändern im Frage-Antwort-Unterprogramm:

```
2010 q=0

2065 if q=1 then 2070
2066 q=1
2067 gosub 90
2068 x=6:gosub 20:? "??":goto 2040
```

In 2010 wird zu Beginn jeder neuen Frage der Wert q auf Null gesetzt, so daß bei einer falschen Antwort im ersten Versuch in 2065 die Bedingung nicht erfüllt ist. Sofort danach wird q als 1 definiert, die Antwortzeile 13 (noch als y aus 2040 definiert) in 2067 nebst 90 gelöscht und der Rechner zurück nach 2040 geführt. Erst wenn auch beim zweiten Anlauf die Frage falsch beantwortet wurde, der Rechner sich nicht direkt nach 2200 retten konnte und wieder bei 2065 angelangt, wird er nach 2070ff weitergeleitet, wo die richtige Lösung zur Abschrift eingeblendet wird.

Wir haben gelernt:

- die Vokabeln unserer Kapitel zu dritteln und bei der Vokabelwiederholung von einem Drittel zum nächsten weiterzublättern.
- unseren Schülern einen zweiten Versuch zu gewähren.

15. Kapitel

Wir bemühen uns in diesem Kapitel darum, die Darstellung des Endergebnisses für die Schüler ansprechender zu gestalten. Dazu errechnen wir eine Punktzahl, die abhängig ist von der benötigten Prüfungszeit und dem Prozentsatz der richtigen Antworten.

Wir streichen die Programmzeilen 10600 bis 10711 und schreiben stattdessen:

```
11000 rem **** "Auswertung des Endergebnisses" ***
11005 l=int(v/r*100)
11010 z=ti/60
11011 z$=ti$
11020 z1$=mid$(z$,1,2)
11030 z2$=mid$(z$,3,2)
11040 z3$=mid$(z$,5,2)
11050 z$=z1$+":"+z2$+":"+z3$
```

Wir kennen die Rechneruhr, die in ti$ die Zeit speichert, die seit dem Einschalten des Rechners oder seit der letzten Nullsetzung durch ti$="000000" vergangen ist. Eine zweite Zeitmessung erfolgt gleichzeitig durch die Rechnervariable **ti**, die den Wert von ti$ in 60stel Sekunden ausdrückt. Durch Dividieren des Wertes aus ti mit 60 erhalten wir damit die Anzahl der Sekunden, die in ti$ gespeichert ist. In 11010 wird die Prüfungszeit in Sekunden in z zwischengespeichert. In 11011 wird die Prüfungszeit aus ti$ in z$ abgelegt. Was dann in 11020 bis 11050 folgt, ist nur eine optische Spielerei für unsere Schüler, durch die wir aber eine sehr wichtige Funktion des Basic kennenlernen: **mid$(..$,x,y)!!**
Die Funktion mid$ trennt aus einem beliebigem String einzelne hintereinander geschriebene Buchstaben heraus. Die erste Zahl (hier: x) gibt an, beim wievielten Buchstaben des Wortes mit dem Ausschnitt begonnen wird, während die zweite Zahl (y) angibt, wieviel Buchstaben -ausgehend von x- herausgeschnitten werden. Ein Beispiel soll dies verdeutlichen:

Wenn wir einen String t$ haben, der "Mollis und Müslis" heißt, so hat t$ eine Länge von 17 Buchstaben (wie wir auch durch "? len(t$)" erfahren könnten!). Wenn wir aus t$ das Wort "Mollis" heraustrennen wollen, müssen wir die mid$-Funktion folgendermaßen schreiben:

```
? mid$(t$,1,6)
```

oder

```
z1$=mid$(t$,1,6)
? z1$
```

wodurch der Ausschnitt in z1$ zwischengespeichert wäre. Die erste Zahl (1) gibt an, daß der Anfang des Ausschnittes beim ersten Buchstaben des Strings t$ zu suchen ist, also beim "M" von "Mollis"; die zweite Zahl (6) gibt die Gesamtlänge des neuen Strings z1$ an. Ähnlich können wir aus t$ auch "Müslis" aussondern und zwar durch:

```
z1$=mid$(t$,12,6)
```

weil hier der erste Buchstabe des auszusondernden Teils der 12. Buchstabe von t$ ist. "und" würde mit:

```
z1$=mid$(t$,8,3)
```

herausgeschnitten.

Zurück zu unserem Programm. Wir sagten schon, daß die jetzt folgende Erweiterung nur eine Spielerei am Rande ist, die das Endergebnis gefälliger machen soll. Wir erinnern uns, daß die Rechnerzeit aus ti$ in der Form "000000" ausgedruckt wird. 5 Minuten und 45 Sekunden werden als "000545" ausgegeben. Es ist dies eine unansehnliche Art der Darstellung, die wir in 11020 bis 11050 korrigieren. In 11020 werden die ersten beiden Buchstaben des Strings z$ in z1$ abgelegt. Bei einem z$ von beispielsweise "000545" wird in z1$ "00" gespeichert. In 11030 werden die Buchstaben Nummer 3 und 4 von z$ -mid$(t$,3,2)- in z2$ abgelegt, z2$ wird "05". Schließlich werden in 11040 die letzten

beiden Buchstaben Nummer 5 und 6 in z3$ zwischengelagert: "45".
Wir haben einen String ("000545") in drei Einzelstrings zerlegt ("00"; "05"; "45") und brauchen diese drei Einzelstrings nur noch neu zusammenzusetzen, um unserem nicht immer grandiosen Commodore eine stilvolle Zeitansage zu ermöglichen. In 11060 setzen wir die Strings z1$, z2$ und z3$ zu einem neuen z$ zusammen, in dem die Einzelstrings durch Doppelpunkte getrennt sind. Beachte hier noch einmal, daß mehrere Strings durch das Pluszeichen zu einem größeren vereinigt werden!

Wir gehen weiter in der Verkündung des amtlichen Endergebnisses:

```
11100 gosub 10
11110 y=2:x=0:gosub 20:? "Benötigte Zeit:"
11120 x=19:gosub 20:? "Prozentsatz richtiger"
11130 y=3:x=25:gosub 20:? "Antworten:"
11140 y=5:x=3:gosub 20:? z$
11150 x=28:gosub 20:? l
11160 f=0:v=0:fe=0:for i=1 to r:a(i)=0:next
```

Wie Strings oder Variablen auf bestimmte Punkte des Bildschirms projiziert werden, haben wir ausführlich in den Kapiteln 11 und 12 besprochen. Die gerade programmierte Passage bietet daher keine Schwierigkeiten.

Wir gehen weiter und errechnen für unsere Schüler eine Punktzahl, die abhängig ist von der Prüfungszeit und dem Prozentsatz der richtigen Antworten. Nachdem wir wissen, daß der Schüler z Sekunden (vgl. 11010) für die gesamte Prüfung benötigt hat, können wir die Zeit zz errechnen, die für eine einzelne Frage gebraucht wurde:

```
11200 zz=z/r
```

Nun legen wir eine Zeit in Sekunden fest, die wir unseren Schülern maximal für die Beantwortung einer Frage zugestehen:

```
11210 t=20
```

und errechnen aus der Differenz zwischen zz und t einen neuen Wert p, der -multipliziert mit l, dem Prozentsatz der richtigen Antworten- die Punktzahl pz ergibt:

```
11220 p=t-zz
11230 pz=p*l
```

Je kleiner z (Gesamtprüfungszeit in Sekunden) ist, so kleiner ist zz (Prüfungszeit pro Frage), um so größer ist p, die Differenz zwischen der festgelegten Höchstzeit und der Prüfungszeit pro Frage. Je höher p ist, umso höher ist pz. Die Punktzahl pz ist umso höher, je schneller die Schüler die Fragen beantworten und je höher der Prozentsatz der richtigen Antworten l ist.

Ein Beispiel verdeutlicht diesen Rechengang. Nehmen wir an, die Prüfung bestand aus 48 Fragen (r=48), ein Schüler benötigte 5 Minuten, also 300 Sekunden (z=300), und erreichte 80% richtige Antworten. zz wird 300/48, etwas mehr als 6 Sekunden. Bei einem maximalen Wert von 20 Sekunden pro Frage (t=20) erhalten wir einen Differenzwert von 14 (p=14). In 11230 erhalten wir so einen Wert pz von 14*80: 1120, die endgültige Punktzahl. Es liegt in unserem Ermessen, ob wir diesen letzten Wert durch eine Konstante noch vergrößern oder verkleinern wollen. Wenn wir den Schülern Punktzahlen in den Fünftausendern bescheren möchten, wie sie es von Videospielen gewöhnt sind, können wir pz noch einmal mit 4 multiplizieren:

```
11240 pz=pz*4
```

Diesen Wert brauchen wir nun nur noch in die Bildmitte als nicht zu übersehende Erfolgsmeldung zu projizieren:

```
11250 y=11:t$="Deine Punktzahl:":gosub 100
11260 y=13:x=16:gosub 20:? int(pz)
11299 fe=0:f=0:v=0
```

und die Schüler werden uns danken für die besondere Pflege, die wir ihnen angedeihen lassen.

Wir blenden in Zeile 23 die Anweisung ein, daß der Schüler mit beliebiger Taste im Programm fortfahren kann:

```
11300 t$="Fortsetzen mit beliebiger Taste"
11310 y=23:gosub 100
11320 get a$:if a$="" then 11320
```

Wir haben gelernt:

- den "Uhr"-string ti$ mit der Funktion "mid$" in seine Einzelteile zu zerlegen und eine das Auge befriedigende Zeitansage zu ermöglichen.
- eine Punktzahl zu errechnen, für die die Prüfungszeit und der Prozentsatz der richtigen Antworten die Bewertungsgrundlage darstellen.

16. Kapitel

Wir bemühen uns in diesem Kapitel, einer Zahl das Laufen zu lehren und lernen dabei eine for-next-Schleife mit dem Zusatz "step" kennen. Wir speichern die Ergebnisse in eine Feldvariable, so daß die Punktzahlen aller Prüfungen des laufenden Tages im Anschluß an das Prüfungsergebnis eingeblendet werden. Wir lernen schließlich, den Mittelwert aller Prüfungspunkte zu ermitteln.

Wenn nach abgeschlossener Prüfung und Schlußwiederholung die Schüler sich zufrieden nach hinten lehnen und auf die Bekanntgabe des Endergebnisses warten, sind wir ihnen zweifellos kleine Sonderleistungen schuldig. Die Zeitangabe haben wir schon im letzten Kapitel eleganter gestaltet als durch das Commodore-Basic vorgegeben. Des weiteren wäre nun zu überlegen, ob wir den Augenblick, in dem die Punktzahl bekanntgegeben wird, nicht spannender gestalten. Wir könnten den Schülern das Ergebnis "scheibchenweise", als von 1 bis zur Endzahl laufende Ziffer präsentieren.
Wir schreiben zur einfacheren Demonstration ein kleines Übungsprogramm:

```
10 ? chr$(147)
20 for i=0 to 1000
30 poke 211,16:poke 214,11:sys 58640
40 ? i
50 next
```

Auf dem Bildschirm erscheinen bei Koordinate 16/11 nacheinander die Zahlen 0 bis 1000. Um die Aufeinanderfolge der einzelnen Zahlen zu verlangsamen und für das Auge nachvollziehbar zu machen, fügen wir nun noch ein:

```
45 for k=1 to 100:next
```

Wenn wir hier angelangt sind, können wir eine wesentliche Neuerung in unser Programm einfügen. Wir ersetzen 20 und schreiben:

```
20 for i=0 to 1000 step 5
```

Wenn wir das Programm wieder starten, sehen wir, daß nicht alle Zahlen von 0 bis 1000 eingeblendet werden, sondern zwischen den einzelnen Zahlen eine Differenz von 5 besteht. Der Rechner setzt beim Durchgang durch Zeile 20 i nicht um 1 hoch, sondern um 5, und es erscheinen die Zahlen 0,5,10,15,20,25,30 etc.

Der Wert nach step kann auch ein Minuswert sein, so daß die Zahlen rückwärts gezählt werden. Wir schreiben in 20:

```
20 for i=1000 to 0 step -5
```

fügen 46 ein, damit die Zeile 11 nach jedem Durchgang wieder gelöscht wird:

```
46 poke 781,11:sys 59903
```

und es erscheinen auf dem Schirm die Zahlen 1000,995,990,985,980,975 etc.

Die Zahl hinter "step", die angibt, um wieviel der Rechner die Schleifenvariable (in unserem Beispiel i) hoch- bzw. herunterzählt, kann auch eine Variable sein:

```
1 a=5
20 for i=1000 to 0 step -a
```

und wir erhalten den gleichen Programmablauf wie zuvor.

Nach diesen Vorbemerkungen gehen wir zu unserem Trainingsprogramm zurück. Wir streichen Zeile 11260 und schreiben statt dessen:

```
11260 x=13:y=16
11270 for i=0 to pz
11280 gosub 20:? int(i)
11290 next
```

Ein einzelnes Detail ist nun noch störend. Wer wenig Punkte macht, bekommt sein Ergebnis in wenigen Sekunden präsentiert, während diejenigen, die hohe Punktzahlen erhalten, unter Umständen ewig warten. Wir müssen deshalb einen Weg finden, damit die Zahl unabhängig vom Ergebnis unserer Schüler immer die gleiche Zeitspanne benötigt, um zu ihrem Endwert zu laufen. Dies erreichen wir durch unseren neuen "step"-Befehl. Wir schreiben in 11270:

```
11270 for i=0 to pz step pz/100
```

Zwei Beispiele sollen uns die Wirkungsweise der letzten Änderung erklären. Nehmen wir an, wir haben zwei Schüler, von denen der erste 5000 Punkte erzielt, der zweite hingegen nur 100. Im ersten Fall zählt der Rechner von 0 bis 5000 in Schritten von 5000/100, also in 50er-Schritten. Um in 50er-Schritten von 0 bis 5000 zu zählen, müssen 100 Zahlen auf den Schirm geschrieben werden.
Beim zweiten, schlechteren Schüler muß der Rechner von 0 bis 100 zählen, in Schritten von pz/100, also 100/100, das heißt in Einer-Schritten. Um von 0 bis 100 in Einer-Schritten zu zählen, muß der Rechner 100 Zahlen auf den Schirm bringen, genau die gleiche Anzahl wie im ersten Beispiel. Wir sehen, daß unabhängig vom Ergebnis des Schülers immer die gleiche Anzahl von Zwischenwerten eingeblendet wird und nur die Zuwachsraten von Zwischenwert zu Zwischenwert verschieden sind.
Ein kleines Detail noch zum Schluß. Wenn der Rechner in 30er-Schritten von 0 bis 1000 zählen soll, wird die letzte ausgegebene Zahl 990 sein, weil die nächsthöhere Zahl -1020- bereits außerhalb des in der for-next-Schleife definierten Höchstwertes liegt. Um zu verhindern, daß der Rechner den Schüler um 10 Punkte betrügt, fügen wir im Anschluß an die Schleife noch ein:

```
11291 gosub 20:? int(pz)
```

Eine letzte Änderung noch, damit auch negative Zahlen laufen können:

Wir definieren in 11260 zusätzlich den step-Wert als Variable sp:

```
11260 x=13:y=16:sp=pz/100
```

und fügen sp in 11270 ein:

```
11270 for i=0 to pz step sp
```

Ein negativer Wert von pz führt zu in 11260 zu einem negativen Wert von sp und die Schleife aus 11270 funktioniert tadellos.

****

Der nächste Service, den wir unseren Schülern vorbereiten, betrifft die Speicherung sämtlicher Tagesergebnisse und deren Bekanntgabe im Anschluß an das letzte Prüfungsergebnis. Wir speichern die Punktzahlen in der Feldvariablen er (ERgebnis), die wir in 10001 zusätzlich auf 20 Einheiten definieren:

```
10001 dim a(50),b(100),er(20)
```

und fahren fort mit:

```
12000 rem **** "Gesamtergebnis" *****
12010 pr=pr+1
12020 er(pr)=int(pz)
12030 gosub 10
12040 x=10
12050 for i=1 to pr
12060 y=i
12070 gosub 20
12080 ? i;".Prüfung: ";er(i)
12090 next
```

In 12010 wird der Wert pr hochgezählt und gibt an, die wievielte Prüfung seit Einschalten des Rechners abgelegt wurde. In der nächsten Programmzeile wird durch pr das Feld

von er definiert, in dem die gerade erreichte Punktzahl pz abgelegt wird. Der Bildschirm wird gelöscht, x wird als 10 für die Spaltenpositionierung definiert, die 12050/12090-Schleife wird pr-mal durchlaufen und gibt die in er() zwischengespeicherten Werte aus. Der y-Wert für Zeile 20 wird in 12060 festgelegt als der aktuelle Schleifenwert i, so daß mehrere Zeilen untereinander geschrieben werden. Beachte Zeile 12080: Der Rechner druckt nacheinander in die gleiche Bildschirmzeile:
1. den Schleifenwert i
2. den Zusatz: ".Prüfung: "
3. den zu i gehörigen Wert aus er()

Wir gehen noch weiter in unserem Service und zählen die Punkte aus allen Prüfungen zusammen, teilen diese Summe durch die Anzahl der Prüfungen und erhalten auf diese Weise den Mittelwert aller Tagesergebnisse:

```
12100 for i=1 to pr
12110 ge=ge+er(i)
12120 next
12125 ge=int(ge/pr)
12130 t$="Mittelwert aller Prüfungen: "
12140 y=21:gosub 100
12150 y=23:t$=str$(ge):gosub 100
12160 ge=0
12170 t$="Fortsetzung mit beliebiger Taste":y=0:gosub 100
12180 get a$:if a$="" then 12180
```

Die 12100/12120-Schleife zählt alle Ergebnisse aus der Feldvariablen zu dem GEsamtwert ge zusammen. In 12130/12140 wird auf schon bekannte Weise die Mitteilung über den Mittelwert aller Prüfungen in Zeile 21 zentriert.
Beachte in 12150 eine bisher noch unbekannte Funktion: **str$()**. In 12150 soll der Mittelwert unabhängig von der Anzahl der Ziffern in die Bildschirmmitte plaziert werden. Da in 100ff jedoch t$ ausgedruckt wird, können wir ge/pr nicht einfach in t$ zwischenspeichern. (Versuche einmal: ge=100:t$=ge) Die numerische Variable ge muß zuerst durch die Funktion str$() in eine Stringvariable umgewandelt

werden. Daher: t$=str$(ge).
Beachte weiterhin 12160!! Die Variable ge muß sofort nach Gebrauch wieder auf Null gesetzt werden, damit sie beim nächsten Prüfungsdurchgang gleich Null ist, um von neuem mit den Tagesergebnissen gefüllt werden zu können!

So schön unserer Programm auch aussieht, so ist es durch die letzten Änderungen aber nicht mehr fehlerfrei. Beachte den Programmabschnitt nach 20000 mit dem Endmenu. Solange wir mit <1> die gleiche Lektion wählen, ist alles o.k. Wenn wir aber auf eine andere Lektion umsteigen wollen, kommen wir in ärgste Schwierigkeiten, weil durch den Befehl "run" alle Variablen auf Null gesetzt werden. Auf Null gesetzt werden deshalb auch die Werte in er(), so daß die Arbeit dieses Kapitels sinnlos geworden wäre. Es ist daher von nun an unmöglich, beim Übergang zu einer neuen Lektion das Programm noch einmal zu starten. Stattdessen müssen wir schreiben:

```
20070 if a$="2" then 10000
```

Auch damit ist uns nicht endgültig geholfen, denn sobald der Rechner von 20070 nach 10000 zurückgeht und nach 10001 gelangt, erscheint die Fehlermeldung:

```
redim'd array error in 10001
```

Der Rechner mag es nicht, daß wir innerhalb eines Programmablaufs die gleiche Feldvariable zweimal definieren. Um zu umgehen, daß dim-Anweisungen dem Rechner zweimal begegnen, müssen wir sie noch vor das Hauptprogramm stellen. Wir löschen daher die Zeilen 10001 und 10050 und schreiben an den Anfang des Programms:

```
1 dim a(49),b(100),er(20)
2 dim a$(49),b$(49)
9 goto 10000
```

Unser Programm ist jetzt wieder fehlerfrei.

Wir haben gelernt:

- eine Zahl vor- und rückwärts auf der Stelle treten zu lassen und haben dabei mit positiven und negativen steps gespielt.
- die Prüfungsergebnisse zu speichern und sie nach jeder einzelnen Prüfung noch einmal im Überblick anzubieten.
- aus den Punktzahlen aller Prüfungen den Mittelwert zu bilden.
- daß Feldvariablen nur einmal dimensioniert werden dürfen.

17. Kapitel

Wir lernen in diesem Kapitel eine besondere Art von Prüfung kennen, durch die den Schülern jede Frage solange gestellt wird, bis sie sie insgesamt zweimal richtig beantwortet haben. Wir teilen den Prüfungsstoff einer Lektion auf in drei gleiche Teile und bieten zum Beginn der Prüfung die Auswahl zwischen der Intensivprüfung in einem der drei Teile und einer einfachen Prüfung über die gesamte Lektion. Die Wirksamkeit unseres Trainingsprogramms wird durch die neuen Änderungen entscheidend erhöht.

Jedes Kapitel hat Wörter, die einfach zu lernen sind und solche, bei denen immer wieder Fehler gemacht werden. Zwar werden alle Fehler in der Schlußwiederholung noch einmal eingeblendet, doch auch hier bereiten besonders ungewöhnliche Wörter oft immer noch Schwierigkeiten. Es ist daher nur naheliegend, einen Prüfungstyp vorzusehen, in dem jedes Wort solange gefragt wird, bis es schließlich auf Anhieb richtig beantwortet wird. Besser noch: wir sehen vor, daß jedes Wort insgesamt zweimal richtig beantwortet werden muß, bevor wir es aus dem Fragenpool herausnehmen.
Bei dieser Art von Intensivprüfung müssen am Ende der Prüfung doppelt soviele richtige Antworten gegeben worden sein wie Vokabeln in der Lektion vorhanden sind. Bei 48 Vokabeln pro Lektion stünde in unserer laufenden Anzeige des Prüfungsstands unterhalb von "richtig" die Zahl 96. Rechnen wir noch eine Reihe von Fragen hinzu, die nicht richtig beantwortet wurden, so würden bei einer 48-Vokabel-Lektion pro Prüfung weit mehr als 100 Fragen gestellt. Die Prüfung wäre zu lang, die Arbeit zu anstrengend und die Schüler würden uns vor Langeweile davonlaufen.

Wir teilen deshalb den Stoff einer Lektion in drei gleiche Teile auf, so daß bei einer Höchstzahl von 48 Vokabeln/Lektion auf jeden Teil maximal 16 Wörter kommen und stellen die Frage, ob eine der drei Teilprüfungen oder die Gesamtprüfung über alle 48 Wörter gewünscht wird. Wenn die Schüler eine der Teilprüfungen wählen, lassen wir im

Programm automatisch die Intensivprüfung einstellen. Wählen sie hingegen die Gesamtprüfung, so wird derjenige Prüfungsmodus eingestellt, den wir bisher kennen, das heißt, jede Frage wird nur einmal gestellt, unabhängig davon, ob sie richtig oder falsch beantwortet wurde.
Nachdem die Vokabeln von Diskette eingeladen worden sind, sollen unsere Schüler die Prüfungsart festlegen. Wir ändern:

```
10090 if a$(i)="r" then 10110
20065 if a$="1" then 10115
```

streichen

```
10210
```

und fügen ein:

```
10110 close2:r=i-1
10115 gosub 10:x=10:for i=1 to 3:y=i+7:gosub 20
10120 ? "<";i;"> ";i;". Teilprüfung":next
10130 y=11:gosub 20:? "<4>  Gesamtprüfung"
10140 get a$
10150 if a$="1" then tp=1:goto 10220
10151 if a$="2" then tp=2:goto 10220
10152 if a$="3" then tp=3:goto 10220
10153 if a$="4" then tp=4:goto 10220
10154 goto 10140
```

Wir lassen in 10110 und 10120 die ersten drei Zeilen aus unserem Zwischenmenu in die Bildschirmzeilen 8 bis 10 (i+7) einblenden.

```
<1>  1. Teilprüfung
<2>  2. Teilprüfung
<3>  3. Teilprüfung
```

und schreiben in 10130 als 4. Option die Gesamtprüfung über alle Vokabeln der Lektion in die 11. Bildschirmzeile. Beachte die gedrängte, durch Doppelpunkte getrennte Schreibweise der einzelnen Befehle in 10110!!

Die Schüler können durch die Zahlen 1 bis 4 eine der vier angebotenen Möglichkeiten wählen. In Abhängigkeit von der Wahl wird die Variable tp definiert.

Ausgehend von diesen Voraussetzungen sind eine Reihe von Programmpassagen umzustellen. Als erstes müssen wir die Zeilen 1000ff bearbeiten, in denen die Vokabeln auf den Schirm geholt werden. Wenn die Schüler die erste Teilprüfung gewählt haben, ist es unsinnig, die Vokabeln der anderen Drittel ebenfalls einzublenden. Wir schreiben die ganze Passage so um, daß bei der Gesamtprüfung alle, bei den Teilprüfungen jedoch nur die entsprechenden Drittel eingeblendet werden. Als Besonderheit lassen wir in der ersten Bildschirmzeile außerdem eine Uhr mitlaufen, durch die den Schülern für die Lernphase maximal 5 Minuten zur Verfügung gestellt werden. Nach Ablauf dieser 5 Minuten geht der Rechner eigenmächtig zur Prüfung weiter. Wir streichen die Zeilen 1010 bis 1210 und schreiben neu:

```
1001 gosub 10:ti$="000000"
1010 s=tp:if s=4 then s=1
1020 r3=int(r/3):r=r3*3
1030 if s=1 then g=1:h=r3
1040 if s=2 then g=r3+1:h=r3*2
1050 if s=3 then g=r3*2+1:h=r
1060 for y=2 to 21:gosub 90:next
1070 x=30:y=0:gosub 20:? "Seite ";s
1080 ?
1090 for i=g to h
1100 ? a$(i) tab(20) b$(i):next
1110 y=23:x=0:gosub 20:? "<P>rüfung ";
1120 if tp=4 then ?"- <B>lättern"
1130 get a$
1140 x=0:y=0:gosub 20:? ti$
1145 if ti$>"000500" then return
1150 if a$="p" then return
1160 if tp=4 then if a$="b" then 1180
1170 goto 1130
1180 s=s+1:if s=4 then s=1
1190 goto 1030
```

Wir setzen die Rechneruhr auf Null und legen in s den Wert tp (Teilprüfung) ab; wenn tp=4 ist (=Gesamtprüfung), ist s=1 und die Vokabeleinsicht beginnt mit dem ersten Drittel.
In 1030 bis 1050 definieren wir g und h für die for-next-Schleife 1090/1100, die die Vokabelliste einblendet. Danach wird in die letzte Bildschirmzeile das Wort "<P>rüfung " geschrieben. Beachte das darauf folgende Semikolon in 1110, das veranlaßt, daß jeder weitere Printbefehl unmittelbar im Anschluß an den letzten noch in die gleiche Zeile ausgegeben wird. Nur wer die Gesamtprüfung gewählt hat, so daß tp=4 ist, bekommt auch die Möglichkeit, von einem Drittel zum nächsten zu blättern und sieht in der letzten Bildschirmzeile

"<P>rüfung - <B>lättern"

In der get-Schleife 1130/1170 fragt der Rechner nun kontinuierlich die Tastatur ab und druckt aus 1140 bei jedem Durchgang die aktuelle Zeit in die obere linke Bildschirmecke. In 1145 prüft er außerdem, ob ti$ schon größer als "000500" geworden ist, ob also die Zeit, die den Schülern für die Vokabelwiederholung zur Verfügung steht, schon die 5-Minuten-Grenze überschritten hat; im positiven Fall bricht er die Wiederholung ab und geht ins Hauptprogramm zurück. Drücken die Schüler auf p, wird die Wiederholung ebenfalls abgebrochen.
Nur wenn tp=4 ist, prüft der Rechner in 1160, ob eventuell b gedrückt wird. Er verläßt die get-Schleife aus 1160 und springt nach 1180.

Beachte 1160:

```
if tp=4 then if a$="b" then 1180
```

Es ist dies die Formel für eine doppelte Bedingung, die in der Rohform heißt:

```
if .x. then if .y. then .z.
```

wobei x die erste Bedingung, y die zweite Bedingung und z

die Konsequenz ist. Nur wenn sowohl Bedingung x wie auch Bedinqung y erfüllt ist, tritt die Konsequenz z in Kraft. Wenn in unserem Fall tp=4 ist (die Schüler hatten zuvor die Gesamtprüfung gewählt) und a$="b" ist (die Schüler haben zum Blättern auf b gedrückt), geht der Rechner nach 1180 und erhöht s um 1. In der gleichen Zeile wird noch festgelegt, daß s=1 ist, wenn s=4 war. Waren die Prüflinge zuvor im 3. Drittel und haben geblättert und damit s um 1 auf 4 erhöht, so wird s=1 und erneut das erste Drittel eingeblendet.
Der Rechner geht zurück nach 1030, definiert erneut die Werte g und h und blendet die nächste Seite ein.

Die Änderungen, die durch die Einfügung der Teilprüfungen notwendig sind, überschreiten die Grenzen eines einzelnen Kapitels. Erst im nächsten Kapitel wird deshalb unser Trainingsprogramm wieder voll funktionsfähig.

Wir haben gelernt:

- einen Prüfungstyp zu konstruieren, durch den eine Frage solange gestellt wird, bis sie insgesamt zweimal richtig beantwortet wurde.
- daß eine doppelte Bedingung durch die Formel "if (Bedingung 1) then if (Bedingung 2) then (Konsequenz)" konstruiert wird.

18. Kapitel

Wir setzen in diesem Kapitel die Arbeit an den Teilprüfungen fort. In die Punktwertung müssen für Teilprüfungen und Gesamtprüfung verschiedene Kriterien bei der Bemessung der Punktzahl eingeführt werden.

Mit der Änderung der Wiederholungspassage sind die Teilprüfungen noch nicht funktionsfähig eingebaut. Jede Frage soll insgesamt zweimal richtig beantwortet werden. Folgende Änderungen sind noch notwendig. Es wird geprüft:

1. Ob ein Vokabelpaar (erkenntlich an der von dem Zufallsgenerator gezogenen Variablen i) noch für die Prüfung frei ist. Wir schreiben 10305 bis 10330

```
10305 kr=0:if tp=2 then kr=r3
10306 if tp=3 then kr=r3*2
10310 if tp=4 then i=int(rnd(1)*r)+1:goto 10320
10311 i=int(rnd(1)*r3)+1+kr
10315 if tp<4 then if a(i)>1 then 10310
10320 if tp=4 then if a(i)=1 then 10310
10330 if tp=4 then a(i)=1
```

sowie

```
2001 an$="*"
```

Wenn eine Teilprüfung absolviert wird (tp ist kleiner als 4!), so ist die Bedingung für eine Rückkehr zum Zufallsgenerator in 10310, daß a(i) größer als 2 ist. Wie wir unter Punkt 2 im nächsten Abschnitt sehen, erhöhen wir bei den Teilprüfungen den Wert a(i) bei jeder korrekten Antwort um 1. Nach der ersten richtigen Antwort wird a(i)=1, nach der zweiten wird a(i)=2. Erst wenn a(i)=2 ist, ist daher die Bedingung von 10315 erfüllt, und der Rechner besorgt sich eine andere Zufallszahl.
Bei tp=4, also bei gewählter Gesamtprüfung, ändert sich nichts am schon bekannten Programmablauf. Sollte a(i)=1

sein, zieht der Rechner eine andere Zahl aus 10310; jede Zahl, die zum erstenmal erscheint, wird in 10330 als erschienen gekennzeichnet: a(i)=1.

2. Es wird im Unterprogramm der Punktauswertung bei 3000ff festgelegt, daß a(i) nur bei den Teilprüfungen (tp<4) im Falle einer richtigen Antwort um 1 erhöht wird (ri ist bei einer richtigen Antwort ungleich 2: <>2):

```
3075 if tp<4 then if ri<>2 then a(i)=a(i)+1
```

3. Schließlich muß in 10370ff nach Teilprüfungen und Gesamtprüfung getrennt geprüft werden, wie oft der Rechner noch nach 10310 zurückgeschickt wird und neue Fragen zu stellen hat:

```
10370 if tp=4 then if f<r then 10310
10371 if tp<4 then if v<r3*2 then 10310
```

Bei der Gesamtprüfung geht der Rechner nach 10310 zurück, solange die Anzahl der Fragen kleiner ist als die Anzahl aller im Rechnerspeicher vorhandenen Vokabeln, solange f kleiner ist als r. Solange sind noch a(i)'s vorhanden, die in 10330 noch nicht auf 1 erhöht worden sind.
Bei den Teilprüfungen wird der Rechner nach 10310 zurückgeschickt, solange die Anzahl der richtigen Antworten (v) kleiner ist als die Anzahl der eingespeicherten Vokabeln multipliziert mit 2 (r*2). Solange sind auch noch a(i)'s vorhanden, die in 3075 noch nicht auf 2 erhöht worden sind.

****

Es bleiben nur noch wenige Veränderungen im Programmteil der Endauswertung zu programmieren. Wir erinnern uns, daß die Punktzahl abhängig war von der benötigen Zeit pro Frage und dem Prozentsatz der richtigen Antworten. Nun ist klar, daß in den Teilprüfungen vor allem gegen Schluß der Prüfungen die Antworten immer schneller gegeben werden können, weil die Wörter gerade erst eingepaukt würden. Wir müssen deshalb

die Maximalzeit aus 11210 getrennt nach Teilprüfungen und Gesamtprüfung festlegen:

```
11210 t=20:if tp<4 then t=15
```

t wird für alle Fälle als 20 definiert. Nur in den Teilprüfungen bei tp<4 wird t neu als 15 definiert.
Wir müssen außerdem bedenken, daß in den Teilprüfungen der Prozentsatz der richtigen Antworten am Ende der Prüfung falsch hoch wird, da der Rechner jedes Wort solange prüft, bis es endlich richtig eingegeben wird. Der Gesamtprüfling ist daher gegenüber dem Teilprüfling benachteiligt. Um diese Benachteiligung auszugleichen, erhöhen wir den Korrekturfaktor aus 11240 für die Gesamtprüfung auf 5:

```
11240 ko=4:if tp=4 then ko=5
11241 pz=pz*ko
```

Dies war die letzte Änderung!! Unser Trainingsprogramm kann wieder benutzt werden und leistet noch Erstaunlicheres als zuvor. Unser Gedächtnis kann sich selbst den schwierigsten Wörtern nicht verschließen, ja, oft sind es gerade diejenigen Vokabeln, mit denen die Schüler die größten Schwierigkeiten haben, die am Ende am besten beherrscht werden. Wir könnten das Programm in dieser Form stehen lassen, doch noch zwei weitere Verbesserungen sind nötig, um unserer Arbeit einen seriösen Zuschnitt zu geben:

1.
Wir sehen für schwache Teilprüfungen einen Rausschmiß vor.
2.
Wir programmieren eine "Sonderprüfung" über die Fehler mehrerer Kapitel und speichern unsere Tagesfehler auf Diskette...

Wir haben gelernt:
- die Ziehung der Zufallszahlen bei Teilprüfungen und Gesamtprüfung getrennt zu handhaben.
- für Teilprüfung und Gesamtprüfung unterschiedliche Maximalzeiten für die Punktauswertung zu definieren.

19. Kapitel

Wir bereiten in diesem Kapitel schlecht vorbereiteten Schülern eine Überraschung, indem wir sie unterhalb eines bestimmten Leistungsniveaus aus der Prüfung entlassen. Die Vokabeln werden noch einmal eingeblendet und für die doppelte Wiederholung in der Endauswertung ein Punktabzug in Rechnung gestellt.
Schließlich führen wir eine "Superprüfung" ein: es soll den Schülern die Möglichkeit gegeben werden, die Fehler aus mehreren Lektionen zu sammeln und diese noch einmal exklusiv im Trainingsmodus zu bearbeiten.

Schon frühe Erfahrungen zeigen, daß die Trainingseinrichtung der Teilprüfungen, die wir im letzten Kapitel vervollständigt haben, ihre Grenzen dann hat, wenn entweder die Wortliste zu schwierig war oder die Schüler zu oberflächlich gelernt haben. Sobald der Prozentsatz der richtigen Antworten während einer Teilprüfung unterhalb von 35% oder gar 30% liegt, wird das Lernen äußerst anstrengend, und die Gefahr besteht, daß die Schüler vor Überlastung aufgeben. Es ist daher durchaus im Interesse unserer Schüler, wenn wir sie bei unzulänglichen Leistungen aus der Prüfung entlassen und zurück zur Vokabelwiederholung schicken. Daß damit ein Punktabzug verbunden ist, versteht sich von selbst. Wir legen fest, daß von einem bestimmten Zeitpunkt an (etwa ab der 10. Frage) bei jeder Frage-Antwort-Sequenz geprüft wird, ob der Prozentsatz der richtigen Antworten oberhalb eines Schwellenwertes liegt, den wir auf 35% festsetzen. Liegt er darunter, gehen wir in ein Unterprogramm, das den Prüfungsabbruch bekanntgibt und blenden die Vokabeln noch einmal ein. Eine Reihe von Änderungen sind notwendig, vor allem dürfen wir nicht vergessen, einige wichtige Variablen auf Null zu setzen. Wir lassen in 10361 bei jedem Durchgang prüfen, ob in den Teilprüfungen (tp<4) nach der neunten Frage (f>9) der Prozentsatz der richtigen Antworten kleiner als 35 (l<35) geworden ist. Im positiven Fall gehen wir in ein neues Unterprogramm bei 3500ff und beginnen, von dort zurückgekehrt, die Prüfung von neuem:

```
10361 if tp<4 then if f>9 then if l<35 then gosub 3500:goto
      10300

3500 rem ****** "Prüfungsabbruch" ******
3510 gosub 10
3520 t$="Ungültiger Versuch!":y=11:gosub 100
3530 for i=1 to 2000:next
3540 t$="Zu hoher Fehlerquotient!":y=13:gosub100
3550 for i=1 to 2000:next
3560 gosub 1000
3570 f=0:v=0:fe=0
3580 for i=1 to r:a(i)=0:next
3590 pa=pa+1000
3600 return
```

Von 3520 bis 3550 wird, mit zwei Warte-Schleifen für den Rechner, den Schülern der ungültige Prüfungsversuch bekanntgegeben. Aus 3560 geht der Rechner nach 1000ff und blendet die Vokabeln des zugehörigen Drittel ein. Von dort zurück, müssen unbedingt folgende Werte auf Null gesetzt werden:

- die Anzahl der gestellten Fragen (f)
- die Anzahl der richtigen Fragen (v)
- die Anzahl der Fehler (fe)
- die Feldvariable a(), in der gespeichert wurde, welche Fragen bereits vorgestellt waren.

Beachte auch 3590, wo für jeden Durchgang durch die Abbruchsroutine der Wert für pa um 1000 hochgezählt wird. Dieser Wert geht nun in die Endabrechnung ein:

```
11245 pz=pz-pa
11246 pa=0
```

Beachte, daß pa in 11246 sofort nach Gebrauch wieder auf Null zurückgesetzt werden muß!

****

Den Fortgeschrittenen unter unseren Schülern wird es schon bald möglich sein, in einer Sitzung mehr als eine Lektion als Gesamtprüfung zu bearbeiten. Je fortgeschrittener die Schüler sind, um so weniger Fragen werden falsch beantwortet. Die Erfahrung zeigt aber, daß es immer wieder die gleichen Wörter einer Lektion sind, bei denen Fehler gemacht werden. Es ist daher nur wünschenswert, in unser Programm eine Vorrichtung einzubauen, durch die wir die Fehler mehrerer Lektionen sammeln und schließlich noch einmal getrennt bearbeiten lassen. Wir sammeln die Fehler in den Feldvariablen c$() und d$(), die wir in 3 dimensionieren:

```
3 dim a$(49),b$(49),c$(49),d$(49)
```

und zählen die Fehler über mehrere Lektionen in der Variablen po (SammelPOol). Zuvor fragen wir aber die Schüler, die sich gerade für eine Gesamtprüfung entschieden haben, ob sie die falschen Antworten getrennt zwischenspeichern möchten. Wir ändern:

```
10153 if a$="4" then tp=4:goto 10160
```

und fahren fort:

```
10160 gosub 10
10165 x=10:y=11:gosub 20:? "Fehler sammeln?"
10166 y=13:gosub 20:? "<1>  Nein"
10167 y=14:gosub 20:? "<2>  Ja"
10170 get a$
10171 if a$="1" then 10220
10172 if a$="2" then sa=1:goto 10220
10173 goto 10170
```

Wir fragen, ob die Fehler gesammelt werden sollen, und setzen bei der Antwort "ja" die Variable sa (SAmmeln) auf 1. Da wir außerdem voraussetzen können, daß die Entscheidung für das Fehlersammeln auf eine gewisse Professionalität unserer Schüler hinweist und die Frage nach Teilprüfungen zwischenzeitlich obsolet wird, überspringen wir für die

nächsten Kapitel die Frage nach Teil- oder Gesamtprüfung. Wir verwandeln Zeile 10115 in 10116 und schreiben nach 10115:

```
10115 if sa=1 then 10220
```

Wir gehen in das Unterprogramm bei 3000ff und legen dort fest, daß bei einer fehlerhaften Antwort das Vokabelpaar in die Variablen c$() und d$() zwischengelagert wird:

```
3001 if ri<>2 or sa=0 or po>47 then 3010
3006 po=po+1
3007 c$(po)=a$(i):d$(po)=b$(i)
```

Beachte die **Dreifachbedingung mit or** in 3001. Dem Rechner wird gesagt, er solle nach 3010 weitergehen, wenn eine der drei Bedingungen erfüllt ist. Er geht nach 3010, wenn
1. die Antwort richtig war, ri<> 2 ist
2. kein Fehlersammeln erwünscht ist, sa gleich 0 ist
3. der Fehlerpool voll ist, po größer als 48 ist.

Nur wenn keine dieser drei Bedingungen erfüllt ist, soll der Rechner in 3006 po hochzählen und in 3007 das Wortpaar a$(i)/b$(i) in c$(po)/d$(po) ablegen.

Wir ändern noch die Anzeige über den aktuellen Prüfungsstand, damit der "Sammler" darüber informiert ist, wieviele Fehler er bereits zwischengespeichert hat:

```
3050 ? "Fragen   o.k.   falsch   Pool   o.k.(%)"
3070 ? tab(4) f tab(12) v tab(20) f-v tab(27) po tab(34) l
```

Nach dem Ende der Prüfung und der Bekanntgabe von Einzel- und Gesamtergebnissen fragen wir nun zusätzlich -allerdings nur, wenn sa=1 ist-, ob die gesammelten Fehler zur Prüfung zubereitet werden sollen:

```
19000 rem **** "Fehler zur Prüfung" ****
19010 if sa=0 then 20000
```

```
19020 gosub 10
19030 t$="Fehler für die Prüfung vorbereiten?"
19040 y=10:gosub 100
19050 x=13:y=12:gosub 20:? "<1> Nein"
19060 y=13:gosub 20:? "<2> Ja"
19070 get a$
19080 if a$="1" then 20000
19090 if a$="2" then 19200
19100 goto 19070
```

Wird die Frage positiv beantwortet, geht der Rechner nach 19200ff, wo die Fehler (Anzahl der Fehler ist gleich po!) von c$() und d$() nach a$() und b$() hinübergespielt werden, po in r abgelegt und sa wieder auf Null gesetzt wird:

```
19200 for i=1 to po
19210 a$(i)=c$(i):b$(i)=d$(i)
19220 next
19230 r=po:po=0
19240 sa=0
```

Und genialer noch:

```
19300 gosub 10
19310 t$="Fehler auf Diskette speichern?"
19320 y=20:gosub 100
19330 x=13:y=22:gosub 20:? "<1>  Nein"
19340 y=23:gosub 20:? "<2>  Ja"
19350 get a$
19360 if a$="1" then 10115
19370 if a$="2" then 19400
19380 goto 19350
```

nebst:

```
19400 gosub 10
19410 x=10:y=11:gosub 20
19420 input "Kapitelnummer";kp$
19430 f$="fe"+kp$
19440 open 2,8,2,"@0:"+f$+"s,w"
```

```
19450 for i=1 to r
19460 print#2,a$(i):print#2,b$(i)
19470 next
19480 print#2,"r":print#2,"r"
19490 close2
19500 goto 10115
```

Wir fragen in 19300ff, ob die Fehler auch auf der Diskette gespeichert werden sollen und gehen im negativen Fall direkt zur Prüfung, im positiven Fall nach 19400ff weiter. Dort wird die Kapitelnummer erfragt, unter der die Fehler gesichert werden sollen. Aus der eingegebenen Nummer (kp$) und den Kennbuchstaben "fe" wird in 19430 der Filename f$ gebildet, unter dem die Daten in 19440 bis 19490 niedergeschrieben werden. Beachte 19480! Dort werden unsere Erkennungsbuchstaben "r","r" ans Ende des Kapitels geschrieben, damit der Rechner beim späteren Einlesen weiß, wann er von der Diskette ablassen darf.

An den Anfang unseres Trainingsprogramms gehört jetzt lediglich die Frage, ob das Kapitel, das wir bearbeiten wollen, ein normales, mit dem Generatorprogramm erstelltes, oder ein Fehlerkapitel ist:

```
10020 x=10:y=10:gosub 20
10021 ? "<1>  Generatorkapitel"
10022 y=11:gosub 20:? "<2>  Fehlerkapitel"
10023 get a$
10024 if a$="1" then id$="ve":goto 10030
10025 if a$="2" then id$="fe":goto 10030
10026 goto 10023
10030 y=13:gosub 20:input "Kapitelnummer";a$
10031 f$=id$+a$
```

Es wird gefragt, ob die Schüler Generator- oder Fehlerkapitel wählen. Bei Generatorkapiteln wird id$ als "ve", bei Fehlerkapiteln als "fe" definiert. Zusammen mit der Kapitelnummer a$ aus 10030 entsteht der Name des Vokabelfiles f$.

Zum Ende noch eine kleine Subtilität. Was, wenn wir die Nummer eines Fehlerkapitels eingeben, das gar nicht existiert? Der Rechner reagiert darauf, daß er unter f$ nichts in der Directory der Diskette findet, mit Unbehagen und steigt mit der Fehlermeldung "file not found error" aus dem Programm aus. Um dies zu vermeiden, lassen wir den Rechner deshalb zuerst prüfen, ob ein Kapitel mit dem Namen f$ überhaupt existiert. Die folgende Programmfolge ist nicht unkompliziert. Wir fügen ein:

```
4     open 15,8,15
10041 input#15,jx
10042 if jx<20 then 10060
10043 gosub 10
10044 t$="Gewählte Kapitel nicht vorhanden!"
10045 y=11:gosub 100
10046 for i=1 to 3000:next
10047 close2
10048 goto 10000
```

Wir öffnen in 4 den Fehlerkanal 15. Sobald wir in 10040 das sequentielle File unserer Wahl eröffnet haben, laden wir in 10041 über #15 den Wert jx ein. Ist jx kleiner als 20, so ist das File mit Namen f$ gefunden und eröffnet worden; der Rechner geht weiter nach 10060. Hat der Rechner aber das File f$ nicht vorgefunden, wird jx größer als 20. In diesem Fall ist die Bedingung aus 10042 nicht erfüllt, der Rechner geht weiter nach 10043ff, löscht den Schirm, gibt die Meldung über den unerwünschten Zwischenfall aus, zählt bis 2000 und geht nach 10000 zurück, nachdem er #2 in 10047 verschlossen hat.

Unser Trainingsprogramm ist in der Form fertig, wie wir es in dem vorliegenden Lehrbuch entwickeln konnten! Im nächsten Kapitel geben wir einige Hinweise zur Arbeitsweise und zu den Lernerfolgen mit dem selbstgestrickten Werk. Noch fast jeder Schüler, der sich überreden ließ, sich auf das augenscheinlich simple Frage-Antwort-Spiel einzulassen, wurde innerhalb kurzer Zeit süchtig und steigerte sich in

einen wachsenden Wahn nach immer mehr Vokabeln. Manch einer wollte von der Maschine gar nicht mehr ablassen und zeigte sich empfindlich gestört durch jede Art von Unterbrechung. Spiel muß sein, auch Wettkampf mit sich und der Maschine, und für die derart Süchtigen bauen wir eine Vorrichtung zur Unterbrechung ein für den Fall einer unaufschiebbaren alltäglichen Verpflichtung. Wenn die Schüler also in Wettkampfstimmung sind und plötzlich das Telefon schellt, der Postbote eine Unterschrift verlangt oder liebe Freunde auf der Matte stehen, können sie in Zukunft "yy" anstelle der Antwort eingeben, wodurch das Programm anhält und die augenblickliche Zwischenzeit festgehalten wird:

```
2061 if an$<>"yy" then 2065
2062 gosub10:   t$="Fortsetzung mit beliebiger
     Taste":y=11:gosub 100:it$=ti$
2063 get a$:if a$="" then 2063
2064 ti$=it$:gosub 10:goto 2000
```

Eine beliebige Zwischentaste führt das Programm mit der alten Zeit zurück nach 2000, als sei nichts gewesen...

Wir haben gelernt:

- unsere Schüler aus der Prüfung zu entlassen, wenn ihre Leistungen zu schlecht sind.
- wie eine Dreifachbedingung mit or funktioniert.
- auf der Diskette zu prüfen, ob dort ein sequentielles File mit einem bestimmten Namen existiert.
- unsere Fehler in ein "Fehlerfile" mit den Kennziffern "fe" abzuspeichern.
- den Süchtigen unter unseren Schülern die Gelegenheit zu geben, unbesorgt auf Toilette, zur Schelle oder zum Telefon zu gehen, ohne daß dadurch das Prüfungsergebnis Schaden leidet...

Ü B E R G A N G

20. Kapitel

Die besten Programm nützen nichts, wenn ihnen keine Gebrauchsanweisung beigegeben ist. Auch zu unserem Trainingsprogramm müssen wir kurz darstellen, wie die Schüler effektiv arbeiten können.
Bewährt hat sich folgendes Verfahren:

1.
Wir bearbeiten nacheinander die 3 Teilprüfungen einer Lektion. Nachdem der Vokabelschatz der Lektion "sitzt", stellen wir abschließend die Gesamtprüfung ein und gehen alle Wörter noch einmal durch.

2.
50 Wörter pro Tag zu lernen, ist nicht schlecht; 100 Wörter zu lernen, ist besser; allein befriedigend wird die Arbeit von 150 Wörtern aufwärts!

3.
Nach einigen Tagen wird zusätzlich zu den jeweils neu gelernten Kapiteln eine Wiederholung des in den letzten Tagen bearbeiteten Stoffes absolviert. Bereits gelernte Kapitel werden nunmehr nur noch als Gesamtprüfungen abgelegt, am besten mehrere Kapitel hintereinander. Danach Sammlung der Fehler und schließlich Teilprüfungen des neuen Fehlerkapitels!

4.
Eine intensive Lernperiode sollte sich nicht länger als einen Monat dahinziehen. Nach dieser Zeit brechen wir den Lernzyklus ab und verdrängen das Gelernte. Zu gegebenem Anlaß (Ferienreise etc) beginnen wir etwa 2 Wochen vor dem Abreisetermin mit einer Auffrischungswiederholung. Alle

gelernten Kapitel werden nacheinander als Gesamtprüfungen abgelegt. Wichtig ist die vorherige Vokabelwiederholung, ohne die die Ergebnisse schon nach wenigen Monaten katastrophal ausfallen; **mit** Vokabelwiederholung in der zulässigen Gesamtzeit von 5 Minuten liegen die Arbeitsergebnisse aber sensationell hoch! Zwei Wochen werden genügen, um einen in einem früheren Lernzyklus bearbeiteten Stoff von über 3000 Wörtern zu aktivieren.

Beachte: die Lernzeiten sowohl im ersten Lernzyklus als auch in einer späteren Wiederholungsphase sind sehr stark abhängig von der Schreibgeschwindigkeit des Schülers. Wer schneller schreibt, setzt sich in der Zeiteinheit einer größeren Zahl von Fragen aus, belastet sich stärker und lernt mehr. Nicht oft genug sollte dem Schüler ans Herz gelegt werden, daß die Ausbildung seiner Maschinenschreibfertigkeiten eine wesentliche Voraussetzung für Höchstleistungen im Computer-gestützen Sprachentraining ist.

Zur weiteren Information sei auf Teil C dieses Buches verwiesen. Dort ist ein Aufsatz zu kurz- und langfristigen Lernerfolgen im Vokabeltraining abgedruckt. Im Anschluß daran folgt ein Interview mit einem 24-jährigen Schüler, der während eines Italienisch-Intensivkurses innerhalb von drei Wochen die Basisgrammatik, 3500 Wörter und sämtliche Verbformen durch alle Zeiten lernte. Verständigungs- und Lesevermögen waren danach ausgezeichnet.

"Teil B"

Im zweiten Teil dieses Buches beheben wir einen unerträglichen Mangel des COMMODORE 64: Die Unfähigkeit, andere als englische Buchstaben auf den Bildschirm zu bringen. Durch Zusatzprogramme lehren wir den Rechner, französische, spanische und italienische Buchstaben in sein Repertoire aufzunehmen. Bevor wir jedoch in die nicht immer leicht einsehbare Materie der "Zeichensatzveränderung" einsteigen, schreiben wir zunächst unser Generatorprogramm aus den ersten Kapiteln um. Die Eingabe unserer Daten passen wir auf diese Weise bereits den Anforderungen an, die durch die Bearbeitung von Sprachen entstehen, in deren Alphabet ungewöhnliche Buchstaben vorkommen. Wir lehren den Commodore 64 unter anderem die deutschen Buchstaben ä,Ä,ö,Ö,ü,Ü und ß.

21. Kapitel

Das Generatorprogramm, das wir in den ersten Kapiteln ausgearbeitet haben, und mit dem wir die sequentiellen Files mit den Anfangsbuchstaben "ve" erstellen konnten, hat uns keinen schlechten Dienst erwiesen. Wir laden "generator" ein, schreiben die Daten in die data-Zeilen und starten das Programm, nachdem wir die Kapitelnummer des neuen Wortschatzes festgelegt haben. Eleganter wäre freilich, wenn wir die Daten direkt über ein Programm eingeben könnten, durch die sich einmal das Schreiben von "data" erübrigt und zum anderen schließlich die Möglichkeit gegeben wird, sofort oder nachträglich die Fehler bei der Dateneingabe zu verbessern, die sich leider immer wieder einschleichen. Am Ende wäre auch zu bedenken, ob wir in einem neuen Programm nicht vorsehen, daß nachträglich Wörter aus dem File gelöscht werden können, die nicht mehr befriedigen.
Wir löschen eventuell im Speicher vorhandene Programme mit "new" und beginnen mit der Abfassung einiger nützlicher Unterprogrammschritte, die wir vom Vokabeltrainingsprogramm kennen:

```
4 open 15,8,15
9 goto 100000
10 ? chr$(147):return
20 poke 211,x:poke 214,y:sys 58640:return
90 poke 781,y:sys 59903:return
99 poke 53280,0:poke 53281,0:poke 646,1:return
100 rem ***** Zentrierung *****
101 le=len(t$)
102 x=(38-le)/2
103 gosub 20:? t$
104 return
200 open 2,8,2," 0:"+f$+"s,w":return
210 open 2.8,2,f$+"s,r":return
59999 end
60000 save "@0:generator",8
60001 verify "generator",8
```

Beachte die neuen Fileöffnungsschritte aus 200 zum Datenschreiben und 210 zum Datenlesen sowie Zeile 99, in der Bildschirm- und Schriftfarbe festgelegt werden.

Wir beginnen unser Programm mit der Frage, ob ein neues Kapitel angelegt oder ein bereits bestehendes Kapitel verändert bzw. erweitert werden soll:

```
10000 rem *****  "Hauptprogramm"  *******
10010 gosub 10:? chr$(14)
10020 x=13:y=11:gosub 20:? "<1> Neues Kapitel einrichten"
10030 y=12:gosub 20:? "<2> Bestehendes Kapitel bearbeiten"
10040 get a$
10050 if a$="1" then gosub 10:goto 10100
10060 if a$="2" then 10500
10070 goto 10040
```

Wir legen für ein neues Kapitel fest, daß es maximal 48 Vokabeln speichern soll. Beachte, daß wir in 1 die Feldvariable a$() doppelt dimensionieren:

```
1 dim a$(49,2)
```

Die **doppelte Dimensionierung** bewirkt, daß für 49 Wörter (48 Vokabeln + der "r"-Marker für das Kapitelende jeweils 2 Speicherplätze vorgesehen werden. Es gibt nun a$(1,1) und a$(1,2); es gibt a$(2,1) und a$(2,2); es gibt a$(3,1) und a$(3,2) usw bis a$(49,1) und a$(49,2). Wir ersparen uns auf diese Weise die Dimensionierung von 2 verschiedenen Feldvariablen für ein Vokabelpaar. Wir erinnern uns, daß wir im Vokabeltrainingsprogramm a$() für die englischen und b$() für die deutschen Wörter dimensioniert hatten. Übertragen auf die doppelt dimensionierte Feldvariable a$(,) unseres jetzigen Programms ergibt sich folgende Verteilung unserer Wortliste auf die Einzelelemente:

	a$(..,1)	a$(..,2)
a$(1,..)	to besiege	belagern
a$(2,..)	dissident	andersdenkend
a$(3,..)	demonstrator	Demonstrant
a$(4,..)	to escape	entkommen
a$(5,..)	peace movement	Friedensbewegung
a$(6,..)	popular uprising	Volksaufstand
a$(7,..)	stone-throwing	steinewerfend
a$(8,..)	to struggle	kämpfen
a$(9,..)	to undermine	unterwandern
a$(10,..)	to survive	überleben

"steinewerfend" wäre hiernach in a$(7,2) gespeichert.
Wir fahren in unserem Programm fort, indem wir alle 49 Wortpaare der Feldvariablen a$(,) mit "r" beschriften:

```
10100 for i=1 to 49
10110 for k=1 to 2
10120 a$(i,k)="r"
10130 next k
10140 next i
10150 b=1
10160 goto 20000
```

Beachte die doppelte for-next-Schleife, in der folgendes geschieht:

Aus 10100 holt der Rechner sich den Wert i=1, aus 10110 k=1 und legt in 10120 "r" folglich in a$(1,1) ab. Von 10130 geht er zurück nach 10110 und erhöht k auf 2. Beim zweiten Durchgang durch 10120 wird somit a$(1,2) mit "r" belegt. Bei der erneuten Rückkehr nach 10110 wird k=3 und erfüllt nicht mehr die Bedingung der Schleife "for k=1 to 2". Der Rechner springt hinter 10130 und von 10140 zurück nach 10100. Dort erhöht er i auf 2, bekommt von 10110 den Wert 1 für k, belegt in 10120 a$(2,1) mit "r" und so weiter. Schließlich wird in 10150 die Variable b als 1 definiert; den Grund dafür verstehen wir später.

Falls wir ein bereits bestehendes Kapitel verändern oder erweitern wollen, geht der Rechner nach 10500:

```
10500 rem ** "Kapitel einspeichern" **
10510 x=7:y=14:gosub 20:input "Kapitelnummer";a$
10520 f$="ve"+a$
10540 gosub 210
10550 input#15,jx
10560 if jx<20 then 10600
10570 gosub 10:t$="Kapitel existiert nicht":gosub 100
10580 close2:for i=1 to 3000:next:gosub 10:goto 10000
```

Es wird nach der Kapitelnummer gefragt und der Filename mit der Vorwahl "ve" in f$ abgelegt (10520). In 10540 bis 10580 wird geprüft, ob ein solches File überhaupt existiert, um im negativen Fall einen Programmabbruch zu verhindern. Wenn das File auf der Diskette existiert -jx ist in diesem Fall kleiner als 20-, geht der Rechner weiter nach 10600:

```
10600 for i=1 to 49
10601 for k=1 to 2
10610 input#2,a$(i,k)
10620 next k
10630 if st=64 then 10700
10640 next i
17000 close2
10710 b=i
10720 for i=b to 49:for k=1 to 2:a$(i,k)="r":next k,i
```

In 10600/10640 werden solange die Vokabelpaare nach a$(,) eingelesen, bis a$(i,1) gleich "r" ist. Der Rechner verläßt daraufhin die for-next-Schleife, schließt das sequentielle File wieder und legt den letzten i-Wert (der dem "r" entspricht in b ab. Wenn das Kapitel weniger als 48 Wortpaare hat, ist b der Startwert für weitere Ergänzungen. Wenn b=15 ist, heißt dies, daß 14 Wortpaare bereits gespeichert sind und das nächste Wortpaar nach a$(15,1) und a$(15,2) abgelegt wird. In 10720 schließlich füllen wir von b ab aufwärts bis nach 49 alle a$(,) mit "r". Der Rechner geht nun weiter ins Zwischenmenu bei 20000, das wir im nächsten Kapitel vorstellen, und von dem aus wir die Wahl haben zwischen Kapitelerweiterung, Korrektur, Löschen oder Abspeichern.
Beachte 10630! Wenn das Diskettenlaufwerk am Ende einer sequentiellen Datei angelangt, stellt der Rechner die Variable st automatisch auf 64. Wir fragen nach dem Einlesen eines jeden Wortpaares den Wert von st ab. Ist er gleich 64, geht der Rechner weiter nach 10700.

22. Kapitel

Wir fahren fort mit der Programmierung eines Zwischenmenus für das neue Generatorprogramm:

```
21 y=y+1:gosub 20:return

20000 rem **** "Zwischenmenu" ****
20010 gosub 10
20020 x=10:y=9:gosub 20:? "<1> Erweiterung"
20030 gosub 21:? "<2> Korrektur"
20040 gosub 21:? "<3> Loeschen"
20050 gosub 21:? "<4> Abspeichern"
20060 gosub 21:? "<5> Zum Zentralverteiler"
20070 get a$
20080 if a$="1" then 25000
20090 if a$="2" then 30000
20100 if a$="3" then 35000
20110 if a$="4" then 40000
20120 if a$="5" then poke 44,8:run
20130 goto 20070
```

Wir löschen den Schirm und projizieren "<1> Erweiterung nach 10/9. Die nächsten 4 Zeilen entstehen, indem der Rechner zuerst in die neue Unterprogrammzeile 21 springt, y um 1 erhöht, von dort weiter nach 20 gosubbt und die Zauberformel spricht.
Bei Wahl <1> bis <4> geht der Rechner in die von 20080 bis 20110 definierten Programmteile weiter. Beachte vorerst nicht Zeile 20120 mit einer noch unbekannten poke-Anweisung, die erst durch spätere Änderungen Sinn bekommt; im Augenblick bewirkt Wahl <5> nur, daß der Rechner das Programm von neuem startet.

Wir gehen nach 25000ff und programmieren die Erweiterung unserer Vokabellisten:

```
25000 rem ****  "Erweiterung"  ****
25010 gosub 10
25020 if b<49 then 25100
```

```
25030 t$="Kapitel ist voll":y=11:gosub 100
25040 for i=1 to 3000:next
25050 goto 20000

25100 t$="ERWEITERUNGSROUTINE":y=2:gosub 100
25101 t$="<r> Abbruch":y=23:gosub 100
25110 for i=b to 48
25120 x=10:y=9:gosub 20:? "<";i;">"
25130 for k=1 to 2
25140 x=8:y=10+k:gosub 20:input a$(i,k)
25150 if a$(i,k)="r" then a$(i,1)="r":goto 25200
25160 next k
25170 for y=9 to 12:gosub 90:next y
25180 next i
25200 b=i
25210 goto 20000
```

Zunächst wird in 25020 geprüft, ob b noch kleiner ist als 49 und im Kapitel damit noch Platz ist für weitere Vokabeln. (Wir erinnern: b ist um 1 größer als die Anzahl der schon vorhandenen Vokabelpaare.) Wenn bereits 48 Wortpaare eingespeichert sind, wird dies in 25030 bekanntgegeben, und der Rechner geht zurück zum Zwischenmenu.
Die Erweiterung selber beginnt mit einer Programmüberschrift aus 25100 und einer Anweisung, wie die Erweiterung abgebrochen werden kann aus 25101. Die Eingabe eines x auf die Inputfragezeichen läßt den Rechner aus dem Erweiterungs-Programmteil aussteigen.
Im folgenden finden wir insgesamt 3 for-next-Schleifen dichtgedrängt auf kleinem Raum. Die umfassendste for-next-Schleife ist die i-Schleife 25110/25180, die Buch führt über die laufende Nummer des gerade einzugebenden Wortpaares; in 25120 gibt der Rechner die laufende Wortnummer am Koordinatenpunkt 10/9 auf den Schirm aus.
Die zweite for-next-Schleife ist die k-Schleife 25130/25160, die für jedes Wortpaar zweimal durchlaufen wird und in 25140 damit die beiden Wörter in a$(i,1) und a$(i,2) einfragt. Die Lokalisation des inputs erfolgt für die y-Koordinate über eine Variable: y=10+k!! Das erste Wort wird demnach in der 11.Zeile, das zweite Wort in der 12.Zeile verlangt. In 25150

prüft der Rechner nach jedem input, ob es mit "x" identisch ist und geht im positiven Fall über das Ende der i-Schleife hinweg nach 25200. Dort wird mit b=i diejenige Wortnummer definiert, bei der eine erneute Erweiterung zu beginnen hätte, und der Rechner kehrt -wie übrigens auch dann, wenn die Erweiterung bis zum 48. Wortpaar gelangt ist und die i-Schleife sich erschöpft hatte- nach 20000 ins Zwischenmenu zurück. Beachte die dritte for-next-Schleife aus 25170, aus der der Rechner viermal mit y-Werten 9,10,11 und 12 in das Unterprogramm bei 100ff springt und die entsprechenden Bildschirmzeilen freimacht für das nächste Wortpaar.

Wenn wir die Eingabe der Vokabeln beendet haben, ist es empfehlenswert, die hinzugefügten Wörter noch einmal auf Eingabe- und Rechtschreibefehler durchzusehen. Wir blenden zu diesem Zweck die Wortpaare auf den Schirm ein und versehen sie mit ihrer Wortnummer. Auf Wahl <2> aus dem Zwischenmenu 20000ff geht der Rechner nach 30000 und findet vor:

```
30000 rem ****  "Korrektur"  ****
30010 s=1
30015 gosub 10
30020 for y=0 to 19:gosub 90:next
30025 if s=4 then s=1
30030 if s=1 then g=1:h=16
30040 if s=2 then g=17:h=32
30050 if s=3 then g=33:h=48
30060 ? chr$(19):for i=g to h
30070 ? i;a$(i,1) tab(21) a$(i,2)
30080 next
30100 x=10:y=20:gosub 20:? "<1> Blaettern"
30110 gosub 21:? "<2> Korrektur"
30120 gosub 21:? "<3> Ins Zwischenmenu"
30130 get a$
30140 if a$="1" then s=s+1:goto 30020
30150 if a$="2" then 31000
30160 if a$="3" then 20000
30170 goto 30130
```

Diese Passage ist vom Trainingsprogramm noch hinlänglich bekannt. Für s werden bei Werten zwischen 1 und 3 verschiedene Werte für g und h definiert, die dann bestimmen, welche 16 Vokabeln auf dem Bildschirm ausgegeben werden. Der Wert s, in 30010 als 1 definiert, wird jedesmal um 1 erhöht, wenn wir "Blättern" aus 30100 wählen. Sollte s gleich 4 werden, also beim Weiterblättern von der dritten Seite, wird s in 30025 auf 1 hinuntergesetzt und Seite 1 erscheint von neuem. Wahl 3 führt zurück ins Zwischenmenu, Wahl 2 den Rechner weiter in die eigentliche Korrekturroutine.
Wir notieren die Wortnummer, wenn wir bei der Durchsicht der Vokabelliste Fehler entdecken. Der nächste Programmabschnitt gibt den Zugriff auf diese fehlerhaften Daten:

```
31000 gosub 10
31005 y=23:t$="Mit 0 abbrechen":gosub 100
31010 x=10:y=3:gosub 20:input "Kennziffer des Wortpaares";p
31015 if p>=b then 31000
31020 if p=0 then 20000
31025 gosub 90
31030 for k=1 to 2
31040 x=10:y=10+k:gosub 20
31050 ? a$(p,k)
31060 next
31070 y=20:gosub 20:input "Wort 1 oder Wort 2";w
31080 gosub 90
31090 if w<1 or w>2 then for y=11 to 12:gosub 90:next:goto
      31010
31100 y=8:gosub 20:? a$(p,w)
31110 y=10+w:gosub 90
31120 x=8:gosub 20:input a$(p,w)
31130 y=8:gosub 90
31140 x=10:goto 31070
```

Der Rechner blendet zuerst in Zeile 23 die Mitteilung für den Benutzer ein, daß die Eingabe einer Null das Programm zur letzten Stufe zurückführt. Dann wird die Kennziffer des Wortpaares verlangt, das wir uns bei der Durchsicht der Vokabelliste als fehlerhaft notiert hatten. Wenn p=0 ist,

geht der Rechner zum Zwischenmenu zurück. Wenn p>=b ist (was bedeutet: wenn p größer als b oder gleich b ist), startet der Programmabschnitt noch einmal durch. Wir erinnern uns, daß bei b noch keine Wörter gespeichert sind, sondern b vielmehr die erste Wortkennziffer ist, die bei einer Erweiterung beschriftet wird!
In 31025 wird die Zeile mit der letzten Frage gelöscht (y ist ja noch als 3 definiert). Die k-Schleife 31030/31060 projiziert das gewünschte Wortpaar auf den Schirm. Aus 31070 wird in Bildschirmzeile 20 gefragt, ob Wort 1 oder Wort 2 verändert werden soll. Nach der Eingabe einer 1 oder einer 2 wird Zeile 20 aus 31080 wieder gelöscht und das geforderte Wort noch einmal zur Erinnerung in Bildschirmzeile 8 ausgegeben (31100). Am ursprünglichen Platz wird dasselbe Wort gelöscht (31110) und das input-Fragezeichen wartet auf die Eingabe der korrekten Form (31120). Sobald dies geschehen ist, wird die Gedächtnisstütze aus Zeile 8 gelöscht (30130) und der Rechner geht zurück nach 31070. Von hier aus könnten wir bei Bedarf noch das andere Wort korrigieren. Bei Eingabe einer Null streicht der Rechner die beiden Wörter vom Bildschirm (31090) und geht zurück nach 31010 und fragt nach einer weiteren Wortpaarkennziffer. Geben wir auch hier eine Null, wird das Zwischenmenu eingeblendet.

Vom Zwischenmenu gehen wir mit Wahl <4> weiter zum Abspeichern der Daten auf Diskette:

```
40000 rem ****  "Abspeichern"  ****
40010 gosub 10
40020 y=11:x=10:gosub 20:input "Kapitelnummer";k$
40030 f$="ve"+k$
40050 gosub 210
40060 input#15,jx:close2
40070 if jx>20 then 41000
40080 gosub 10
40090 x=7:y=13:gosub 20:? "Kapitel ist bereits vorhanden"
40100 y=15:gosub 20:? "<1> Nicht ueberschreiben"
40110 gosub 21:? "<2> Ueberschreiben"
40120 get a$
```

```
40130 if a$="2" then 41000
40140 if a$="1" then 40000
40150 goto 40120
```

Wir fragen als erstes nach der Kapitelnummer k$, die mit dem Vorspann "ve" zum Kapitelnamen für das sequentielle File zusammengesetzt wird. Alsdann tun wir in 40050 so, als wollten wir unser Kapitel f$ zum Lesen öffnen. Wir holen den Wert jx aus dem Fehlerpool, verschließen das File mit Namen f$ und prüfen, ob jx größer als 20 ist. Wenn dies der Fall ist, heißt das, daß der Rechner das File f$ nicht öffnen konnte, weil es nicht existierte, und wir können beruhigt in unserem Programm bei 41000ff fortfahren. Ist jx aber kleiner als 20, so heißt dies , daß das File f$ aus 40050 mit Erfolg zum Lesen geöffnet wurde un somit bereits existiert. Um zu verhindern, daß bereits abgespeicherte Daten von den gerade im Speicher vorhandenen Daten überschrieben und gelöscht werden, geben wir bekannt, daß ein Kapitel mit der gerade eingegebenen Nummer bereits existiert und fragen, ob es überschrieben werden soll. Wahl 2 geht weiter nach 41000. Bei Wahl 1 geht der Rechner zurück nach 40000 und erfragt eine neue Kapitelnummer.

Bei 41000ff beginnt das Sichern unserer Daten auf Diskette:

```
41000 gosub 10
41010 x=18:y=11
41020 gosub 200
41030 for i=1 to b
41040 gosub 20:? i
41050 for k=1 to 2
41060 print#2,a$(i,k)
41070 next k,i
41080 close2
```

Wir definieren in 41010 die Koordinate 18/11, wo später aus 41040 die Nummer des gerade gespeicherten Wortpaares eingespielt wird, damit wir dem Speichervorgang folgen können. Das File f$ wird in 41020 eröffnet und anschließend die Wortpaare 1 bis b (womit auch der Marker "r" für das

Ende des sequentiellen Files verbucht wird) durch i- und k-Schleife auf die Diskette gespeichert werden. Nach getaner Arbeit wird das File 2 wieder ordnungsgemäß verschlossen.
Wir wissen, wie lästig und zeitraubend das Eingeben von Daten ist, und es ist sinnvoll, die gerade verrichtete Arbeit vorsichtshalber noch einmal auf eine andere Diskette abzuspeichern. Wir geben im Abschlußmenu daher diese Wahlmöglichkeit:

```
42000 gosub 10
42010 x=7:y=10:gosub 20:? "<1> Noch einmal abspeichern"
42020 gosub 21:? "<2> Zurueck ins Zwischenmenu"
42030 gosub 21:? "<3> Neustart des Programms"
42040 get a$
42050 if a$="1" then 43000
42060 if a$="2" then 20000
42070 if a$="3" then run
42080 goto 42040
43000 gosub 10
43010 t$="Nach Diskettenwechsel beliebige Taste":gosub 100
43020 get a$:if a$="" then 43020
43030 gosub 10:goto 40040
```

Zum den Einzelheiten der letzten Erweiterung sind keine weiteren Erläuterungen notwendig. Bei Wahl <2> geht der Rechner zurück nach 20000 ins Zwischenmenu; bei Wahl <3> wird das Programm neu gestartet; bei Wahl <1> wird Zeit für den Diskettenwechsel gegeben, und eine beliebige Taste befreit den Rechner aus 43020 und geht zurück nach 40040.

Zur Vervollständigung unseres neuen Generatorprogramms muß nur noch die Passage 35000ff programmiert werden, die es uns ermöglicht, einzelne Wortpaare aus der Vokabelliste zu löschen. Wir fragen zuerst nach der Kennziffer des zu löschenden Wortpaares und blenden es zur Kontrolle noch einmal auf den Schirm ein:

```
35000 gosub 10
35010 t$="Abbruch mit 0":y=23:gosub 100
35020 x=8:y=2:gosub 20:input "Kennziffer des Wortpaares";p
35030 if p>=b then 35000
35040 if p=0 then 20000
35050 for k=1 to 2
35060 y=10+k:gosub 20:? a$(p,k):next
35070 t$="<1> Löschen - <2> Nicht löschen":y=23:gosub 100
35080 get a$
35090 if a$="1" then 35200
35100 if a$="2" then 35000
35110 goto 35080
35200 for k=1 to 2
35210 a$(p,k)=a$(b-1,k):next
35220 b=b-1
35230 goto 35000
```

In 35030 wird geprüft, ob die eingegebene Kennziffer überhaupt zulässig ist, andernfalls noch einmal bei 35000 von vorn begonnen. Mit 0 geht der Rechner ins Zwischenmenu (35040). Die k-Schleife 35050/35060 bringt das geforderte Wortpaar auf den Schirm und zusätzlich aus 35070 die Frage, ob tatsächlich gelöscht werden soll. Wahl 2 führt zurück nach 35000, Wahl 1 weiter nach 35200ff. In der k-Schleife 35200/35210 wird das letzte Wortpaar unserer Vokabelliste mit der Nummer b-1 in denjenigen Platz der Feldvariable a$(,) hineinkopiert, wo das zu löschende Wort steht. Die beiden Wörter mit der Kennzahl p werden also überschrieben, und der Rechner geht nach 35000 zurück, nachdem in 35220 der Wert b neu definiert ist als b-1, weil nun ein Wortpaar weniger als zuvor im Speicher steht.

Unser Generatorprogramm ist vollendet. Bis auf Zeile 20120 sind alle Passagen erklärt und verständlich geworden. Damit haben wir zusammen mit dem Vokabeltrainingsprogramm zwei leistungsfähige Programme, die wir im nächsten Kapitel zu einem "Programmpaket" zusammenfassen werden. Wir programmieren dort die Möglichkeit, von einem Programm zum anderen zu springen und lernen dabei noch eine Reihe von nützlichen Tricks im Umgang mit dem C64 kennen.

23. Kapitel

Wir schreiben ein neues Programm, das den Benutzer fragt, ob er weiter ins Trainingsprogramm oder aber zum Generatorprogramm will:

```
1 goto 10000
10 ? chr$(147):return
20 poke 211,x:poke 214,y:sys 58640:return
90 poke 781,y:sys 59903:return
99 poke 53280,0:poke 53281,0:poke 646,1:return
100 le=len(t$)
101 x=(36-le)/2
102 gosub 20:? t$
103 return
10000 rem *****  "Zentralverteiler/Hauptprogramm"  *****
10010 gosub 99:gosub 10:? chr$(14)
10020 t$="ZENTRALVERTEILER":y=2:gosub 100
10030 x=8:y=11:gosub 20:? "<1> Trainingsprogramm"
10040 y=12:gosub 20:? "<2> Generatorprogramm"
10060 get a$
10070 if a$="1" then n$="vokabel":goto 10100
10080 if a$="2" then n$="generator":goto 10100
10090 goto 10060
```

Bis hierher bietet das Programm keine Schwierigkeiten. Vor 10000 stehen die schon bekannten Unterprogrammschritte der letzten beiden Programme. Der Rechner fragt, ob wir das eine oder andere Programm nachladen wollen und definiert in Abhängigkeit von unserer Antwort n$ mit dem Programmnamen. Es folgt:

```
10100 gosub 10
10110 x=0:y=3:gosub 20:? "load ";chr$(34);n$;chr$(34);",8"
10120 y=8:gosub 20:? "run:"
```

Wenn wir das Programm durchstarten lassen, sehen wir, daß nach dem Löschen des Bildschirms in die 3.Bildschirmzeile die Anweisung steht, die wir gemeinhin zum Laden eines Programm schreiben. Bei Wahl <1> wird als n$="vokabel", bei

Wahl <2> "generator" ausgegeben. In der 8.Zeile erscheint "run". Beachte das zweimalige "chr$(34)" in 10110! Nur auf diese Weise können Anführungsstriche (") mit einer Print-Anweisung auf den Bildschirm gebracht werden!
Das Programm geht weiter mit dem genialen Passus:

```
10130 ? chr$(19)
10140 for i=631 to 633:poke i,13:next
10150 poke 198,3:end
```

Mit chr$(19) geht der Cursor in die linke obere Bildschirmecke. In 10140 nun werden in die Speicherstellen 631, 632 und 633 der Wert 13 hineingepokt. Diese 13 in 631 bis 633 bedeutet, daß dort ein dreimaliges Return bereitgehalten wird. Mit poke 198,3 werden diese drei Returns aktiviert: mit dem ersten lädt der Rechner sich das Programm ein, daß in der bekannten Ladeformel in Zeile 3 steht; mit der zweiten wird auf das "run" aus Zeile 8 ein Return gegeben; das dritte Return verliert sich im Rechner, ist jedoch für die Ausführung des Programms wichtig. Beachte, daß erst durch "end" aus 10150 das 198er-poke aktiviert wird!

Als einzig häßlicher Punkt bleibt in unserem Programm, daß die Befehle aus Zeile 3 und 8, die das nächste Programm laden und starten, unser Anfangsmenu entstellen. Wir können dies umgehen, indem wir Zeile 3 und 8 in der Farbe des Bildschirmhintergrundes schreiben, also schwarz auf schwarz, was den Rechner nicht stört:

```
10101 poke 646,0
```

Damit unser nächstes Programm nicht auch schwarz auf schwarz und unerkenntlich auf den Schirm projiziert wird, stellen wir in "generator" und "vokkabel" die Schriftfarbe wieder weiß:

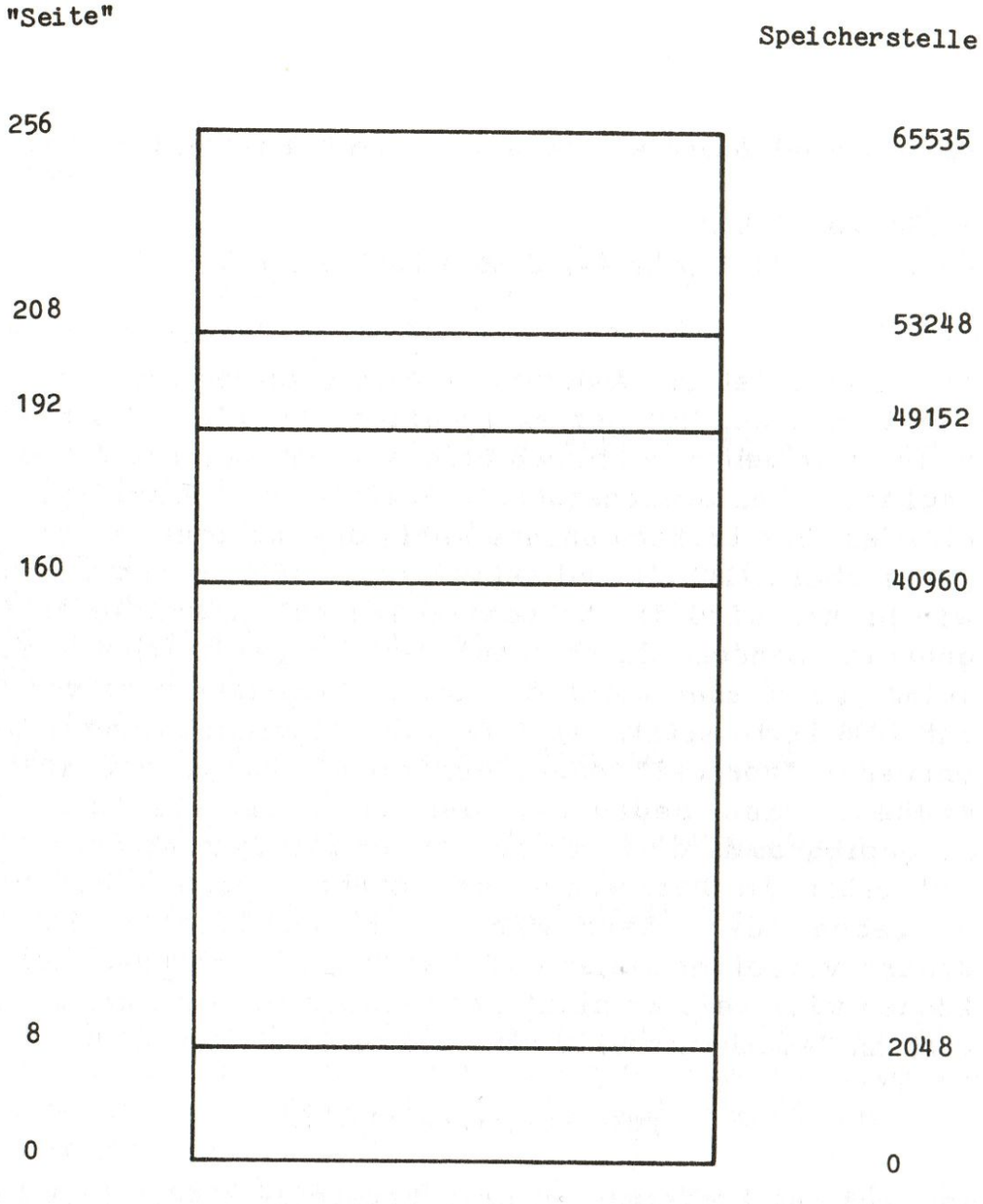
```
7 poke 646,1
```

Es bleibt nur noch, das Programm mit einem Namen, beispielweise "go" zu versehen und auf Diskette zu sichern:

```
59999 end
60000 save "@0:go",8
60001 verify "go",8
```

Der Programmbenutzer kann nun, einmal das Startprogramm "go" eingeladen, wahlweise ins Generatorprogramm weitergehen und neue Daten eingeben oder aber sich diese oder zuvor eingegebene Kapitel über's Trainingsprogramm zur Bearbeitung vorlegen lassen. Es wäre nicht unelegant, wenn auch von den beiden Hauptprogrammen aus ein Rückgang in den Verteiler "go" möglich wäre, und es wäre nunmehr leicht, diesen Übergang ähnlich wie mit 10000/10150 des Zentralverteilers zu konstruieren. Es gibt aber eine noch charmantere und sparsamere Lösung, die uns endlich auch die noch mysteriöse Zeile 10120 des Generatorprogramms illustriert. Erinnerst Du Dich? Poke 44,8,:run? Um diese weitere grandiose Möglichkeit des C64 zu entdecken, müssen wir aber ein wenig ausholen.

Der Commodore 64 wurde so genannt, weil er über etwas mehr als 64000 Speicherstellen verfügt. Die 65536 Speicherstellen verteilen sich auf 256 **Seiten** von jeweils 256 Speicherstellen: 256*256=65536. Wenn wir alle Speicherstellen in einem Schema darstellen wollen, so erhalten wir des Schema der gegenüberliegenden Seite.
Die ersten 8 "Seiten" (Die Seiten 0 bis 7), die Speicherstellen 0 bis 2047 sind besetzt mit lebenswichtigen Funktionen des Rechners. Im Anschluß an die ersten 8 Seiten folgt der Bereich von 2048 bis 40959, in den wir unsere Programme schreiben. Die Differenz von 40959 und 2048 ergibt 38911 freie Byte. Dieser Wert wird vom Rechner kurz nach seinem Einschalten auf dem Bildschirm ausgegeben. Ein Programm, das eingeladen wird, wird von 2048 aufwärts im Rechnerspeicher abgelegt.
Wir könnten nun aus privaten Gründen wünschen, ein Programm beispielsweise erst oberhalb von 4096 (von Seite 16 an aufwärts) einladen zu lassen. Dazu müßten wir vor dem



Einladen des Programms eine Formel schreiben, auf deren genaue Funktion wir hier nicht eingehen:

poke 43,1:poke 44,(Seitenzahl):poke 256*(Seitenzahl),0:new

Auf Speicherstelle 4096 und somit Seite 16 bezogen ergibt sich folgende Zeile:

poke 43,1:poke 44,16:poke 256*16,0:new

Diese Formel bauen wir in unseren Zentralverteiler ein:

```
10150 poke 198,3
10160 poke 43,1:poke 44,16:poke 256*16,0:new
```

Nach dieser letzten Änderung geschieht folgendes:
Sobald wir den Rechner einschalten, load "go",8 schreiben und return geben, wird das Programm des Zentralverteilers beginnend bei Speicherstelle 2048 in den Rechnerspeicher geladen. Wir treffen unsere Wahl, dreimaliges return wird gespeichert (10140) und aktiviert (10150). Nicht mit "end" wie bisher wird die Automatik-Sequenz endgültig in Gang gesetzt, sondern durch "new" der neu geschriebenen Zeile 10160. Zuvor aber wurde die untere Programmgrenze von 2048 auf 4096 hochgesetzt. Der Anfang des Programms unserer Wahl, entweder "vokabel" oder "generator" kommt bei 4096 zu stehen. Dies bedeutet, daß nach dem Einladen des Folgeprogramms auch das zuerst eingeladene Programm "go" weiterhin im Rechnerspeicher steht. Während wir Daten eingeben oder unser Wissen prüfen lassen, ist der Zentralverteiler immer noch bei 2048ff vorhanden. Dorthin können wir, weil er nicht neu geladen werden muß, deshalb auch in Sekundenschnelle zurückkehren. Ein einfaches

```
poke 44,(Seitenzahl):run
```

startet das Programm bei Speicherstelle 2048ff (bei Seite 8). Vergleiche Zeile 10120 des Generatorprogramms...

Wir rufen das Trainingsprogramm auf und sehen auch dort die Rückkehr zum Zentralverteiler durch folgende zusätzliche Zeilen vor:

```
20045 ?"<3> Zum Zentralverteiler"
20075 if an$="3" then poke 44,8:run
```

24. Kapitel

Eine letzte Spielerei noch, bevor wir uns in den Ernst der Zeichensatzprogrammierung begeben:
Autostartprogramm und Blockierung von RUN/STOP, RESTORE-Taste und dem list-Befehl...

Wir rufen den Zentralverteiler auf und fügen folgende Programmzeilen bei 20000ff ein:

```
20000 gosub 10
20001 input "Autostartname";p$
20005 poke 792,192:poke 793,254
20006 poke 788,52
20010 ? chr$(147);
20020 ? "poke45,"peek(45)":poke46,"peek(46)":run"
20030 ?
20040 ? "poke631,19:poke632,13:poke198,2:poke43,198:
      poke44,0:";
20041 ? "save ";chr$(34);"@0:";p$;chr$(34);",8"
20050 end
```

Wir schreiben "goto 20000" und der Rechner löscht den Schirm und fragt nach dem Namen, unter dem unser Autostartprogramm abgespeichert werden soll. Dieser Name wird unter p$ zwischengespeichert.
In 20005 und 20006 blockieren wir zwei Tasten der Rechnertastatur:

1. in 20005 die RESTORE-Taste, indem wir in Speicherstelle 792 und 793 die Werte 193 bzw. 254 poken.
2. in 20006 die RUN/STOP-Taste, indem wir den Wert 52 in Speicherstelle 788 poken.

In 20010 bis 20040 schreiben wir 3 Zeilen auf den noch einmal gelöschten Schirm. Auf die genaue Funktionsweise der dort geschriebenen Befehle können wir nicht eingehen. Wir sehen aber in der letzten Zeile deutlich ein "save (neuer Dateiname)",8. Hinter diese Zeile gehen wir mit dem Cursor und geben return. Das Diskettenlaufwerk beginnt zu laufen und sichert das neue Programm. Nach kurzer Zeit erscheint die Nachricht "syntax error", das wir mit dem Ausschalten

des Rechners oder einem RESET beantworten.
Nehmen wir an, wir nannten die Autostartversion "gogo" und schreiben:

```
load "gogo",8
```

so erscheint nach einiger Zeit die fruchtlose Nachricht "ready", denn in der Tat ist der Rechner zu nichts bereit. Weder auf "run" noch auf "list" gibt er sinnvollen Äußerungen von sich. Erst wenn wir schreiben:

```
load "gogo",8,1
```

gehen die Dinge den gewollten Gang. Der Bildschirminhalt verschwindet und macht den 3 Zeilen Platz, die wir durch "goto 20000" aus dem Zentralverteiler heraus erzeugt hatten. Dann verschwinden auch diese wieder, und der Rechner bringt die Wahl zwischen Trainingsprogramm und Generatorprogramm. Das Programm hat sich selber gestartet, ohne daß wir zuvor "run" eingeben mußten! Da weder RUN/STOP- noch RESTORE-Taste reagieren, kann niemand mehr die Programme listen und unsere Programmierkünste bewundern!

Eine kleine Änderung noch ist im Zentralverteiler jedoch vonnöten, weil durch poke 788,52 auch die Rechneruhr blockiert wird, die wir im Trainingsprogramm zur Ermittlung der Punktzahl noch brauchen. Wir entblocken die RUN/STOP-Taste durch poke 788,49, nicht aber ohne sie zuvor mit poke 808,234 auf eine andere Art neu blockiert zu haben:

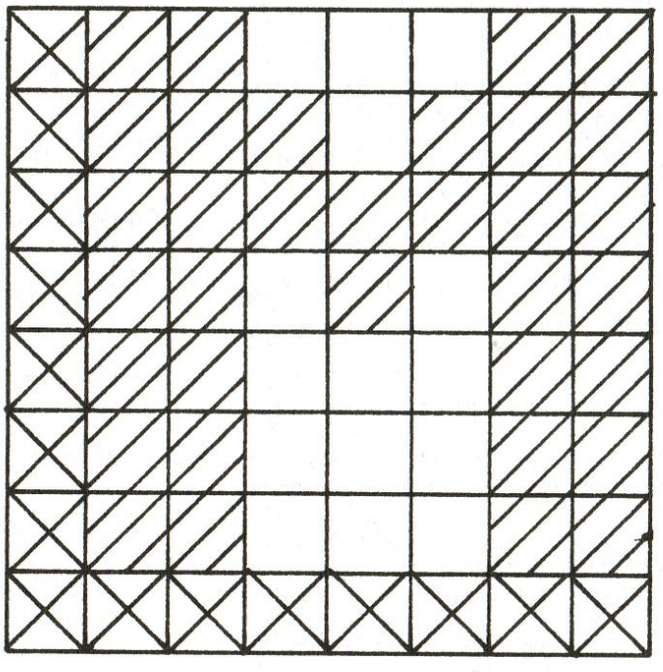
```
10007 poke 808,234:poke 788,49
```

Niemand mehr wird das Programm weder stoppen noch listen noch restoren...

25. Kapitel

Wir lernen in diesem Kapitel, wie der Rechner Buchstaben kontruiert und begegnen der 8x8-Matrix. Schließlich kopieren wir den Zeichensatz des Commodore in einen für ein Basicprogramm erreichbaren Bereich.

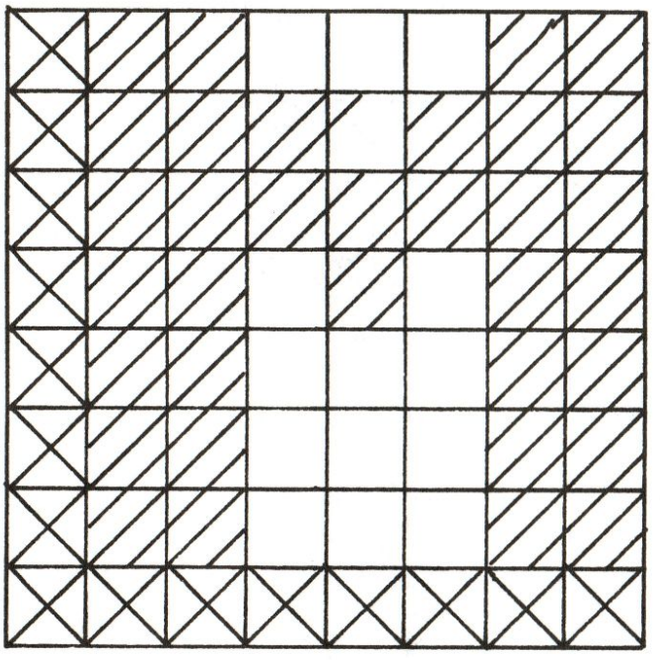
Jeder Buchstabe, der vom Rechner auf den Schirm projiziert wird, besteht aus einer Vielzahl von kleinen Bildschirmpunkten. Diese Punkte sind auf ein Quadrat von 8x8 Punkten verteilt.



Durch die Verteilung von aktivierten und nichtaktivierten Einzelpunkten ergibt sich schließlich entsteht schließlich der Buchstabe. Beachte, daß sowohl die linke Spalte als auch die letzte Zeile des Quadrats in der Regel frei bleiben, damit die einzelnen Buchstaben nicht an- oder aufeinandergeklebt erscheinen! Beachte weiterhin, daß ein Punkt nur dann deutlich sichtbar auf dem Schirm erscheint, wenn er in der Horizontalen aus mindestens 2 Matrixpunkten besteht.

Wie aber geben wir dem Rechner zu verstehen, welche Einzelpunkte zu aktivieren sind und welche nicht? Über jeden Einzelpunkt gesondert Buch zu führen, wäre eine zu umfangreiche Arbeit. Stattdessen wird nur für jede einzelne Matrixzeile ein Zahlenwert bestimmt, durch den dann festgelegt ist, welche Einzelpunkte zu aktivieren sind. Dieser **Zahlenwert** errechnet sich, indem wir aus unserer Matrixzeile die Spaltenwerte aller Kästchen, die zu aktivieren sind, zusammenzählen und getrennt notieren. Für den Buchstaben M erhalten wir folgende Werte:

128 64 32 16 8 4 2 1



64+32+2+1	= 99
64+32+16+4+2+1	= 119
64+32+16+8+4+2+1	= 127
64+32+8+2+1	= 107
64+32+2+1	= 99
64+32+2+1	= 99
64+32+2+1	= 99
	= 0

Mit 8 Zahlenwerten ist ein Buchstabe definiert!!

****

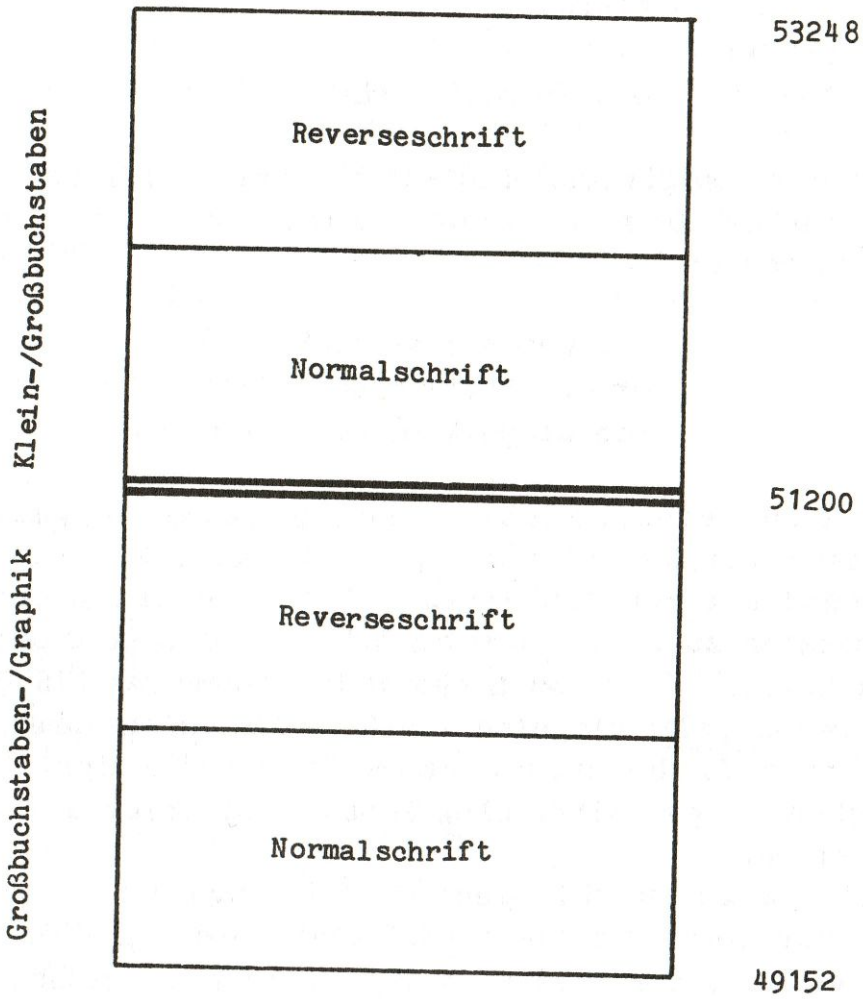
Auf der Tastatur unseres Rechners sind mehr Buchstaben gleichzeitig gespeichert als es auf den ersten Blick scheint. Wir schalten nach Einschalten des Rechers durch gleichzeitiges Betätigen der SHIFT-Taste + Commodore-Taste auf den Groß-/Kleinschreibmodus um und sehen nach, wieviel Buchstaben allein über die A-Taste erreichbar sind. Wir haben:

1.: a kleines a
2.: Shift+a großes A
3.: CO+a ein Graphikzeichen

Wenn wir nun gleichzeitg auf die Taste CTRL und 9 drücken, erscheint das gleiche Repertoire noch einmal im Reversemodus:

4.: kleines a (reverse)
5.: großes A (reverse)
6.: ein Graphikzeichen (reverse)

Auf eine einzige Taste fallen also 6 verschiedene Buchstaben! Das gleiche gilt für alle Buchstabentasten, während auf den Zahlentasten nur jeweils 4 verschiedene Buchstaben zu liegen kommen. Wir können über unsere Tastatur gleichzeitig fast 256 Buchstaben ansteuern. 256 Buchstaben, von denen jeder einzelne mit 8 Zahlen (siehe oben) definiert werden muß, bedeutet, daß wir für die Speicherung des Zeichensatzes 256*8, also 2048 Speicherstellen im Rechner benötigen.
Damit aber nicht genug! Wie bekannt, kann durch gleichzeitiges Betätigen der SHIFT- und der Commodore-Taste von einem Schreibmodus in den anderen umgeschaltet werden. Im Großbuchstaben-/Graphikmodus, der nach dem Einschalten des Rechners eingestellt ist, liegt auf jeder Taste nur ein (Groß-)Buchstabe, dafür aber 2 verschiedene Graphikzeichen. Für diesen Schriftmodus, ist noch einmal jeder Buchstabe für sich im Rechnerspeicher definiert, und es werden daher ein zweites Mal 256*8 Speicherstellen benötigt. In unseren Rechner sind also gleichzeitig 512 Buchstaben gespeichert; jeweils 256 Buchstaben sind gleichzeitig erreichbar, durch SHIFT/CO wird auf die jeweils anderen 256 zugegriffen.
Wenn jeder Buchstabe aus 8 Zahlen definiert ist, so benötigen wir 512*8, also 4096 Zahlen (= Speicherstellen), um unsere beiden Zeichensätze zu konstruieren.



Nach dem Einschalten des Rechners sind die Zeichensätze in den 4096 Speicherstellen von 53248 an aufwärts lokalisiert. Dort sind sie jedoch nicht dem direkten Zugriff von einem Basicprogramm aus zugänglich. Um die Zahlen und damit das Aussehen unserer Buchstaben verändern zu können, müssen wir die Zeichensätze zuerst in einen zugänglichen Bereich kopieren. Dieser zugängliche Bereich beginnt bei 49152 (eine ganz wichtige Zahl!!) und endet bei 53247. Es handelt sich dort um einen für den Programmierer freien Bereich, der aber außerhalb der Basiczone liegt und von Basicprogrammen nicht beeinträchtigt wird. In diesen freien Bereich von insgesamt 4 k (was genau dem Umfang unserer Zeichensätze entspricht) kopieren wir unsere Daten. Der Kopiervorgang dauert etwa 50 Sekunden:

```
1 goto 10000
10 ? chr$(147):return
20 poke 211,x:poke 214,y:sys 58640:return
90 poke 781,y:sys 59903:return
99 poke 53280,0:poke 53281,0:poke 646,1:return
10000 rem ********** "Zeichensatzveränderung" *******
10010 gosub 99:gosub 10:? chr$(14)
10020 x=8:y=10:gosub 20
10030 ? "Es soll bearbeitet werden:"
10040 y=12:gosub 20
10050 ? "<1> Originalzeichensatz"
10060 y=13:gosub 20:? "<2> Diskettenzeichensatz
10070 get a$
10080 if a$="1" then 10200
10090 if a$="2" then 10500
10100 goto 10070
10200 rem *** Rechnerzeichensatz nach 49152ff ***
10210 gosub 10
10220 x=10:y=11:gosub 20
10230 ? "50 Sekunden warten!"
10250 poke 56334,0:poke 1,51
10260 for i=0 to 4095
10270 poke 49152+i,peek(53248+i)
10280 next
10290 poke 1,55:poke 56334,1
10300 goto 20000
```

Wir fragen den Benutzer, ob er den Orginialzeichensatz bearbeiten will, also jenen Zeichensatz, der nach dem Einschalten des Geräts bereit liegt, oder ob wir einen "Diskettenzeichensatz" verändern wollen. Diese letzte Wahl, die im Programm nach 10500ff weiterführt, lassen wir zunächst außer acht, weil auf der Diskette bislang noch kein alternativer Zeichensatz gespeichert ist. Wir wählen mit 1 das Kopieren des Originalsatzes aus 53248ff nach 49152ff. Auf die komplexen Speicherstellen 1 und 56334 , die in 10250 und 10290 verändert werden, können wir hier nicht eingehen.
In 20000ff fahren wir fort mit der Frage, ob der Großbuchstaben-/Graphikzeichensatz (der, der beim

Einschalten des Rechners aktiv ist), oder der Klein-/Großbuchstabenzeichensatz (jener, in den man mit SHIFT/CO überwechselt) verändert werden soll:

```
20000 rem *****  "Groß/Klein"  *****
20010 gosub 10
20020 x=8:y=2:gosub 20
20030 ? "Es soll veraendert werden:"
20040 x=6:y=11:gosub 20
20050 ? "<1> Grossbuchstaben/Graphik"
20060 y=12:gosub 20:? "<2> Gross-/Kleinbuchstaben
20070 get a$
20080 if a$="1" then bb=49152:goto 22000
20090 if a$="2" then bb=51200:goto 22000
20100 goto 20070
```

Je nachdem, ob wir den ersten oder den zweiten Zeichensatzverändern wollen, wird für den Speicherbereich, der nachher für uns interessant wird, eine unterschiedliche Startadresse in bb abgelegt. Während der Graphik-Zeichensatz bei 49152 beginnt, liegt der Anfang des Schrift-Zeichensatzes bei 51200. Nun wird's kompliziert:

```
22000 rem ***** "Einblenden" ******
22010 gosub 10
22020 y=5:x=2:gosub 20
22030 input "Kennziffer eingeben (666=saven)";k
22040 if k=666 then 40000
22050 k=k-256
22060 if k<0 or k>256 then 22000
22070 s=bb+k*8
22100 for i=0 to 7
22110 x=3:y=i+8:gosub 20
22120 ? i;s+i;peek(s+i);
22130 p=peek(s+i)
22140 x=25:gosub 20
22150 for j=0 to 7
22160 if p>=2^(7-j) then ? "*";:p>=p-2^(7-j):goto 22180
22170 ? ".";
22180 next j
```

```
22190 next i
22200 get a$:if a$="" then 22200
22210 goto 22000
```

Wir fragen nach der Kennziffer des Buchstabens, der verändert werden soll und legen ihn bei k ab. Bei k=333 geht der Rechner nach 40000 weiter (siehe unten), wo der veränderte Zeichensatz unter einem neuen Namen auf Diskette geschreiben wird. In 22050 verringern wir k um 256, weil die Kennziffern unser folgenden Tabelle die Werte über 256 enthält. Aus bb und k wird in 22070 danach die erste der 8 Speicherstellen als s (Start) definiert, die den gewählten Buchstaben konstruieren. Die folgende i-Schleife 22100/22190 bringt die Speicherstelle (s+i in 22120), die dort gespeicherte Zahl (peek(s+i) aus 22120) und eine Kennziffer (i) auf den Bildschirm. Nun wird der in s+i abgelegte Zahlenwert nach p zwischengespeichert (22130), ein neuer x-Wert festgelegt und schließlich in der j-Schleife dieser Wert p analysiert und danach eine Reihe von 8 Symbolen auf den Bildschirm in die gleiche Reihe projiziert, die angeben, welches der Kästchen besetzt ist (der Kreis) und somit später auf dem Bildschirm einen Leuchtpunkt erzeugt, und welches Kästchen hingegen leer (als Punkt dargestellt) ist und schwarz bleibt. Auf die genaue Funktionsweise der Kernformel der j-Schleife in 22160 können wir hier aus Platzgründen nicht eingehen. Beachte unbedingt das Semikolon (;) in den Zeilen 22120, 22160 und 22170, ohne das das Programm nicht zufriedenstellend arbeitet!!

Bleibt nur noch die Frage, mit welcher Kennzahl wir zu welchem Buchstaben gelangen. In der nachfolgenden Tabelle 1 sind die Kennzahlen aller Buchstaben angegeben. Siehe dazu Tabelle 1 auf der nächsten Seite.

Tabelle 1:
Kennzahlen für die Zeichen auf der Commodore-Tastatur

	NORMSCHRIFT			REVERSESCHRIFT		
	Normal	SHIFT	CO	Normal	SHIFT	CO
a	257	321	368	385	449	496
b	258	322	383	386	450	511
c	259	323	380	387	451	508
d	260	324	364	388	452	492
e	261	325	369	389	453	497
f	262	326	379	390	454	507
g	263	327	357	391	455	485
h	264	328	372	392	456	500
i	265	329	354	393	457	482
j	266	330	373	394	458	501
k	267	331	353	395	459	481
l	268	332	374	396	460	502
m	269	333	359	397	461	487
n	270	334	362	398	462	490
o	271	335	377	399	463	505
p	272	336	367	400	464	495
q	273	337	363	401	465	491
r	274	338	370	402	466	498
s	275	339	366	403	467	494
t	276	340	355	404	468	483
u	277	341	376	405	469	504
v	278	342	382	406	470	510
w	279	343	371	407	471	499
x	280	344	381	408	472	509
y	281	345	375	409	473	503
z	282	346	365	410	474	493
0	304	-	-	432	-	-
1/!	305	289	-	433	417	-
2/"	306	290	-	434	418	-
3/#	307	291	-	435	419	-
4/$	308	292	-	436	420	-

5/%	309	293	-	437	421	-
6/	310	294	-	438	422	-
7/'	311	295	-	439	423	-
8/(	312	296	-	440	424	-
9/)	313	297	-	441	425	-
:/<	314	283	-	442	411	-
;/>	315	285	-	443	413	-
=	317	-	-	445	-	-
,/<	300	316	-	428	444	-
/>	302	318	-	430	446	-
//?	303	319	-	431	447	-
**	256	378	356	384	506	484
*	298	320	351	426	448	479
Pfeil h.	286	350	350	414	478	478
Pfeil l.	287	287	387	415	415	415
+	299	347	358	427	475	486
-	301	349	348	429	477	476
Pfund	284	361	360	412	489	488

Wir können nun jeden Buchstaben des Zeichensatzes einblenden und die Leertaste aus 22200 fragt nach dem nächsten Buchstaben. Im nächsten Kapitel wird verändert...

26. Kapitel

Wir lernen in diesem Kapitel, einzelne Zeichen teilweise oder ganz zu verändern. In einer ausführlichen Tabelle werden die Zahlenwerte aufgeführt, die zur Konstruktion der Sonderzeichen der großen westeuropäischen Sprachen benötigt werden.

Wir streichen die Zeilen 22200f und schreiben die Fortsetzung:

```
22200 x=3:y=20:gosub 20
22210 ? "<1> Löschen und verändern"
22220 y=21:gosub 20:? "<2> Verändern"
22230 y=22:gosub 20:? "<3> Keine Veränderung"
22240 get a$
22250 if a$="3" then 22000
22260 if a$="1" then gosub 150:goto 23000
22270 if a$="2" then 23000
22280 goto 22240
```

Wahl 1 soll heißen, daß alle Sternchen in Punkte verwandelt werden, damit wir eine freie Matrix haben, in die wir unser neues Zeichen einbauen können. Wahl 2 sagt, daß wir verändern wollen, ohne daß das bisherige Zeichen gelöscht wird. Mit Wahl 3 geht der Rechner nach 22000 zurück und fragt nach einer neuen Kennziffer. Bei 150ff programmieren wir ein Unterprogramm zum Löschen des augenblicklich eingespeicherten Zeichens:

```
150 rem ***** "Matrix löschen" *****
155 x=25
160 h$="********"
165 for y=8 to 15
170 gosub 20:? h$
175 next
180 x=3:return
```

Wir erinnnern uns, daß wir oben im Programm den x-Wert für unsere Matrix bei 25 festgelegt hatten. Wir erinnern uns weiter, daß insgesamt 8 Zeilen geschrieben wurden, und zwar von Zeile 8 bis Zeile 15. In 150ff definieren wir x wiederum als 25, konstruieren einen String h$, der aus 8 aneinandergereihten Sternchen besteht (160) und programmieren eine y-Schleife, die von 8 bis 15 hochzählt. Mit x- und y-Wert geht der Rechner in das Lokalisationsunterprogramm bei 20 und schreibt insgesamt achtmal den String h$. Es entsteht auf dem Bildschirm ein 8x8-Quadrat aus Sternchen.

Nachdem der Rechner von 150ff zurückgekehrt ist, geht er weiter nach 23000ff in die eigentliche Passage der Zeichenveränderung:

```
23000 rem ***** "Einzelne Zeilen verändern" *****
23010 gosub 50

(50 for y=20 to 22:gosub 90:next:return)

23020 y=21:gosub 20:input "Zeilennummer (8=Ende)";z
23030 if z=8 then 22000
23040 if z<0 or z>7 then 23000
23050 y=22:gosub 20:input "Neuer Zeilenwert";w
23060 if w<0 or w>255 then gosub 90:goto 23050
23070 y=z+8:gosub 90
23080 poke s+z,w
23090 x=3:gosub 20
23100 ? z;s+z;peek(s+z);
23110 x=25:gosub 20
23120 for j=0 to 7
23130 if w=>2^(7-j) then ? "*";:w=w-2^(7-j):goto 23150
23140 ? ".";
23150 next
23160 x=3:goto 23000
```

Der Rechner löscht mit 23010 nebst 50 die Bildschirmzeilen 20 bis 22 und fragt in 23020 nach der Nummer der Zeile, die verändert werden soll. Wir geben eine Zahl zwischen 0 und 7

ein (entsprechend der ersten Zahl in jeder der 8 Bildschirmreihen der Matrix). Nach zwei Prüfzeilen, die ausschließen, daß unzulässige Werte eingegeben werden und daß bei z=8 der Veränderungsvorgang abgebrochen wird, fragt der Rechner aus 23050 nach dem neuen Zeilenwert. Wieder werden unzulässige Werte in 23060 ausgeschlossen. Danach errechnet der Rechner mit y=z+8 die Bildschirmzeile, die gelöscht werden muß, pokt den neuen Zeilenwert in die zugehörige Speicherstelle s+z (s war der erste der 8 Werte, der den eingeblendeten Buchstaben definierte) und schreibt die gelöschte Zeile mit dem neuen Wert w um. Zunächst in 23090/23100 Zeilennummer, Speicherstelle und Wert der Speicherstelle, schließlich die Analyse und Aufschlüsselung in Punkte und Sternchen des w-Wertes in 23110/23150. Aus 23160 geht der Rechner zurück nach 23000 und fragt erneut nach einer zu verändernden Zeile.

Selber neue Buchstaben zu konstruieren, ist zwar leicht, denn wir brauchen die Figur nur auf eine 8x8-Matrix zu verteilen; dennoch ist es eine langwierige Arbeit, und es ist nicht nötig, die Arbeit zu wiederholen, die andere vor uns gemacht haben. In der nachstehenden Tabelle 2 geben wir daher eine Übersicht über die Buchstaben der europäischen Alphabete und die 8 Zahlenwerte, aus denen sie sich konstruieren lassen.

Tabelle 2:
Matrixwerte für die Commodore-Zeichen

a	0	0	60		62	102	62	0
b	0	96	96	124	102	102	124	0
c	0	0	60	96	96	96	60	0
d	0	6	6	62	102	102	62	0
e	0	0	60	102	126	96	60	0
f	0	14	24	62	24	24	24	0
g	0	0	62	102	102	62	6	124
h	0	96	96	124	102	102	102	0
i	0	24	0	56	24	24	60	0
j	0	6	0	6	6	6	6	60
k	0	96	96	108	120	108	102	0
l	0	56	24	24	24	24	60	0
m	0	0	102	127	127	107	99	0
n	0	0	124	102	102	102	102	0
o	0	0	60	102	102	102	60	0
p	0	0	124	102	102	124	96	96
q	0	0	62	102	102	62	6	6
r	0	0	124	102	96	96	96	0
s	0	0	62	96	60	6	124	0
t	0	24	126	24	24	24	14	0
u	0	0	102	102	102	102	62	0
v	0	0	102	102	102	60	24	0
w	0	0	99	107	127	62	54	0
x	0	0	102	60	24	60	102	0
y	0	0	102	102	102	62	12	120
z	0	0	126	12	24	48	126	0
A	24	60	120	124	102	102	102	0
B	124	120	120	124	102	102	124	0
C	60	102	96	96	96	102	124	0
D	120	108	102	102	102	108	120	0
E	126	96	96	120	96	96	126	0
F	126	96	96	120	96	96	96	0
G	60	102	96	110	102	102	60	0
H	102	102	102	126	102	102	102	0
I	60	24	24	24	24	24	60	0

J	30	12	12	12	12	108	56	0
K	102	108	120	112	120	108	102	0
L	96	96	96	96	96	96	126	0
M	99	119	127	107	99	99	99	0
N	102	118	126	126	110	102	102	0
O	60	102	102	102	102	102	60	0
P	124	102	102	124	96	96	96	0
Q	60	102	102	102	102	60	14	0
R	124	102	102	124	120	108	102	0
S	60	102	96	60	6	102	60	0
T	126	24	24	24	24	24	24	0
U	102	102	102	102	102	102	60	0
V	102	102	102	102	102	60	24	0
W	99	99	99	107	127	119	99	0
X	102	102	60	24	60	102	102	0
Y	102	102	102	60	24	24	24	0
Z	126	6	12	24	48	96	126	0
1	24	24	56	24	24	24	126	0
2	60	102	6	12	48	96	126	0
3	60	102	6	28	6	102	60	0
4	6	14	30	102	127	6	6	0
5	126	96	124	6	6	102	60	0
6	60	102	96	124	102	102	60	0
7	126	102	12	24	24	24	24	0
8	60	102	102	60	102	102	60	0
9	60	102	102	62	6	102	60	0
0	60	102	110	118	102	102	60	0
!	24	24	24	24	0	24	24	0
"	102	102	102	0	0	0	0	0
#	102	102	255	102	255	102	102	0
$	24	62	96	60	6	124	24	0
%	98	102	12	24	48	102	70	0
	60	102	60	56	103	102	63	0
'	6	12	24	0	0	0	0	0
(	12	24	48	48	48	24	12	0
)	48	24	12	12	12	24	48	0
Pfeil l.	0	16	48	127	127	48	16	0
Pfeil h.	0	24	60	126	24	24	24	0

Pfund	12	18	48	124	48	98	252	0
(	60	48	48	48	48	48	60	0
)	60	12	12	12	12	12	60	0
*	0	102	60	255	60	102	0	0
+	0	24	24	126	24	24	0	0
-	0	0	0	126	0	0	0	0
/	0	3	6	12	24	48	96	0
=	0	0	24	0	0	24	0	0
<	14	24	48	96	48	24	14	0
>	112	24	12	6	12	24	112	0
,	0	0	0	0	0	24	24	48
;	0	0	24	0	0	24	24	48
:	0	0	24	0	0	24	0	0
?	60	102	6	12	24	0	24	0
Kl.affe	60	102	110	110	96	98	60	0

Tabelle 3:
Matrixwerte für europäische Sonderzeichen

									Kennzahl
Deutsch:									
ä	0	102	0	60	102	126	102	0	315
Ä	102	24	60	102	126	102	102	0	285
ö	0	102	0	60	102	102	60	0	283
Ö	102	60	102	102	102	102	60	0	?
ü	0	102	0	102	102	102	60	0	256
Ü	102	0	102	102	102	102	60	0	378
ß	30	51	62	51	62	48	48	0	299
Französisch:									
a^	24	60	60	6	62	102	62	0	287
a`	24	12	60	6	62	102	62	0	289
e^	24	60	60	102	126	96	60	0	290
e'	12	24	60	102	126	96	60	0	291
e`	24	12	60	102	126	96	60	0	292

c,	0	0	60	96	96	108	60	48	293
oe	0	0	118	219	223	220	119	0	294
u^	24	60	0	102	102	102	60	0	295
i^	24	60	0	56	24	24	60	0	296
o^	24	60	6	60	102	102	60	0	297
Spanisch:									
a'	12	24	60	6	62	102	62	0	301
e'	siehe Französisch								
i'	12	24	0	56	24	24	60	0	284
o'	12	24	60	102	102	102	60	0	298
u'	12	24	102	102	102	102	60	0	286
n(Tilde)	59	110	0	124	102	102	24	0	362
Sardisch/Italienisch									
a`	siehe Französisch								
e`	siehe Französisch								
i`	24	12	0	56	24	24	60	0	361
o`	24	12	60	102	102	102	60	0	320
u`	24	12	102	102	102	102	60	0	350

27. Kapitel

Im letzten Kapitel speichern wir den veränderten Zeichensatz auf die Diskette und rufen ihn von anderen Programmen aus wieder in den Rechnerhauptspeicher. Als letzte Aufgabe bleibt dann die Umstellung des Bildschirms auf den neuen Zeichensatz.

Wir fahren bei 40000 ff fort mit:

```
40000 rem  *** "Abspeichern" ***
40010 gosub 10
40020 x=3:y=11:gosub 20:input "Zeichensatzname";n$
40030 if len(n$)>14 then 40000
40040 h$=chr$(34)
40050 x=0:y=3:gosub 20
40060 ? "sys(57812)"h$"@0:"n$h$",8:poke193,0:poke194,192:";
40070 ? "poke174,0:poke175,208:sys 62957
40080 ? chr$(19)
40090 for i=631 to 633:poke i,13:next:poke 198,3:end
```

Der Rechner fragt nach einem Namen für den neuen Zeichensatz und speichert ihn auf die Diskette.

Von einem beliebigen Programm aus können wir den Zeichensatz wieder einladen. Haben wir unserem Zeichensatz den Namen "helmut" gegeben, so müssen wir schreiben:

```
1 if d=1 then 10000
2 d=1
3 load "helmut",8,1
10 ? chr$(147)
10000 ff (weiteres Programm)
```

Nachdem der Befehl aus Zeile 3 ausgeführt wurde, kehrt der Rechner an den Anfang des Programms zurück. Dort wird ihm, die Variable d wurde durch die Operation nicht auf Null gestellt, gesagt, er solle weiter nach 10000 gehen.

Damit der neue Zeichensatz auch auf dem Bildschirm ausgezeichnet wird, die letzte Veränderung:

```
130 gosub 10
131 poke 53272,peek(53272) and 240 or 2
132 poke 56576,0:poke 648,196
133 gosub 10:return

140 gosub 10
141 poke 53272,peek(53272) and 240 or 4
142 poke 56576,3:poke 648,4
143 gosub 10:return
```

Auf die genaue Funktionsweise der letzten Passagen kann nicht eingegangen werden.
Merke also: ein an beliebiger Stelle des Programmes plaziertes "gosub 130" stellt die Bildschirmverwaltung auf den neuen Zeichensatz um. Ein "gosub 140" stellt den ursprünglichen Zustand wieder her.

Viel Spaß!

”Teil C“

Langzeitergebnisse im computer-assistierten Vokabeltraining

Abstract

Wir untersuchten bei einer 30-jährigen Versuchsperson die Lernzeiten für 500 neugriechische Wörter. Nach 9 auf zehn Arbeitstage verteilte Arbeitsstunden wurde der Wortschatz zu 97% beherrscht. Die letzte, über alle 500 Wörter gehende Abschlußprüfung dauerte 38 Minuten.
Nach 5 Monaten überprüften wir die Überlebensquote des Gelernten. Am ersten Tag lag ohne vorherige Vokabelwiederholung der Prozentsatz der richtigen Antworten bei 11%. Bei vorheriger Einsicht der Vokabellisten in der Dauer von 10 Minuten pro 100 Vokabeln erreichte der Proband 86% richtige Antworten. Am 4. Tag dauerte die abschließende Prüfung über den gesamten Stoff 42 Minuten bei einer Trefferquote von 94%.
In 4.1 auf fünf Tage verteilte Arbeitsstunden wurde ein 5 Monate altes unterschwellig vorhandenes Wissen auf das Ausgangsniveau reaktiviert. Die Reaktivationszeit betrug 46% der Dauer der initialen Lernphase. Wir folgern, daß Computer-gesteuerte Lernsequenzen sowohl in der initialen Stoffbearbeitung als auch während einer späteren Wiederaufbereitungsphase herkömmliche Lernmethoden an Wirksamkeit übertreffen.

Vorbemerkungen

Von früheren Beobachtungen und Versuchen war bekannt, daß ein mit Hilfe von Computerlernprogrammen bearbeiteter Stoff in überraschend kurzer Zeit beherrscht werden konnte. Beeindruckend war vor allem die Erfahrung mit einer 24-jährigen Testperson, die nach 3 Wochen intensiven Vokabel- und Konjugationstrainings und nur wenigen Grammatikübungen in der Lage war, sich problemlos verständlich zu machen und gängige Zeitungen zu verstehen. Eben diese letzte Aussage war aber vage und konnte zu Mißverständnissen Anlaß geben, zudem war nichts gesagt über die Langzeiterfolge der Methode. Wir nahmen uns daher vor, in einem neuen Versuch die Lernzeiten und Lernerfolge quantitativ zu erfassen und außerdem nach einer Latenzzeit von mehreren Monaten die Überreste des so gelernten Stoffes zu überprüfen.

Methoden

Wir wählten eine 30-jährige Versuchsperson mit guten Maschinenschreibfähigkeiten (400 blinde Anschläge pro Minute). Prüfungsstoff war Neugriechisch in lateinischer Umschrift. Der Proband hatte zuvor weder Alt- noch Neugriechisch gelernt, so daß von der Morphologie her ein submaximaler Schwierigkeitsgrad gewährleistet war. 500 Wörter wurden willkürlich auf 5 Kapitel verteilt. In der ersten Phase lernte der Proband täglich 100 Wörter mit einem Steinhäuser-Lernprogramm (Version 8408) auf Toshiba T-100 Personalcomputer. Nach einer kurzen Vorbereitungszeit mit Einsicht in die Wortliste wurden die Prüfungswörter in zufälliger Folge zur Übersetzung angeboten. Nach zwei fehlerhaften Versuchen blendete der Rechner die richtige Antwort zur Abschrift ein. Am Ende der Prüfung mußte jeder Fehler ein weiteres Mal beantwortet werden.
Jedes Kapitel wurde zweimal hintereinander bearbeitet. Im ersten Durchgang wurden die 100 Wörter gedrittelt (33 bzw. 34 Wörter) und jedes Drittel in einer Prüfungsform dargeboten, die jedes Wort als Frage immer wieder

einblendete, bis es insgesamt zweimal korrekt beantwortet worden war. Nach Abschluß der letzten Drittels wurden in der Abschlußprüfung alle 100 Wörter noch einmal zur Beantwortung vorgestellt. Jedes Wort erschien hier nur einmal (Einfache Prüfung). Ein auf diese Weise bearbeitetes Kapitel wurde im Durchschnitt nach knapp einer Stunde abgeschlossen.

In der zweiten Phase (5 Tage der darauffolgenden Woche) bearbeitete der Proband täglich die zuvor gelernten 5 Kapitel. Einsicht in die Vokabellisten wurde nicht gewährt. Der Stoff wurde als einfache Prüfung studiert, jede Frage erschien also nur einmal zur Beantwortung, falsche Antworten wurden im Anschluß an die Prüfung noch einmal vorgelegt. Prozentsatz der richtigen Antworten und benötigte Prüfungszeit wurden registriert.
In der dritten Phase (nach 5 Monaten) wurde der Stoff erneut überprüft, nachdem in der Zwischenzeit der Proband sich des Neugriechischen peinlich enthalten hatte. Die 5 Kapitel wurden 4 Tage lang in einfacher Prüfung wiederholt. 2 Kapitel (Gruppe A; Kap. 1,2) wurden ohne Einsicht der Wortlisten bearbeitet, für die übrigen Kapitel (Gruppe B; Kap. 3,4,5) war lediglich am ersten Tag eine Vorbereitungszeit von 10/Minuten pro 100 Wörter und am zweiten Tag von 5 Minuten pro 100 Wörter zugestanden. Prozentsatz der richtigen Antworten und benötigte Prüfungszeit wurden registriert.

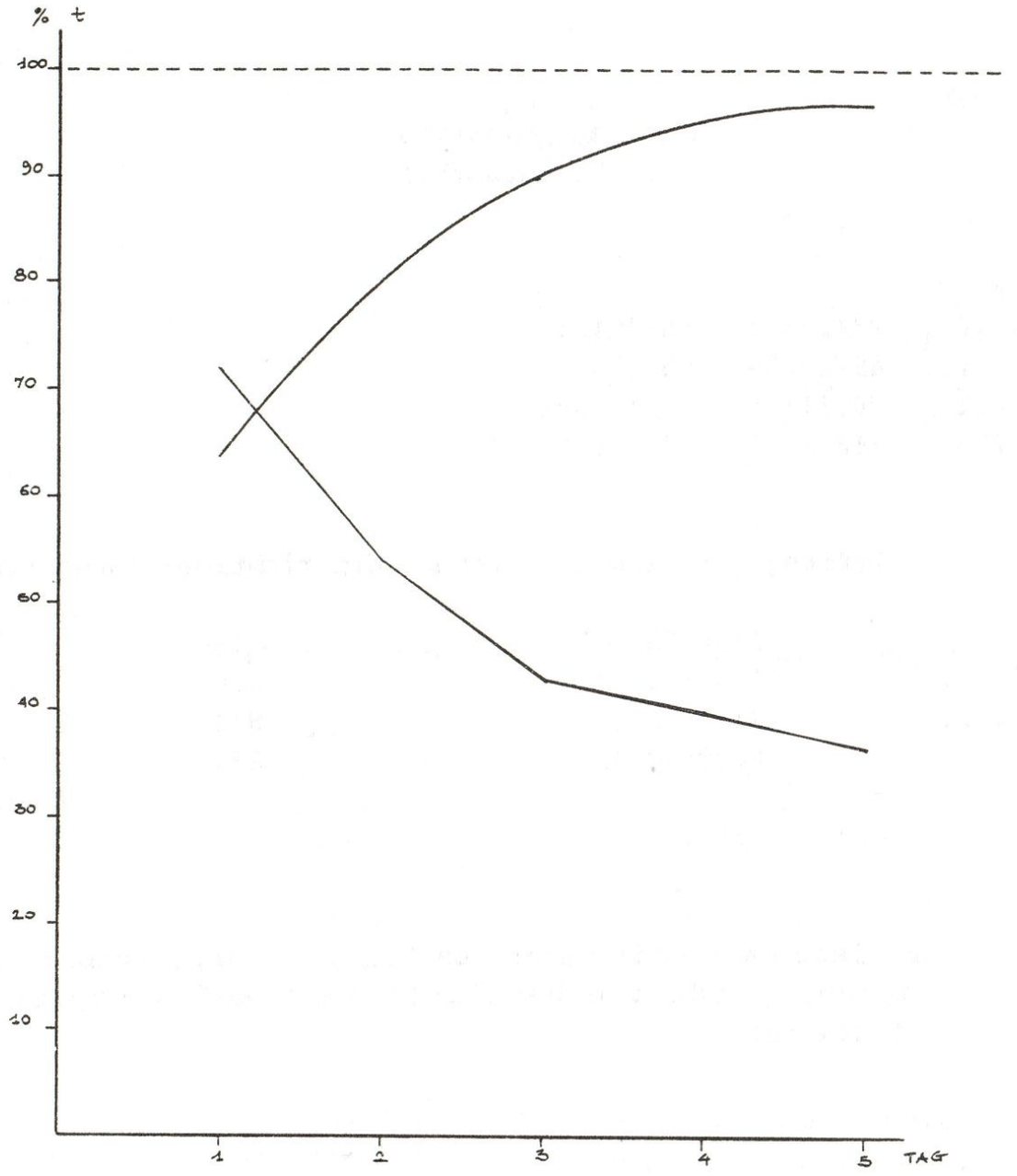
Resultate

Tabelle 1: Anzahl der falschen Antworten und Prüfungszeit in Phase 2

	Kapitelnummer falsche Antworten/Zeit				
	1	2	3	4	5
Datum					
7.10.	36/13:25	47/17:35	15/ 9:49	34/12:37	53/18:24
8.10.	19/10:49	22/12:50	12/ 8:15	8/ 8:43	18/13:35
9.10.	8/ 8:18	12/10:17	3/ 7:01	7/ 7:10	16/10:16
10.10.	8/ 8:14	2/ 8:23	3/ 6:44	3/ 6:24	8/ 9:49
11.10.	5/ 7:04	2/ 7:30	- (*)	2/ 7:22	3/ 7:44

	Prüfungszeit insg.	Prozentsatz richtiger Antworten
7.10.	1:11:50	63.9
8.10.	0:54:12	80.5
9.10.	0:43:02	90.8
10.10.	0:39:43	95.2
11.10.	0:36:24	97.0
Summe:	4:05:11	

(*) Dem Probanden war freigestellt, bei weniger als 5 falschen Antworten das jeweilige Kapitel abzuschließen. Für die Gesamtwertung wurde für Kapitel 3 am 11.10. die Werte des Vortages berechnet.



Graphik 1: Prüfungszeit und Prozentsatz richtiger Antworten in Phase2

Tabelle 2: Anzahl der falschen Antworten und Prüfungszeiten nach 5-monatiger Abstinenz (Phase 3)

Gruppe A

	Kapitelnummer falsche Antworten/Zeit	
	1	2
Datum		
20.2.	83/27:01	95/36:23
26.2.	49/15:38	55/17:55
28.2.	20/11:35	13/11:08
2.3.	16/ 9:47	8/ 9:26

	Prüfungszeit insg.	Prozentsatz richtiger Antworten
20.2.	1:34:23 (*)	11%
26.2.	0:50:19 (*)	48%
28.2.	0:34:05 (*)	83%
2.3.	0:28:50 (*)	88%
Summe:	3:27:37 (*)	

(*) Um diese Werte mit denen aus Gruppe B vergleichbar zu machen, wurde die benötigte Zeit auf 3 Kapitel aufgerundet.

Tabelle 2 (Fortsetzung)

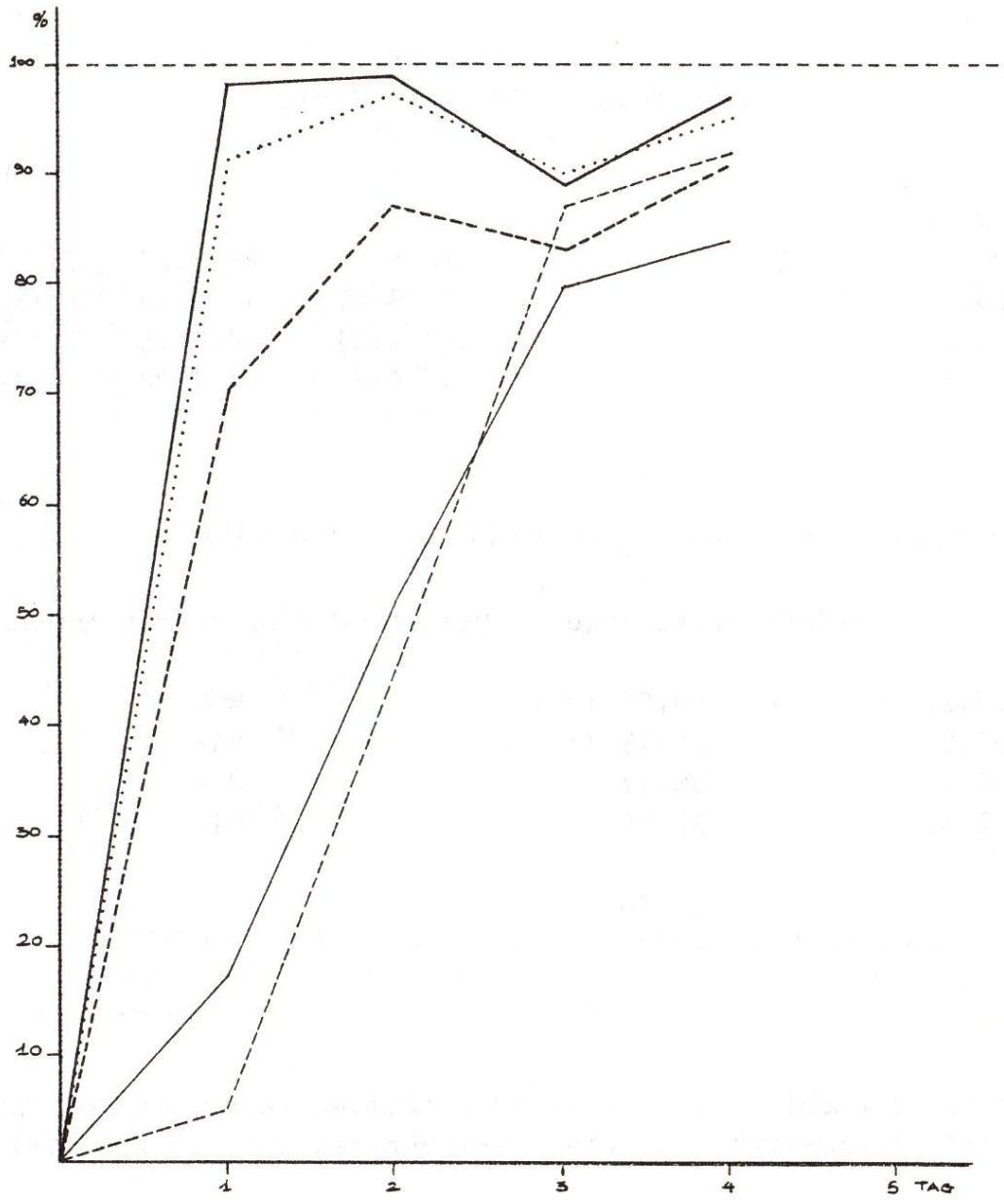
Gruppe B

Kapitelnummer
falsche Antworten/Zeit

Datum	3	4	5
20.2.	2/ 6:51	9/ 8:15	30/13:54
26.2	1/ 6:27	3/ 6:54	13/10:29
28.2.	11/ 7:42	10/ 7:45	17/11:07
2.3.	3/ 6:44	5/ 7:00	9/ 9:12

	Prüfungszeit insg.	Prozentsatz richtiger Antworten
20.2.	59:00 (**)	86%
26.2.	38:15 (***)	94%
28.2.	26:34	87%
2.3.	22:56	94%
Summe:	2:26:45	

(**) Einschließlich einer 30-minütigen Vorbereitungszeit.
(***) Einschließlich einer 15-minütigen Vorbereitungszeit.



Graphik 2: Prüfungszeit und Prozentsatz richtiger Antworten in Phase3

Diskussion

Graphik 1 zeigt eine fast ideale Kurve zur Bewältigung eines gegebenen Wissensstoffes: steiler Anstieg zu Beginn und Abflachen in fast paralleler Gangart zur 100%-Grenze zum Ende hin. Die Fragen zu 500 Wörtern wurden nach 9 Arbeitsstunden (5 Stunden in der ersten Phase und 4 Stunden in der zweiten) zu 97% richtig beantwortet. Die Reflexionszeiten lagen bei einer Gesamtprüfungszeit von 36 Minuten am letzten Prüfungstag bei 4.2 Sekunden pro Wort; bei diesem Arbeitsrhythmus hätten in 60 Minuten 833 Wörter wiederholt werden können.
Der erste Prüfungstag der zweiten Phase wurde vom Probanden als anstrengend bezeichnet, Tag 2 und 3 seien angenehm, Tag 4 und 5 "langweilig" gewesen. Dies bestätigt frühere Beobachtungen, nach denen Erfolgsquoten von über 90% nur vorübergehend stimulierend sind, mehrfach hintereinander erzielt der Arbeit jedoch die Spannung nehmen. Für weitere Versuche haben wir uns daher überlegt, die Fehler des 3. Prüfungstags zu sammeln und vom Rechner in ein getrenntes Kapitel schreiben zu lassen. Nur dieses würde dann am 4. Tag exklusiv in besonders intensiver Form bearbeitet.

Graphik 2 und Tabelle 2 geben die Ergebnisse der dritten Phase wieder. Fast enttäuschend die Ergebnisse in Gruppe A am 20.2.: 11 Prozent richtige Antworten bei einer Gesamtprüfungszeit von 1:34:23 pro 300 Wörtern ließen an der Methode Zweifel aufkommen. Das 5-Monate-Intervall zwischen erster Bearbeitung und Aufarbeitung schienen genug, um die Anstrengung vom Oktober zu annullieren. Eine leichte Modifizierung der Versuchsanordnung zeitigte denn aber die erwarteten Resultate: schon 10 Minuten Vorbereitungszeit pro 100 Wörter (6 Sekunden/Wort!) reichten aus, um den Prozentsatz der richtigen Antworten auf 86% bei 300 Fragen zu heben. Der gelernte Stoff war nur scheinbar abhanden, er lag reaktivierbar und wegen seiner Irrelevanz für den täglichen Bedarf richtigerweise verdrängt in schnell auffüllbarer Form bereit. Dementsprechend waren auch am 2. Tag die Ergebnisse in Gruppe A erheblich besser als am

ersten; die vortägliche Auseinandersetzung mit Kapitel 1 und 2 (Eintippen der richtigen Antwort, Schlußwiederholung der Fehler) hatten den Stoff aktiviert, wenn auch langsamer als die gezielte Wörterwiederholung in Gruppe B.
Schon Tag 3 brachte schließlich die Übereinstimmung der Kurvenpaare, die erst für einen späteren Zeitpunkt erwartet wurde. Während Gruppe B wieder absank, weil die Prüfung nun ohne vorherige Wiederholung des Wortschatzes abgelegt wurde, stieg Gruppe A weiter bis auf einen Prozentsatz von 83% richtige Antworten. Der Unterschied zwischen beiden Gruppen an diesem und am nächsten Tag betrug nunmehr nur noch 4 bzw. 6 Prozentpunkte. Am 4. Tag erreichte Gruppe B einen dem Oktobermonat vergleichbaren Wert.

Wenn wir den zuletzt bestehenden Unterschied von etwa 5 % zwischen den beiden Gruppen vernachlässigen, spielt es für das Endergebnis einer Nachbereitung keine Rolle, ob der Arbeitsstil gemäß Gruppe A oder gemäß Gruppe B gewählt wird. Das Vorgehen nach B ist schneller und wahrscheinlich ein wenig effektiver, es ist nach Aussagen des Probanden aber auch anstrengender. Nach der 20-Minuten-Bearbeitung eines B-Kapitels am ersten Tag sei er "gerädert, völlig geschafft" gewesen. Demgegenüber sei der erste Tag in Gruppe A relativ schonend ausgefallen. Es ist am Ende daher wohl eine Frage des persönlichen Vorzugs und der augenblicklichen Verfassung, welche Art des Vorgehens gewählt wird, ob gemächlicher und zeitaufwendiger (2 Stunden 26 gegen 3 Stunden 27) oder aufreibender und schneller. Beim Vorgehen nach Gruppe B betrug die Gesamtprüfungszeit der 4 Tage 2:26 für 300 Wörter. Bei 500 Wörtern wären dies 4:05 gewesen. Nach 4 Stunden Arbeit wäre also der Stoff in einer Intensität präsent gewesen wie 5 Monate zuvor. Die Reaktivierungszeit im Vergleich zur Arbeitszeit aus Phase 1 hätte 46% betragen.

Erfolgsquoten beim Computer-assistierten Vokabeltraining sind im wesentlichen abhängig von 2 Faktoren:
- der Maschinenschreibgeschwindigkeit
- dem Schwierigkeitsgrad der gewählten Sprache.

Beobachtungen an einer Anfängerin im Maschinenschreiben zeigten, daß bei einer von ihrer Morphologie her wenig vertrauten Sprache (in diesem Fall Fremdsprache Deutsch, Muttersprache Französisch) die Lernquoten der ersten Phase etwas geringer sind als im oben beschriebenen Versuch. Nach der ersten Woche, in dem sich die junge Französisch mit der Schreibmaschinentastatur vertraut gemacht hatte, lagen die Lernzeiten für 50 Wörter und abschließenden 90% richtiger Antworten zwischen 40 und 45 Minuten. Der Stundenschnitt hätte hier also bei 66 bis 75 Wörtern gelegen.
Auf der anderen Seite hätte der Proband unserer Versuchsanordnung noch Besseres von sich geben können, wenn die Prüfungssprache nicht Griechisch, sondern eine andere, ihm vertrautere Sprache gewesen wären. Es ist sicher nicht unrealistisch vorherzusagen, daß bei günstigen Ausgangsbedingungen (etwa: Prüfungssprache Italienisch oder Portugiesisch bei sehr guten Kenntnissen des Spanischen) die stündliche Lernquote bei 200 Wörtern liegen kann. Für den Schreibmaschinenanfänger errechnen sich im gleichen Fall Quoten von 100-120/h. Eine interessanter Gedanke drängt sich auf: es scheint, als sei die Lerngeschwindigkeit im Computertraining weniger eine Funktion der Hirnleistung als vielmehr der Fingerfertigkeit. Spitzenleistungen im Computerlernen bleiben demnach vorerst Sekretärinnen und Schriftstellern vorbehalten.

Wenn auch die durch den beschriebenen Versuch angedeuteten Trends grundlegend sind, so sind die vorliegenden Zahlen doch durch weitere Versuchsreihen zu bestätigen. Ein umfangreicherer Wortschatz müßte von einer großen und nach Maschinenschreibfertigkeiten unterteilten Gruppe bearbeitet werden. Kontrollen würden nach unterschiedlichen Intervallen (3, 6, 9, 12, 18 und 24 Monaten) durchgeführt. Um definitiv aussagekräftig zu sein, werden für solche Untersuchungen mehrere 100 Personen benötigt. In Erwartung endgültiger Resultate können wir dennoch heute schon formulieren, daß wir derzeit neben dem Computer-assistierten Vokabeltraining über keine weitere Methode verfügen, die ähnlich mächtig ist.

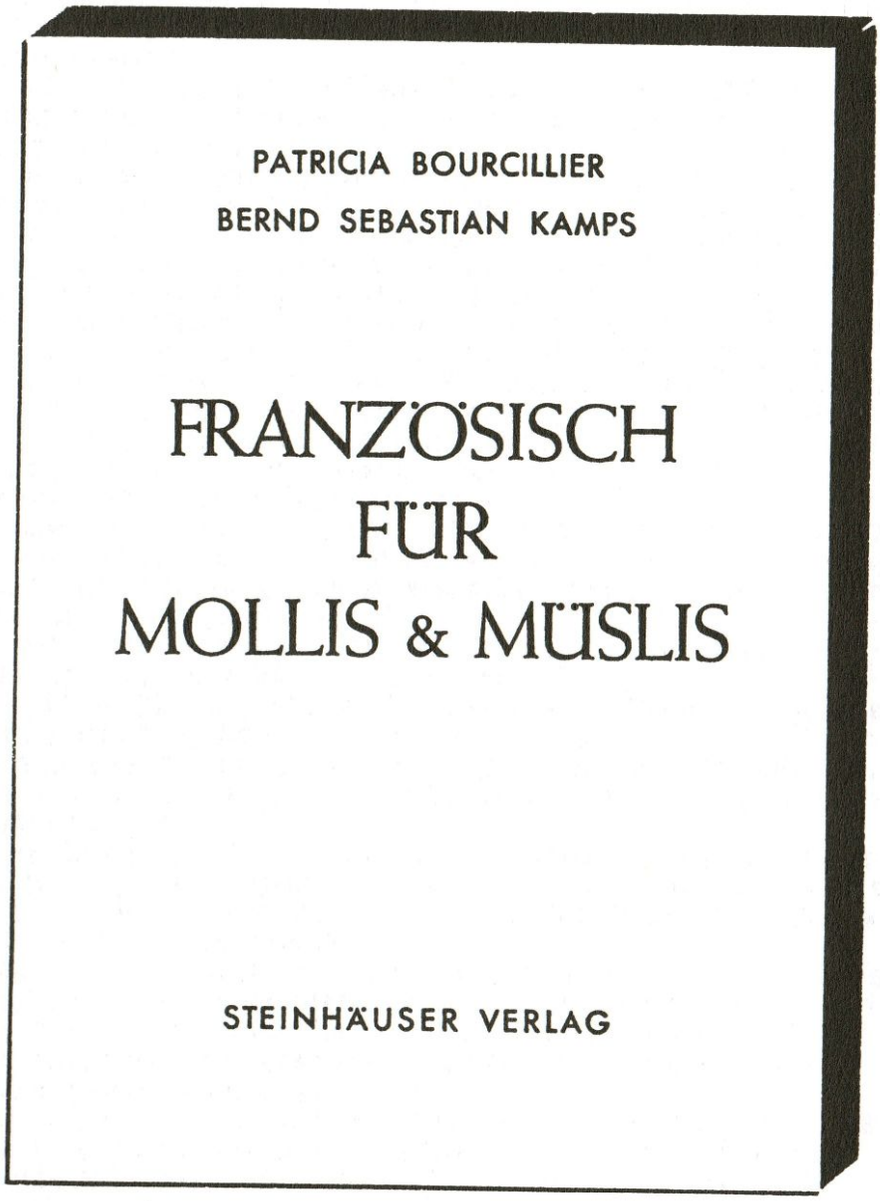
"Grunzlaute aus vollgefressenen Wohlstandsbäuchen"

Benjamin Delacroix im Gespräch mit Bernd Sebastian Kamps und Thomas Kamradt

B.D.: Herr Kamradt, Sie haben an guten Tagen 300 Vokabeln in 3 Stunden gelernt. Sie sind also ein Genie?
Kamradt: Das würde ich bescheidener sehen. Sicher ist nur, daß ich optimal motiviert war. Der Tapetenwechsel, das Wiedersehen mit alten Freunden, die Atmosphäre des Versuchsaufbaus. Außerdem wollte ich natürlich auch selber sehen, bis zu welchen Grenzen man mit seiner Gedächtnisleistung vorstoßen kann.
B.D.: Wie können Sie, Herr Kamps, dann die Ergebnisse Ihres gemeinsamen Versuchs verallgemeinern, wenn unser Proband sich solch außergewöhnlicher Umstände erfreuen durfte?
Kamps: Weil Beobachtungen, die ich auch an anderen Schülern gemacht habe, ähnlich gute Resultate zeigten. Der Versuch mit Thomas sticht allerdings heraus, weil es der erste Langszeitversuch dieser Art war.
B.D.: Dann haben Sie mit Ihrem Computer-Programm den Schlüssel der Weisen gefunden?
Kamps: Mit großen Einschränkungen. Der angegebene Wert von 250-300 Vokabeln/3 Stunden unterliegt zwar keinen großen Schwankungen von Individuum zu Individuum, höchstens +- 20%, aber ...
B.D.: Wenn Sie all die ausklammern, die man landläufig als sprachunbegabt bezeichnet!
Kamps: Wissen Sie, Herr Delacroix, der Begriff der Sprachbegabung und -unbegabung ist eine Erfindung von frustrierten Lehrern. Es gibt nur motivierte und unmotivierte Schüler, und letztere werden als sprachunbegabt bezeichnet. Die Wirklichkeit aber ist die, daß der Lernerfolg gerade bei Fremdsprachen in linearer Beziehung zur investierten Zeit steht.
B.D.: Also doch der Schlüssel der Weisen?
Kamps: Nein, die Einschränkungen, die ich gerade ansprechen wollte, sind wesentlich. Mit unseren Programmen kann nur der arbeiten, der visuell, also über's geschriebene Wort lernt. All jene, die einen Lernstoff besser über's gesprochene Wort verdauen, und dies sind etwa 20% aller Schüler, sind für unser Verfahren eher ungeeignet. Eine weitere bedauerliche Schwäche ist zur Zeit auch noch, daß die Aussprache des fremdsprachlichen Wortes fehlt. Im Italienischen ist dies glücklicherweise wegen der für uns Deutsche einfachen Ausspracheregeln kein sehr großes Problem. Für andere Sprachen wie das Französische etwa wird man aber noch ein paar Jahre warten müssen, bis eine weiter entwickelte Hardware auch dieses Problem löst. Großcomputer sind schon heute in der Lage, die menschliche Stimme sehr getreu nachzubilden.
B.D.: Dann können wir den Menschen also bald ganz abschaffen?
Kamps: Quatsch! Sie vergessen, daß wir von einem Lernprogramm sprechen, das auf wenige Wochen beschränkt ist und nur als Vorbereitung einer Reise ins Ausland verstanden

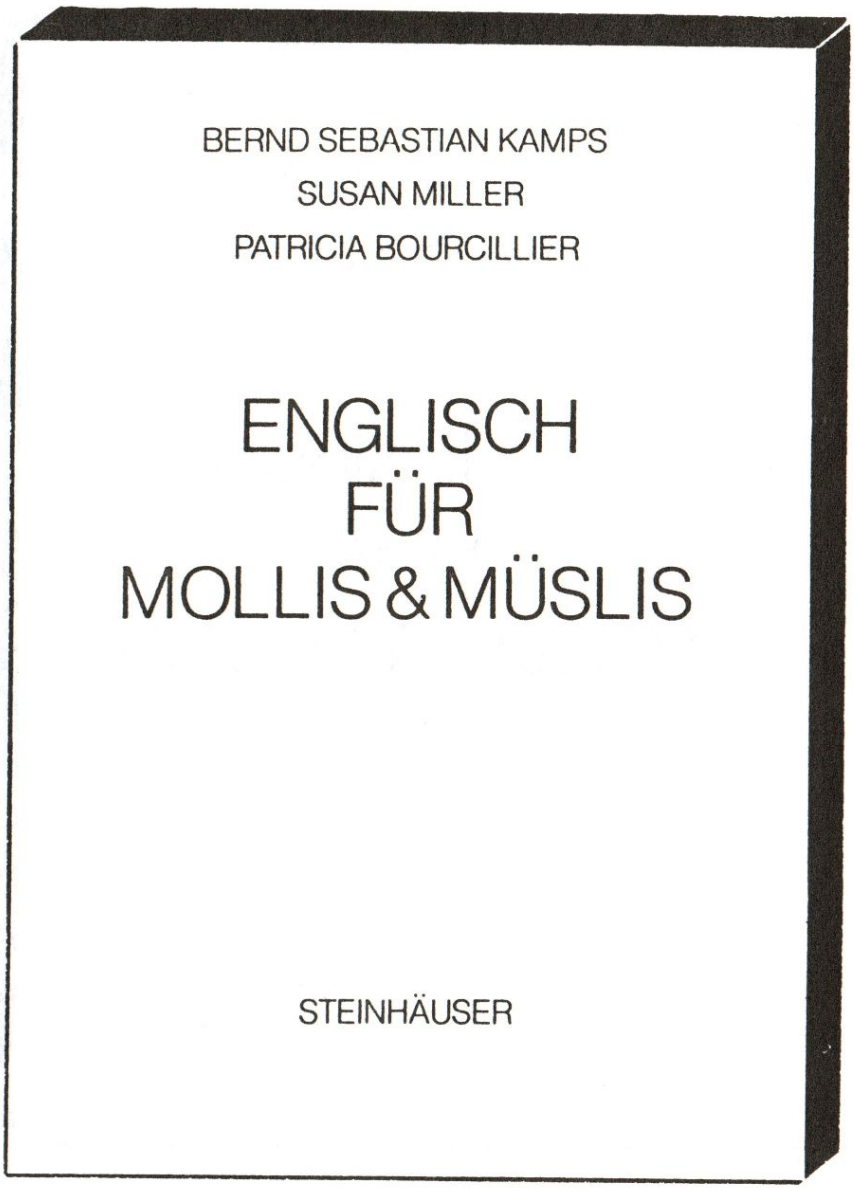
wird. Alles, was danach kommt, geht über den Menschen.
B.D.: Mir ist bei dem Testlauf der Programme aufgefallen, daß keine Übungen zur Grammatik vorgesehen sind. Sind technische Schwierigkeiten der Grund?
Kamps: Nein. Wir bezweifeln grundsätzlich die Übernahme von Übungen in den Erwachsenenunterricht. Übungen sollen den Schüler nur in die Lage versetzen, sein Wissen bei Prüfungen für gute Noten aktiv anzuwenden. Der Erwachsene aber lernt zumeist zum eigenen Vergnügen und legt keine Prüfungen ab. Für die Ankunft im Gastland muß er statt dessen genügend Wissen passiv "abgspeichert" zu haben, das erst dort aktiviert wird. Übungen im Erwachsenenunterricht sind ein Klotz am Bein.
B.D.: Herr Kamradt, Sie haben vor einigen Tagen davon gesprochen, daß sie die Anwesenheit anderer Personen als störend empfunden haben, wenn sie sich vor dem Computer der italienischen Grammatik widmeten. Sehen Sie nicht eine Gefahr in dieser Isolation, glauben Sie nicht, daß eine Fremdsprache im Gespräch, im Kontakt mit anderen Personen erlernt werden sollte?
Kamradt: Unfug! Lernen ist Isolation, jedenfalls phasenweise und vor allem immer dann, wenn auswendig gelernt werden muß. Dieser ganze "Lernen-beim-Sprechen-Kult" scheint mir ein reichlich fadenscheiniges Mäntelchen über der Scham der Faulheit zu sein. Sehen Sie: unregelmäßige Verben müssen einfach auswendig gelernt werden, das erfordert einen definierten Zeitaufwand und geht am schnellsten, wenn man sich eine gewisse Zeit zurückzieht, bis man diese Formen eben gelernt hat. Sie haben recht, daß es sinnlos wäre, wollte man zum Beispiel das Schreinern eines Tisches am Computer erlernen. Dabei wäre sicherlich der lebendige Kontakt zu einer kompetenten Person zweckmäßiger. Ich bin jedoch davon überzeugt, daß es Dinge gibt, seien es die Formen des italienischen Konjunktivs II oder die Sofortttherapie des akuten Herzversagens, die man nicht im freundlichen Geplauder lernt, sondern nur in disziplinierter Arbeit.
B.D.: Mir ist aufgefallen, daß in unserem Gespräch öfter die Worte Disziplin, Arbeit und Fleiß gefallen sind. Das hört sich ein wenig nach deutschen Volkstugenden an und macht mich unruhig.
Kamps: Deutsche Volkstugenden machen auch mich unruhig, seien Sie unbesorgt. Nein, wenn diese Begriffe zeitweilig fallen, so nur, um sie dem Phantombild des leichtgemachten Fremdsprachenunterrichts entgegenzuhalten. Da wird dem lernwilligen Schüler von findigen Lehrern vorgegaukelt, es gäbe DIE Methode, die dem Schüler die lästige Arbeit abnimmt und ihm Wissen gleichsam im Schlaf zuführt. Dies ist gefährlich und dubios. Gefährlich deshalb, weil der Schüler entmündigt wird und weil ihm nicht gestattet wird, sich auf seine eigenen Fähigkeiten zu verlassen. Es hat ganz den Anschein, als lerne nicht der Schüler die Sprache, sondern die Methode. Dubios ist dieses Vorgehen deshalb, weil es ein schlechtes Licht auf den Berufsstand der Sprachlehrer wirft. Als könne die Entmündigung des Schülers die eigene Existenzberechtigung garantieren! Daß dann die Möglichkeiten, die dem menschlichen Gehirn innenwohnen, brachliegen bleiben,

wird in Kauf genommen, solange der eigene Mythos stimmt.
B.D.: Zügelung der Lernfähigkeit zur eigenen Beweihräucherung?
Kamps: Und zur wirtschaftlichen Absicherung! Sehen Sie, je länger ein Patient krank ist, umso reicher wird der Arzt. Übersetzt heißt das: Je länger ein Schüler lernt, umso reicher wird der Lehrer, umso mehr Bücher braucht er, umso mehr verdienen die Verlage. Zwar wird immer betont,wie wichtig die Kenntnis von Fremdsprachen in unserer Zeit ist, doch im gleichen Zug wird eine Aura des Geheimnisvollen um die ganze Materie aufgebaut. Die leidliche Schulerfahrung tut sicner ein ihres, um diese Aura zu ermöglichen. Doch es ist nicht zu übersehen, daß sich um das Fremdsprachenstudium eine Schmarotzerszene von Verlagen und Sprachschulen aufgebaut hat, in der es niemandem ernsthaft darum geht, dem Schüler seine Aufgabe zu erleichtern.
B.D.: Herr Kamradt, ich komme noch einmal auf ihre Aussage von soeben zurück. In bestimmten Fällen solle man sich in sein Kämmerlein zurückziehen und still mit sich arbeiten. Sie lernen sozusagen prophylaktisch für einen späteren Aufenthalt im Ausland, ohne Befriedigung in der Gegenwart. Wo bleibt das Hier und Jetzt?
Kamradt: Gehören Sie auch zu den Abdominalspießern?
B.D.: Ja, ehm... Wie meinen Sie das?
Kamradt: Wenn Sie an ihren Biologieunterricht zurückdenken, werden sie sich erinnern, daß zwischen Saat und Ernte ein wohldefinierter Zeitabschnitt liegt. Das Gerede ums Hier und Jetzt sind wohl nur Grunzlaute aus vollgefressenen Wohlstandsbäuchen.
B.D.: Herr Kamps, Sie sprechen am Ende Ihres Berichts von möglichen Auswirkungen auf die Lehrpläne zukünftiger Generationen. Bekommen die Schüler mehr Freizeit oder werden die Anforderungen steigen?
Kamps: Dazu läßt sich im Augenblick wirklich nichts sagen. Zuerst muß abgeklärt werden, ob ein Computerprogramm wie das unsrige, das für Erwachsene mit einer vom Schüler ganz unterschiedlichen Motivation entworfen wurde, überhaupt auf die Schule übertragbar ist.
D.B.: Ich male mir da schreckliche Bilder aus. Stellen wir uns vor, bei einem Schüler läßt die Sucht, die bei Herrn Kamradt glücklicherweise schnell abebbte, nicht nach. Wohin soll das führen?
Kamps: Die Sucht läßt sicner nach, spätestens wenn der Stoff beherrscht wird. Das Problem sehe ich hier vielmehr auf der Seite der Lehrer, die mit dem Wissen ihrer Schüler fertigwerden müssen. Lehrer, die in ihrer Materie nicht firm sind, werden sich auf amüsante Situationen einzustellen haben.
D.B.: Herr Kamps, Herr Kamradt, ich danke Ihnen für das Gespräch.

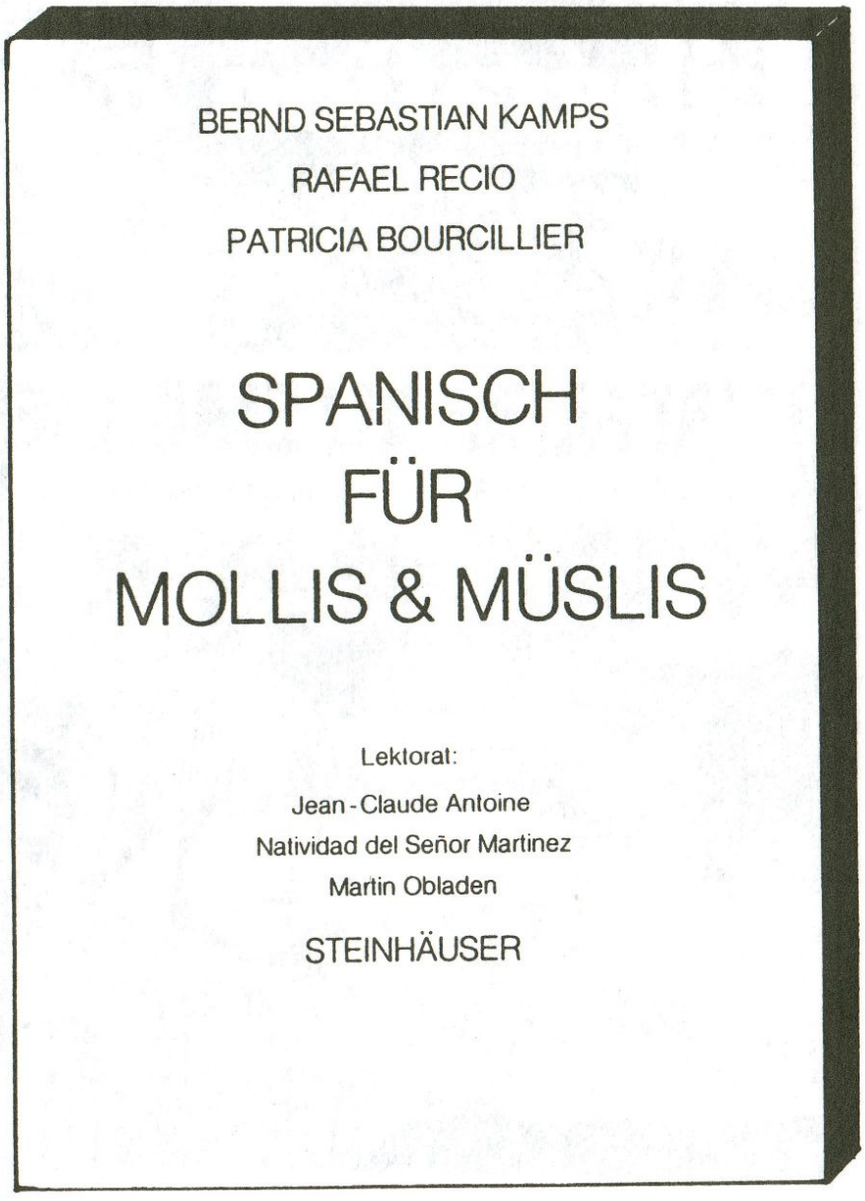


Erster, 1983 erschienener Band der Steinhäuser-Sprachreihe.
Mit Lehrbuchtexten zu Deutschland und Themen der Jugend.
Konzentrierung des Grammatikstoffes in den ersten Lektionen.
Betonung der Verbformen.
148 Seiten, Format 14 x 21, Paperback.
DM 15.-

Mit außergewöhnlichen Lehrbuchtexten zu Beziehungskisten, Drogen, Rassismus, Homosexualität, Atomkrieg. Konzentration des Grammatikstoffes entsprechend der Motivationsspitze in den ersten Lektionen. Umfangreiches Wortverzeichnis.
200 Seiten, Format 14 x 21, Paperback.
DM 15.-

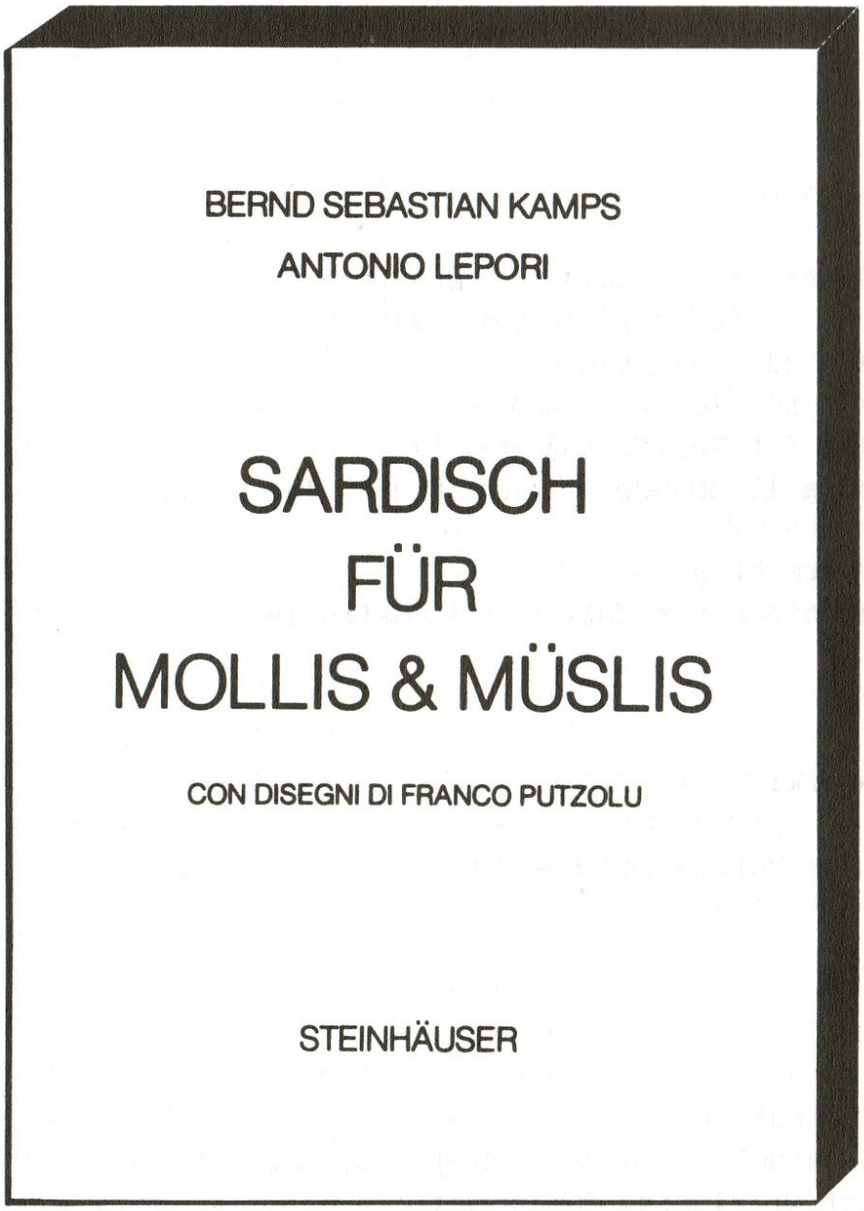


Mit Lehrbuchtexten zu Kriegsdienstverweigerung, Beziehungsandwich, Bullenschlacht, Wiedervereinigung, Frauenarbeitslosigkeit, Selbstmordgeneration. Vokabelverzeichnis am Ende des Buches.
132 Seiten, Format 14 x 21, Paperback.
DM 15.-



Mit Lehrbuchtexten zu Bishop, Liebeswahn, Parlamentarierpack, gerodeten Hochspannungsmasten, Folter in Europa, Impotenten, Anmacher, Hexenjagd. Ausführlicher Anhang mit ausgewählten Zeitungstexten.
168 Seiten, Format 14 x 21, Paperback.
DM 15.-

Ein Französisch-Textbuch mit Originaltexten über Deutschland: Studentenbewegung, Anfänge der RAF, Isolationshaft, Todesfälle in Stammheim, Schleyer-Entführung. Interviews mit Heinrich Böll, Andre Glucksmann, Peter Brückner, Daniel Cohn-Bendit.
168 Seiten, Format 14 x 21, Paperback.
DM 15.-



Das erste Sardisch-Lehrbuch in Deutschland und Italien! Mit Lehrbuchtexten über wirtschaftliche und politische Situation Sardiniens. Grammatikerklärungen zweisprachig auf deutsch und auf italienisch. Vokabellisten dreisprachig Sardisch - Italienisch - Deutsch.
176 Seiten, Format 14 x 21, Paperback.
DM 15.-

STEINHÄUSER LEHRBÜCHER

Fremdsprachen:

Französisch für Mollis und Müslis	15.-
Italienisch für Mollis und Müslis	15.-
Englisch für Mollis und Müslis	15.-
Spanisch für Mollis und Müslis	15.-
Sardisch für Mollis und Müslis	15.-
L'Affaire Allemande (französisches Textbuch)	15.-

In Vorbereitung:
Portugiesisch für Mollis und Müslis (Herbst 1986)

Informatik:

Basic für Mollis und Müslis	19.80

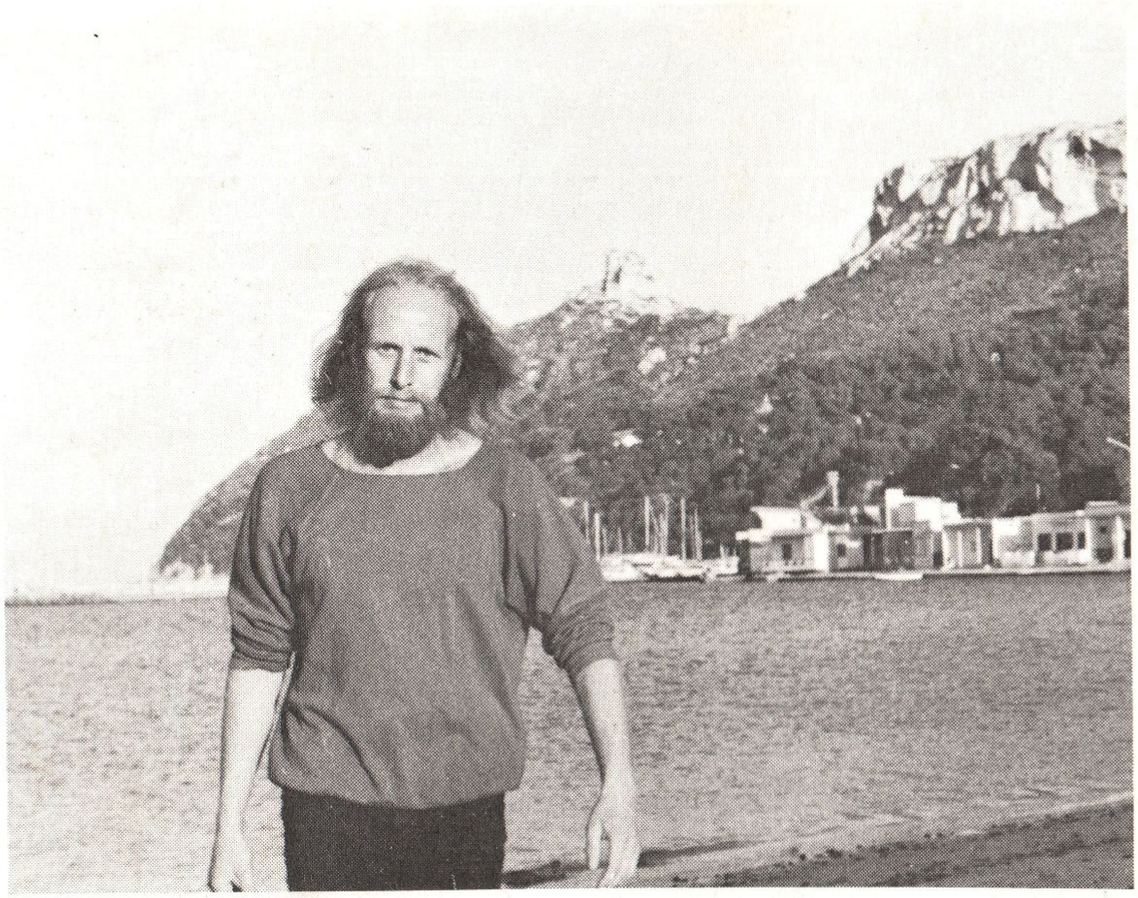
Medizin:

Präparierkurs (Präparieranleitung für den großen anatomischen Kurs der Vorklinik, 520 Seiten)	29.80

POLIGLOTTUS, das vom Steinhäuser-Verlag herausgegebene Programmpaket zum Fremdsprachenstudium, ist erhältlich in den nachstehend aufgeführten Buchläden. Programme und Datenbänke haben einen Umfang von **500 K (...)** und sind auf vier Disketten für COMMODORE 64 UND 1541-DISKETTENLAUFWERK gespeichert. Die in diesem Buch entwickelten Programme liegen dort in einer professionelleren Version vor. Folgende Sprachen sind zur Zeit schon in Vokabel- und Konjugationsübungen verfügbar: **Englisch, Französisch, Spanisch, Italienisch, Sardisch.** Grammatikübungen folgen in Kürze.

Die Disketten können gegen Hinterlegung einer Kaution von 50 DM für einen Tag zum kostenlosen Kopieren ausgeliehen werden.

1000 Berlin, Büchereck in Kreuzberg, Admiralstr. 1
1000 Berlin, Schwarze Risse, Gneisenaustr. 2
2000 Hamburg, Academic-Bücher, Schlüterstr.22
2800 Bremen, Ostertor Buchladen, Im Fehrfeld 60
2900 Oldenburg, C.v.Ossietzky Buchhandlung, Kurwigstr. 14
3400 Göttingen, Buchladen Rote Strasse, Rote Strasse 10
3500 Kassel, ABC Buchladen, Goethestr. 77
3550 Marburg, Roter Stern, Am Grün 28
4000 Düsseldorf, Stern-Verlag, Universitätsstr. 1
4630 Bochum, Politische Buchhandlung, Unistr. 26
5000 Köln, Der andere Buchladen, Zülpicherstr. 197
5300 Bonn, Buchladen 46, Kaiserstr. 46
6000 Frankfurt, Ypsilon, Bergerstr. 27
6000 Frankfurt, Karl-Marx-Buchlden
6200 Wiesbaden, Kassandra Buchladen
6300 Giessen, Ferber'sche Universitätsbuchhandlung
6600 Saarbrücken, der buchladen, Försterstr. 14
6800 Mannheim, Buchladen M2,6
7000 Stuttgart, Wendelin Niedlich, Schmale Str. 9
7400 Tübingen, die gruppe, Marktgasse 13
7410 Reutlingen, Jakob Fetzer Buchladen
7800 Freiburg, Jos Fritz, Wilhelmstr. 15
8000 München, Trampelpfad, Elsässerstr. 15
8000 München, adalbert 14, Adalbertstr. 14
8520 Erlangen, Buchladen am Lorlebergplatz
8700 Würzburg, Werner Beyer Buchladen



Bernd Sebastian Kamps, geb. 1954 in Wuppertal, lebt seit 1981 auf Sardinien. Seit 1983 Entwicklung von Computerlernprogramm für den Fremdsprachenunterricht.

Das vorliegende Lehrbuch schlägt zwei Fliegen mit einer Klappe: Einmal wird der Leser systematisch in die Programmiersprache BASIC eingeführt. Er gewinnt Einblick in die innere Organisation seines Rechners. Gleichzeitig lernt er, funktionsfähige und sehr leistungsfähige Trainingsprogramme für den Fremdsprachenunterricht zu schreiben. Beachtenswert ist auch der logische Aufbau der Stoffdarbietung: Nicht eine Fülle von kleinen und untereinander nicht zusammenhängenden Nonsensprogrammen wird dem Leser und Jungprogrammierer zur Abschrift und Probe vorgelegt, stattdessen wird von Anfang bis Ende des Buches an dem "Großprojekt Fremdsprachen" gearbeitet. Von Kapitel zu Kapitel werden die Programme professioneller und leistungsstärker, mit dem wachsenden Wissen wird zwiebelschalenartig ein immer komplizierteres Programmgebäude errichtet.