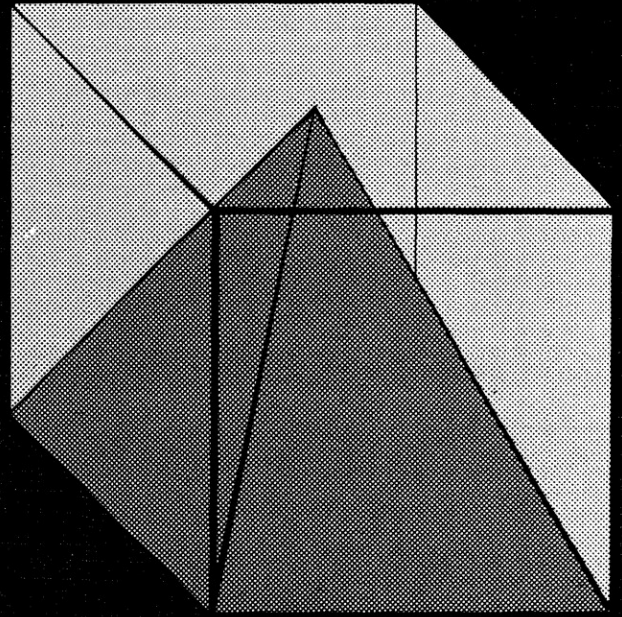


# SINCLAIR BASIC

Wer sich ernsthaft mit dem SINCLAIR beschäftigt, wird dankbar sein, wenn er mehr als nur „Spiele“ zu Gesicht bekommt. Die Grenzen von ZX81 und Spectrum liegen jenseits von STARTREK. Immer mehr Jugendliche und Erwachsene fragen, was man darüber hinaus mit einem „Micro“ anstellen kann. Viele fangen an, selber in BASIC, der verbreitetsten Programmiersprache, zu programmieren. Für den BASIC-Programmierer ist guter Rat oft teuer, was heißt: nicht erhältlich. Wie man effektiv und schnell programmiert will dieses Buch aufzeigen. In mehreren abgegrenzten Kapiteln wird systematisch gezeigt, wie man mehr aus seinem SINCLAIR herausholt. An alles – sowohl für den Anfänger wie den Fortgeschrittenen – ist gedacht.

All das ist verständlich und nachvollziehbar, keineswegs in geheimnisvollem EDV-Kauderwelsch geschrieben. Hier war ein Programmwerk, der seine jahrelange Erfahrung mit dem ZX81 und Spectrum klar und einleuchtend zu Papier gebracht hat. Kurzum, ein Buch für alle geschrieben, die selber mit dem SINCLAIR programmieren.



FJ  
5420

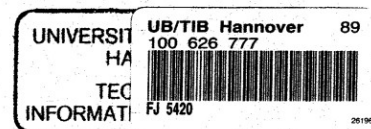


Tom Softwell

# Sinclair BASIC

Programmiertips & -tricks

für ZX 81 und Spectrum





FJ 5420

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Softwell, Tom:  
 Sinclair BASIC : Programmiertips & -tricks für  
 ZX 81 u. Spectrum / Tom Softwell. -  
 Gensingen : Luther, 1985  
 ISBN 3-620-00137-5

Titelseite: Peter Spann, Karlsruhe

Alle Rechte, auch die der Übersetzung in fremde Sprachen, vorbehalten. Kein Teil dieses Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden. Bei der Zusammenstellung wurde mit größter Sorgfalt vorgegangen. Fehler können trotzdem nicht vollständig ausgeschlossen werden, so daß weder der Verlag noch der Autor für fehlerhafte Angaben und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen. Warennamen sowie Marken- und Firmennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt. Für Verbesserungsvorschläge und Hinweise auf Fehler ist der Verlag dankbar.

© Copyright 1985 by W.-D. Luther - Verlag,  
 6531 Gensingen, Printed in Germany  
 \*\* ISBN 3-620-00137-5 \*\*

Vorwort

7

## KAPITEL I: DIE SPRACHE

## EIGENSCHAFTEN

Stärken	9
Syntax und Semantik	9
Flexibilität	10
Narrsicherheit	12
Schwächen	13
Allgemeine Nachteile	13
Fehlende Befehle	14
Fehler und Fallen	14

## AUSNUTZEN DES SINCLAIR-BASIC

Wertzuweisungen	17
Initialisieren	17
Vielseitiges UAL	17
Input vom Programm - DATA	18
Abfragen und Vergleiche	19
Die Booleschen Operatoren	20
IF-lose Abfragen	20
Sprünge und Schleifen	22
Berechnete Sprungziele	24
Schleifenvariationen	24
Programmoptimierung	25
Speicher sparen	28
Speeding up (Schnellermachen)	28

## BESSERS BASIC MIT BETABASIC

Auf dem Weg zur Hochsprache	30
Maschinencode ist nicht jedermanns Sache	30
Einzelroutinen bleiben Stückwerk	30
Die wichtigsten Schwerpunkte	31

Ein Außenseiter zeigt, was möglich ist	31
Arbeitserleichterungen	32
Beurteilung	33
Weniger Speicher und ein paar 'Bugs'	33
Zusammenfassung	33

## KAPITEL II: G R A F I K

ZAHLENDARSTELLUNG	35
Monatstage optisch aufbereitet (KALENDER)	35
Aufgabenstellung	35
Lösungbestandteile	35
Programmbeschreibung	36
Skulengrafik	36
Zahlen in HiRes	38
Kernproblem Darstellungsmaßstab	38
Angaben zum Programm	42
BILDSCHIRMARBEIT	43
Screenbewegungen	43
Inscreen Copy	43
Screen Switch	45
Scrolls	48
Bildschirmhilfe DRAWUTIL	49
Was muß ein Grafik-Utility alles können ?	49
Features	50
Funktionen	51
Programmbeschreibung	55
SCREEN - MANIPULATIONEN	57
Viele Daten auf dem Bildschirm mit HEIMBUDGET	57
Grafik voll ausgeschöpft	57
Was kann das Programm ?	58
Wie ist das Programm aufgebaut ?	59
Programmbeschreibung	60
Screenricks (im Programm)	70

Datenverwaltung	70
Grafikeinsatz	70
Weitere Programmricks	72

## KAPITEL III: Z E I C H E N & T E X T

ZEICHEN AUF DEM BILDSCHIRM	
Manipulation des 'character sets'	74
Darstellung von Zeichen	74
Die Adressierung	75
BIG PRINT oder die Erzeugung von Riesenlettern	75
Weitere Manipulationen	77
Die Generierung eines neuen Zeichensatzes	81
Wie ist vorzugehen ?	81
Arbeiten mit VCHAR (Vertikalcharacters)	82
Texte auf dem Bildschirm	84
Inputs	84
Inscreen - Input	84
Text + Formatierung	86
Blocksatz	88
PROGRAMMENTWICKLUNG VON "TEXTVER"	90
Von der Zielsetzung zum Programm	95
Vorüberlegungen	95
TOP DOWN Vorgehensweise	95
Von der STEUERUNG zum Modul	96
Die Programmierung der Module	97
Text-Handling	101
Die Formatierung von Texten	105
Zwei Lösungen zur Auswahl	106
Die Lösung mit Flattersatz	106
Die Lösung mit LPRINT	107
Programmdokumentation	110
Modulbeschreibung	112
Variablenliste	112
Benutzungshinweise	114

# KAPITEL IV: VERARBEITUNG VON DATEN

DATEN IN LISTEN	116
Arbeiten mit Records + Reports	116
Datenorganisation	116
Die Synthese von Daten und Listen	118
Die Zuordnung von Daten zu Reports (Programm REPDAT)	119
Aufgabenstellung	119
Programlösung	120
Programmdokumentation	124
SUCHEN UND SORTIEREN	134
Sortiervverfahren	134
Aus der Fülle der Möglichkeiten	134
Einfach aber langsam - der BUBBLE-Sort	135
Der effektive SHELL-Sort	136
Ganz schnell - der QUICKSORT	137
Suchen und Wiederfinden	139
Ideal - der Direktzugriff	139
Linear contra binär	140
Blocksuche	141
FORMEN DER SPEICHERUNG	143
Der universelle Datenzeiger	143
Speicherungsgrenzen	143
Platz sparen mit Pointern (Programm KUERZEL)	145
Sonderformen der Speicherung	150
UDG's als Floskelpeicher	150
Der Screen als Speicher	152
Literaturverzeichnis	156

Vorwort: Heimcomputern will gelernt sein

Wer sich ernsthaft mit dem SINCLAIR beschäftigt, hat das "Spielestadium" schnell überwunden. Er merkt - die Grenzen von ZX81 und Spectrum liegen jenseits von STARTREX. - Wie man effektiv und pfiffig programmiert will dieses Buch aufzeigen. In mehreren abgegrenzten Kapiteln wird systematisch entwickelt, wie man mehr aus seinem SINCLAIR herausholt. Für alle, die BASIC können, wird das hierzu Erforderliche verständlich und nachvollziehbar beschrieben. Der Autor hat versucht, seine jahrelange Erfahrung klar und einleuchtend zu Papier zu bringen. Allein die vollständigen Listings (komplette Programme und viele Routinen) sind eine wertvolle Hilfe. Aber das ist nicht alles..

Das Buch wendet sich an Heimcomputerbesitzer, die Basic beherrschen und die Maschine voll nutzen wollen. Praktische Anwendung und Verständnis für den flexiblen, trickreichen Einsatz der Sprache stand im Vordergrund. SINCLAIR-Basic ist dermaßen mächtig, daß die Grenzen weit gezogen werden können. Wir beschränken uns zwar auf ZX81 und Spectrum als "Maschine", werden aber viele Verweise zur Befehlserweiterung bringen. Strukturierte Elemente sind als "Selbstbau", Zusatzsoftware und Hinweise aufs QL - Superbasic mit behandelt. Maschinencoderoutinen werden hingegen nur zur Ergänzung an wenigen Stellen herangezogen. Oder wußte der SINCLAIR-Fan bereits, daß

- man in Basic Windows schaffen kann
- sich eigene Scrolls aufrufen lassen
- Vertikalschrift und neue Zeichensätze machbar sind
- Schriften vergrößert und verdichtet werden können
- Text leicht in Blocksatz formatierbar ist
- effektive Suchprozeduren mit Datenzeigern arbeiten
- Listen, Daten und Grafik integriert werden können ?

Das Buch bringt alle Lösungen in Basic, es schöpft die Möglichkeiten der Sprache voll aus. Nach einer Einführung in diese (mit vielen Tips & Tricks) geht es los mit Grafik und Bildschirmarbeit, weiter mit Zeichen und Textmanipulation bis hin zur Verweitung umfangreicher Daten und Listen. Allein das Programmangebot ist beträchtlich, es umfaßt:

- KALENDER erzeugt tageskorrekte Monats/Jahresdisplays
- SAEULENGRAFIK bringt hochauflösende Balkengrafik mit automatischer Skalierung
- DRAWUTIL ist ein Zeichenprogramm, mit welchem Text, Box und Strich, Kreise und Punkte gemalt werden
- HEIMBUDGET verwaltet beliebig viele Positionen und breitet Summen, Prozentanteile etc. optisch auf
- BIG PRINT + UCHARS generieren Riesenbuchstaben bzw. Vertikalschrift (mit einem völlig neuen Screengefühl)
- TEXTVER bringt eine komplette Textverarbeitung incl. Formatierung und Blockoperationen
- REPDAT verwaltet Daten beliebigen Typs gemischt (Text + Zahl)

und stellt sie in frei definierten Reports dar

Aber all das ist nicht isoliert in "SINCLAIR-Basic"; vielmehr wird der Leser an die Lösung herangeführt. Nachvollziehbar nimmt er an deren Entwicklung teil. Vieles wird in überschaubaren Modulen aufgezeigt, anderes diskutiert und erläutert, alles jedoch beispielhaft dokumentiert. Die Programme/Routinen sind nicht nur benutzerfreundlich, effizient und "trickreich", sondern auch mit allem Nötigen genau beschrieben: Aufbau, Funktion, Schaubilder, Hardcopies, Variablenliste, Benutzungshinweise etc. So kann der Leser höchsten Nutzen aus dem Gebotenen ziehen. Und da uns die Praxis der beste Lehrmeister scheint, haben wir

- den gesamten Text mit einem Textverarbeitungsprogramm erstellt
- alle Diagramme mit dem abgedruckten Programm (DRAWUTIL) gemalt
- die Listings mit einer kritisch beschriebenen Spracherweiterung (BETABASIC) formatiert.

Ein Wort zum Abschluß. Wer SINCLAIRs Basic ernst nimmt und die Sprachmöglichkeiten voll ausschöpft, läuft Gefahr, sich an den Heimcomputer zu verlieren. Eigentlich müßte jeder SINCLAIR die Aufschrift tragen

Der Bundesminister für Forschung und Technologie :

VORSICHT, DER HC KANN IHR FAMILIENLEBEN GEFÄHRDEN !

## Kapitel I : Die Sprache

### EIGENSCHAFTEN

#### Stärken

#### Syntax und Semantik

Dies Kapitel ist als Einführung zu den Anwendungen in SINCLAIR-Basic zu verstehen. Man erwartet keine Befehlsübersicht oder ähnliche Buchfüllsel. Das Handbuch ist dazu gut genug. Wir wollen vielmehr auf die Eigenheiten der Sprache, deren Beherrschung beim Leser vorausgesetzt wird, eingehen. Es soll der "Nerv" dafür geweckt werden, was sich mit ihr alles anstellen läßt. Nicht Einzelheiten der Befehlsformate, sondern Eigenheiten des Instruktionssatzes. Die Ausführungen des Kapitels I sind Merkpunkte für Programme bzw. Routinen des Buches - ohne Anspruch auf Vollständigkeit. Eine Sprache - und das ist SINCLAIR-Basic nun mal - ist bestimmt von Syntax und Semantik. Beides steht voneinander abgegrenzt: quasi das Regelwerk, kurzum die Grammatik; Semantik ist die Ausdrucksfähigkeit, kurzum der Wortschatz. Der Gegensatz beider ist jedoch nur bedingt. Eine Programmsprache kann wenige Befehle haben, aber mächtig sein, also viele Operationen und "Fälle" abdecken. Andererseits bedeutet großer Instruktionssatz nicht unbedingt hohe Flexibilität bei Problemlösungen. Statementklauseln können so starr sein, daß nur ein Haufen Spezialfälle übrig bleibt. Ist die Sprache befehlssarm, aber die Verwendung (bei wenig Vorschriften) "frei", läßt sich viel mit ihr anstellen. Wenn jedoch ein Basic befehlss- und anwendungstarr ist (wie viele Computer, die stolz den Zusatz 64 im Namen tragen), bleibt der Heimcomputerist der Dumme. Bei aller Intelligenz kann er nur teure Erweiterungen kaufen, die schnell veralten - sprich erst langsam bugfree werden -, oder es sein lassen.

SINCLAIR-Basic kann man als "starke" Programmsprache charakterisieren; sein Basic ist mächtig aufgrund flexibler Verwendbarkeit der Statements. Aber hören wir zu Anfang, was aus berufenem Munde über Sinclairs Basic gesagt wird, ehe wir auf Schwerpunkte der "Programmarbeit" eingehen. Zu dem Zweck sei auf einige Aussagen zurückgegriffen, die der Autor in einem anderen Buch des Verlages gesammelt hat ("ZX 81/Spectrum - Anwendungen", s. Literaturverzeichnis). - Mike James, Experte von "Electronics & Computing Monthly", bewertet Sinclair-Basic so: "Wie (seiner) Eigenschaften zusammenfassen, um ... eine ansprechende und sparsame Sprache abzugeben, das macht ZX-BASIC für Anfänger und Experten gleichermaßen geeignet." Und um gleich auf ein

wichtiges Feature, das das erhärtet einzugehen, wieder Mr. James: "ZX-BASIC selektiert seine Datenwerte und Datentypen, um es möglich zu machen, Ausdrücke so frei wie möglich miteinander zu vermischen."

Die Freiheit eines SINCLAIR - Programmierers fängt schon bei Variablennamen an. Anfängern kommen beliebig lange, sprechende Bezeichner (für Numeric-Variablen) sehr entgegen. Sie dürfen sogar Leerzeichen enthalten, eine Seitenheit in der HC-Landschaft. Dem stehen vielfältige Möglichkeiten indirekter Adressierung zur Seite. Sowohl Sprungadressen, als auch Indices und Berechnungen können mittels sgn. expressions (Formelausdrücke wie etwa  $A * B + 1$ ) erweitert werden. Darüber hinaus lassen sich 'Formeln' mit logischen Operatoren anreichern, sodaß einer ausgefeilten, kompakten Logik wie z.B.  $GOTO A + (C = 5)$  kaum Grenzen gesetzt sind. Wir werden noch auf Einzelheiten eingehen.

Gerade die mannigfaltigen String- und Vergleichsoperationen machen die Stärke des SINCLAIR - Basic aus. Nochmal abschließend Mr. James: "Meiner Meinung nach ist die Art, wie ZX-Basic Zeichenketten handhabt, eine seiner besten Eigenschaften - seine Methode sollte Standard werden." Um 'Standard' Microsoft allerdings ist das Basic etliches entfernt. - Da gibt es die Stringaufteiler LEFTS, RIGHTS und MIDS beispielsweise. Jahrelang kritiklos übernommen und gedankenlos "geschluckt" kam Sir Clive und zeigte, daß man mit einfacher TO-Klausel besser arbeiten kann. Drei Befehle weniger, aber vielseitiger, einleuchtender und eingabefreundlicher. Das ist typisch fürs SINCLAIR-Basic. Übrigens, wer zur Konvertierung ein Umwandlungshilfe braucht - die folgenden Functions erledigen L-left, R-right und M-id:

```
DEF FN LS (XS,X) = XS ( TO X )
DEF FN RS (XS,X) = XS (LEN XS - X + 1 TO )
DEF FN MS (XS,X,Y) = XS (X TO X + Y - 1)
```

Daß hinter dem Basic ein durchdachtes Systemkonzept steckt, zeigt sich an zahlreiche Arbeitshilfen, welche Sinclairs Maschine bietet. Neben Token-Eingabe (warum haben das nicht alle HCs ?) und Syntaxcks gibt es so angenehme Dinge wie SCREENS zur Zeichenbestimmung u n d Bildschirmladen/Sehen, leicht zu definierende UDG's - ohne Adressrechnung (ähnlicher Komfort bei CLEAR Memory) und einfaches Abspeichern von Variablen bzw. Arrays. - Aber hängen wir uns nicht an Details auf, beschreiben wir allgemeine Eigenschaften des SINCLAIR-Basic, seine Flexibilität, seine Narrensicherheit, ehe wir kritisch Schwächen aufzeigen. Auf Einzelheiten der Sprache wird noch im nächsten Kapitelteil einzugehen sein.

#### Flexibilität

Eine Programmiersprache hat ein "Sprachangebot" im wesentlichen zu folgenden Punkten zu machen; welche Befehle sie dazu zur Verfügung stellt, bestimmt entscheidend ihre Flexibilität :

Systembefehle	EDIT RUN NEW PAUSE LOAD SAVE STOP etc.
Entscheidungen	IF > < >= <= <> ...
Schleifen	FOR NEXT
Sprünge und	
Unterprogrammaufruf	GOTO GOSUB
Datenein- und	
-ausgabe	INPUT INKEYS READ PRINT LPRINT
vordefinierte(mathe-	
matische) Funktionen	FN X() LN SIN ABS usw.
String- + Arrayhand-	
ling	DIM + Subskribierung
Zuweisungen und	
Datenkonvertierung	LET STRS CHR\$ CODE VAL VALS
Gebrauch logischer	
Ausdrücke	AND OR NOT plus Kombinationen
Grafik,Ton + Farbe	PLOT DRAW CIRCLE INK PAPER u.m.

Verlassen wir die allgemeine Erwartungsebene von Basic - Dialekten; greifen wir auf, was SINCLAIR-Basic von anderen unterscheidet. Wo zeigt sich die besondere Flexibilität der Sprache ? Es sind drei Schwerpunkte auszumachen

1. die Stringverwendung,
2. logische Operatoren und noch
3. ein paar sonstige Befehlsvorteile.

Strings Zeichenketten können (als single string) fast beliebig lang sein. Sie sind einfach zu handhaben und erlauben vielfältige Manipulationen. Hervorragend ist die einfache Aufteilung durch TO - ohne Angabe erste und letzte Stelle. Umfangreiche Umbesetzungen ohne Zwischenspeicherung werden in einem Befehl erledigt, etwa so:

```

+-----+
v         I
|====I |====I |====I + I Ys I
- I I I I I I I I I I I
I   n1   I   n2   I
+-----+
```

LET XS = XS(n1 TO n2-1) + XS( TO n1-1) + XS(n2 TO) + Ys  
logische  
Operatoren

Die Verwendung logischer Vergleichsergebnisse ("True" und "False") geht noch über das hinaus, was der Datentyp 'Boolean, etwa in Pascal, bietet. Nicht allein wird das Ergebnis 1 = richtig oder 0 = falsch geliefert, die vereinfachte IF-Abfrage gestattet es auch, diese Wertigkeit selber zu testen: IF N THEN .. heißt IF N <> 0 THEN .. Logische Vergleichsergebnisse können zudem direkt eingebunden werden, also bei GOTO, PRINT-Selektion, Variablen- und Stringzuweisungen und vielem mehr. In welchem Dialekt kann man so prägnant wie hier formulieren ? Wer kann sowas ?

LET X = NOT Y OR Z=5

Dasselbe "lang" programmiert müßte lauten

```
IF Z=5 THEN LET X = 1
IF Z <> 5 AND Y = 0 THEN LET X = 1
IF Z <> 5 AND Y <> 0 THEN LET X = 0
```

Nicht nur bei Berechnungen ist ein logischer Operator nützlich. Er ist bei jedem Variablengebrauch möglich, z.B. als Pausenfunktion:

PAUSE N OR N oder PAUSE N<>0

Das bedeutet kurze Pause, wenn N<>0 und unendliche Pause, wenn N gleich 0. Fürs Update durchaus angebracht kommt es doch nur zu einem Halt, falls N noch unbesetzt ist..

#### Sonstiges

Die vielen Möglichkeiten indirekter Adressierung werden dadurch interessanter, daß man ad hoc Berechnungen mit einfügen kann. Sowohl bei Sprungzielen wie auch bei allgemeinen Variablen lassen sich 'offsets' (etwa Distanzwerte) einbinden. Beispiele:

GOTO SCHREIBEN + 5 oder PRINT IS (ELEMENT + 2\*N - 1)

Überall, wo es um Zahlen geht, kann man Formeln, Funktionen, logische Operatoren und Wertsurrogate (wie CODE "A" gleich 65) einbauen. Zwar hat SINCLAIR-Basic keine Proc's (genauer - erst das SUPERBASIC des QL), aber mit DEF FN und VAL lassen sich eine Menge indirekter Wertaufrufe realisieren. Da wir gerade bei Werten sind, im Gegensatz zur Konkurrenz ist eine beliebig tiefe Dimensionierung von Arrays möglich. Ein Plus, das sich lediglich wegen der Hauptspeichergrenze von rund 40 K RAM nicht ausschöpfen läßt.

Auch das Arbeiten mit dem Interpreter gestaltet sich recht testfreundlich. Dauernde Interaktion von Programmänderung und Lauf erlauben schnelles Arbeiten. Das Zusammenspiel geht etwa so vonstatten: Lauf, STOP oder BREAK, Befehlsmodifikation bzw. Echtwertabfrage oder -änderung (Direktmodus) - Restart mit CONTINUE oder GOTO..

Verlassen wir den Punkt "Flexibilität", ohne alles erschöpfend behandeln zu haben. Wenden wir uns noch einer Stärke des SINCLAIR-Basic zu, ehe die Lücken und Nachteile kurz abgehandelt werden.

#### Narrensicherheit

Darunter ist die Unempfindlichkeit gegen nicht ganz korrekte

Befehlsverwendung zu verstehen (engl. Fachterminus error trapping). Manche MCs, die mit viel Werbeaufwand in den Markt gedrückt werden, zeigen sich gegen Fehleingaben hoch sensibel, oder melden sich mit Systemstop, wenn es nichts mehr zu retten gibt. Der größte SINCLAIR - Konkurrent etwa kennt keinen Syntaxcheck, keine Tokeneingabe und keine selbsttätige GOTO-Zeilenumkorrektur. Letzteres ist beim SINCLAIR wie sein einzigartiger Befehlscheck ein echter Dienst am Kunden:

GOTO 10 springt zur nächst höheren Zeile, wenn keine Zeile 10 existiert

Auch der Zahlencheck bei INPUT "Zahl ?"; Z ist eine Aufmerksamkeit des Systems; aber es gibt noch weitere gleicher Art :

#### Stringaufteilung

Zeichenketten mit "unmöglichem" Aufteiler werden zum Leerstring, um keinen Schaden anzurichten. Beispiel :

PRINT IS (5 TO 2)

Das gilt auch für Index 0, wo ansonsten eine Fehlermeldung kommt: (N=0) PRINT IS ( TO N) Wird einem Kettenglied andererseits ein Leerstring zugewiesen, behandelt das System ihn wie Space (wie in allgemeinen Procrustes großen läßt !):

```
(AS=" ") LET IS(5) = AS
PRINT IS -> **** *
```

#### Print +

##### Plot Positionierung

Sowohl PRINT AT wie PLOT Y,X funktionieren auch bei gebrochenen Zahlen. Insbesondere bei der Umrechnung von Zahlenwerten in Strecken kann man sich die INT-Klausel sparen. Expressions sind wie allgemein zulässig, TAB bildet sogar automatisch den richtigen Modulus 31 zur Positionierung

##### Schleifen

Die universelle FOR NEXT-Schleife ist gegen Fehldurchläufe gesichert, falls eingangs Startwert > Begrenzer ist. Sie verhält sich wie eine abweisende DO WHILE-Schleife. Man kann die Laufbedingung bereits so formulieren, daß k sie nicht aktiv wird:

```
(Bedingung N = 0 -> kein Durchlauf)
FOR X = 1 TO 20 * (N<>0)....
```

Diese und weitere Dinge sind natürlich keine Medizin gegen Unachtsamkeit des Heimprogrammierers; gegen logische Fehler ist schließlich noch kein Kraut gewachsen.

#### Schwächen

##### Allgemeine Nachteile

Es wäre zu schön, wenn SINCLAIR-Basic ohne Fehler wäre. Zwar kann man sich u.U. Ersatz "stricken", aber an manchen Stellen



zeigt die Sprache Blößen. Das reicht von schmerzlich vermißten Instruktionen bis zu Schlampereien im ROM. Was alles möglich ist, bringen dann käufliche Erweiterungen (wie BETABASIC, auf das wir noch eingehen werden). Wie gesagt, manche fehlenden Befehle sind einfach zu ersetzen und werden oft überbewertet. Ein Beispiel:

```
ON ... GOTO/GOSUB(...)
Dazu ist eine ganze Skala von Surrogaten programmierbar, die
einfachste Form fanden wir für
ON X GOTO (100,80,94,122)
mit GOTO CODE ("dP-z")(X)
```

Als allgemein nachteilig wurde schon oft der umständliche Zeileneditor kritisiert. Er in Verbindung mit dem Syntaxcheck ist für die "anerkannte" Langsamkeit des Basic verantwortlich, sowohl bei Zeileneingabe bzw. -listen wie auch bei RUN. Es ist und bleibt ein unumstößliches Mandicap

SINCLAIR BASIC ist pfiffig aber lahm

Weitere allgemeine Nachteile sind keine vollausgebaute Grafik, unzureichende Blockfarbe, schwache Tönchenerzeugung u.a.m. Das alles sind Dinge, die heute zum Standard gehören, und an dem muß sich Sir Clive, wenn er denn überleben will, messen lassen.

#### Fehlende Befehle

Zum Standard gehören heutzutage Dinge, die einerseits den Instruktionssatz ergänzen, zum anderen die Programmierarbeit als solche erleichtern. Arbeitshilfen wie TRACE, RENUMBER etc. haben das Stadium von Zusatzsoftware oder Cluibtips verlassen. Gravierender sind hingegen Befehle, die SINCLAIR erst seinen QL-Superbasic vorbehalten hat; als da sind Selbstverständlichkeiten wie

```
IF THEN ... ELSE
DO WHILE/UNTIL + LOOP + REPEAT
DEF PROC + PROC-Aufruf
ON ERROR..
```

Gerade eine Brücke zur strukturierten Programmierung, die allenthalben versucht wird, fehlt. Was lediglich von einigen Kraut-und-Rüben-Programmierern nicht vermißt wird.. Nun aber zu echten Fehlern und Fallen des Basic, die den Heimcomputeristen zur Verzweiflung bringen können.

#### Fehler und Fallen

Nicht gemeint sind Flüchtigkeits- und (logische) Programmfehler. Natürlich sollte man bei unerklärlichen Ergebnissen und Crash zuerst vor der eigenen Haustür kehren. In 99 % der Fälle ist die Ursache eigene Dummheit oder Unachtsamkeit. Dennoch gibt es

Fallen vom System, die in erster Linie auf zwei Dingen beruhen

Ungenauigkeiten bei Floating-Point-Zahlen und ein nicht "entgratetes" ROM im SINCLAIR.

Leider kennt der Rechner keine INTEGER-Zahlen, mit denen sich viel Speicher und manche Unannehmlichkeit ersparen ließe (Wie man sie konstruiert und damit arbeitet, steht in Kapitel IV). Man hüte sich daher vor "Gleich" - Abfragen bei gleichzeitiger Rechnung mit vielen Stellen! Die Darstellung normaler Zahlen sieht intern ganz anders aus, der Computer rechnet mit Exponenten und Mantisse logarithmisch. Und da ist nach spätestens 8 Nachkommastellen die Genauigkeit weg. Wird viel multipliziert und dividiert, schaukeln sich Rundungsfehler weiter hoch. Man darf sich dann nicht über unmögliche Resultate wundern. Abhilfe? Selber Zwischendurch runden oder gleich die Zahlenbasis wechseln. Der Rechner hat es sowieso lieber, wenn ihm das E-Format vorgesetzt wird.

Besonders offensichtlich ist die Ungenauigkeit bei kleiner STEP-Größe. Man vermeide Werte kleiner 1 oder mit Nachkomma, die Zahl der Durchläufe ist ansonsten zufällig +/-1 der Vorgabe. Kein direkter Fehler des Systems, wenngleich durch seine Arbeitsweise bedingt, ist die GOSUB-Gefahr. Gemeint ist wiederholtes "gewaltsamen" Verlassen einer Sequenz vor RETURN. Wer aus einem Modul, statt es ordentlich zu beenden, mit GOTO rausspringt, riskiert langsames Verstopfen des Memory mit "liegengelassenen" Rücksprungadressen. Auf einmal ist 'out of memory' und Schlimmeres da..

Auf Banalitäten wie falsch angelegten ifs (letztes Zeilenstatement, sonst wird alles danach als "Ja"-Zweig gewertet!) wollen wir gar nicht erst eingehen. Was uns noch interessiert, sind echte ROM-Fehler. Da fragt sich mancher, liegt bloß der Kase im Pfeffer, die Logik ist doch ok?

ROM-Fehler sind naturgemäß von der jeweiligen Version des Rechners abhängig, jedoch haben sich die meisten bis zu Issue 3 "gehalten". Ian Logan, der als erster das ROM vollständig disassembelt (= Auflösung des Maschinencodes) hat, fand viele Bugs (=Systemfehler) unterschiedlichen Kalibers. Manches ist unbedeutend, anderes störend, einige Dinge sind schlichtweg ein Mergnis. Beschränken wir uns auf die letzten beiden Kategorien.

Zur Gattung "störend" gehören:

- + Bei rund 7000 MCs ROM sind Controllcodes für alle Richtungen außer "left" auf der Strecke geblieben. Aber selbst CHR 8 hat seine Tücken - positioniert man jenseits von AT 0/0, wandert der Cursor aus dem Display heraus. Es erscheinen dann die bekannten "grünen Männchen"..

- + Floatingpoint bei INT - 65536 kippt die Reihe um. Wird von der Minuszahl -1 subtrahiert, wird

plötzlich -1 ausgeworfen.

- + Direkter Andruck von Tokens (mit CHR\$ > 164) läßt manchmal führenden Spaces weg. Ähnlich wird das erste Zeichen gelegentlich bei direktem Stringendruck verschluckt; PRINT "X" + STR\$ 0.1 bringt nur "0.1", PRINT 1 + STR\$ 0.1 liefert "0.1".
- + Schließlich gibt es noch einige Ungereimtheiten beim Editieren. Man drücke nach "Scroll?"-Anfrage einmal CAPS + SYMBOLIC SHIFT..

Zur Kategorie  
"Argernis" gehören:

- + Auf das Konto Schlamperei bei Gleitkommazahlen geht die unglaubliche Division bei 1/2. Man vermeide den Ausdruck und ersetze ihn durch .5, was besonders für "Halbierungsrechnungen" gilt. Multiplikation geht sowieso im Rechner leichter.
- + Die Bildschirmabfrage mit der SCREENS-Funktion ist eine gute Sache. Man kann mit ihr sogar eine kleinen Textprozessor "bauen", indem immer angesteuerte Zeichen zurück geprintet werden. Aber Vorsicht! PRINT SCREEN (0,0) + SCREENS (0,1) ersetzt das erste durch das zweite vorgefundene Zeichen. Es wird statt dem ersten doppelt ausgegeben (Man muß eine ärgerliche Zwischenspeicherung vornehmen, um dem zu entgehen).
- + CLOSE-Versuche von ungeöffneten Streams lassen das Schlimmste eintreten - Crash. Wenn man Glück hat, kommt man mit einer wirren Fehlermeldung davon ..

Da wir schon mal beim famosen Microdrive und seinen Befehlen sind, - wir werden auf dieses Basic-Subset nicht weiter eingehen. Nicht weil die Syntax zu umständlich ist, sondern weil mit dem Drive keine richtige Verarbeitung von Daten möglich ist. Niemand außer ein paar Werbemannern von SINCLAIR wird das Speichermedium mit echten Floppys vergleichen; man darf nichts anderes als serielle Verarbeitung erwarten. Jedoch sollte wenigstens eine APPEND-Funktion vorhanden sein. Wir stufen das Drive aber aus einem anderen Grund zum schnelleren Ladegerät herab - es ist einfach zu unzuverlässig. Das sei aufgrund eigener Erfahrungen gesagt. Da es inzwischen gute Floppys, die mit MicroDrive-Befehlen laufen, gibt, kann keiner einwenden, eine Platte anzuschließen ginge technisch nicht..

## Wertzuweisungen

### Initialisieren und Default-Werte

In folgenden Abschnitt gehen wir die Sprache aus Sicht des praktischen Gebrauchs an. Wir werden zeigen, wie man effektiv programmiert, Besonderheiten sinnvoll einsetzt und systematisch die wichtigsten Statements fortentwickelt. Der Leser wird vieles finden, was nicht im Handbuch steht; das meiste ist Ergebnis umfangreicher eigener Erfahrung. So vorbereitet dürfte es dem aufgeweckten SINCLAIR-Programmierer nicht schwer fallen, den Rest des Buches nachzuvollziehen. Wir fordern ausdrücklich zum Mit- und Weiterdenken auf. Übung ist der beste Lehrmeister.

Unsere Reihe "aus der Basic-Trickkiste" beginnt dort, wo jedes Programm anfängt - beim Initialisieren. Das ist Setzen auf einen gewünschten Anfangszustand, bei Strings in der Regel Space/Leerstring, bei Zahlen meist Null. Andere Ausgangszustände sind möglich, etwa bei Begrenzern. - Am einfachsten hat man es bei Arrays, die initialisiert das System mit DIM bereits passend (Vorhandenen Variablen bringt CLEAR auf 'nicht vorhanden', was etwas anderes ist!). Einzelvariablen müssen solo erstbesetzt werden; es empfiehlt sich, Bezüge untereinander herzustellen, das spart Programmspeicher. Zum Beispiel so:

```
LET IS = "": LET U = NOT PI: LET X = U: LET Y = X  
DIM AS(5): LET BS = AS
```

Da Einzelvariablen mit Name anzusprechen sind, liegt die einfachere Anlage vieler Arrays nahe (wie man indirekt auch Felder mit ihrem Namen per VAL\$ ansprechen kann, kommt später). Aber Tabellen kosten Speicherplatz - a) weil auf die Maximallänge abzustellen ist, und b) weil Zahlen eh viel Speicher kosten. Es gibt sinnvollen Ersatz (s.u. Stringzahlen). Textarrays (etwa für Anzeigen, Beschriftungen) wird man einfach aus einem String laden. Folgende Befehlsfolge berücksichtigt sogar unterschiedliche Textlängen:

```
DIM NS(5,6): LET MS="MAENNL*WEIBL*ERW*JUGL": LET A=1: LET B=A  
FOR N=A TO LEN MS  
IF MS(N)<>" " THEN NEXT N  
LET NS(A) = MS(B TO N-1): LET A = A + 1: LET B = N + 1: NEXT N  
LET MS = ""
```

Beim Initialisieren mit den üblichen Anfangsgrößen haben sich ein paar speicherarme Klauseln bewährt, die auf der speziellen Vergleichslogik des SINCLAIR fußen (dazu später noch Genaueres):

Null ist gleich NOT PI oder PI - PI

Eins bekommt man mit SGN PI  
Zwei ist LEN "  
Drei wird durch INT PI gewonnen  
32 ist mit CODE " " einfach zu erzeugen  
usf.

Sogenannte Defaultwerte (= Werte, die gelten, solange nichts anderes bestimmt wird) lassen sich auf unterschiedlichen Wegen besetzen. Voraussetzung ist im Gegensatz zum Initialisieren, daß die Variable bereits existiert. - Am einfachsten ist die Wertbestimmung mit Null und Eins; hier kann ein logischer Vergleich bemüht werden:

LET X = X OR NOT X => X wird 1, wenn X = 0, sonst ungeändert

Soll X bei Null mit Y besetzt werden, modifiziert man etwas:

LET X = X + (Y AND NOT X)

Eine direkte Festwertzuweisung ist genau so einfach (hier 255):

LET X = X + 255 \* NOT X

Defaultwerte bei Grenzwertüberschreitung sind häufiger und komplizierter; X soll 255 bekommen, wenn X > 255:

LET X = X AND NOT 255 \* (X > 255)

Häufig muß der Fall abgefangen werden, daß bei No Input der alte Wert stehen bleiben soll. Auch das geht mittels 'logical expression':

LET X = X AND NOT (VAL IS AND LEN IS)

Allerdings ist die Frage berechtigt, ob es nicht einfacher gewesen wäre, gleich mit IF zu fragen:

IF IS <> "" THEN LET X = VAL IS

Vielseitiges VAL

Die Extensionen von VAL und dem weithin ungebrauchten VALS sind kaum bekannt. Mit der Konvertierung (von Zahlen im String) kann man vieles anstellen, einschließlich der nur so möglichen Stellenabfrage von Zahlen. Auch der Unterschied zwischen VAL und VALS ist allein durchs Handbuch nicht zu eruieren. Arbeiten wir zuerst das heraus, ehe wir auf so interessante Dinge wie indirekte Feldnamensbestimmung (per VALS) eingehen. VAL gibt den numerischen Wert, der als Zeichenfolge in einem String steht, VAL kann aber auch eine Vielfalt von Rechenvorschriften verarbeiten (wie eine 'function'). VAL AS

gibt ferner den Inhalt einer Variablen an, wenn der String sie als Name enthielt!

LET ZAHL = 1000: LET AS = "ZAHL": PRINT VAL AS bringt keinen 'error', sondern den Inhalt von ZAHL zutage -> 1000

VALS demgegenüber liefert das Ergebnis als Zeichenkette zurück, setzt bei Zahleninhalt in folgedessen Anführungszeichen voraus. PRINT VALS "AS" ist gleichbedeutend mit PRINT AS, druckt also "ZAHL" (als Zeichenfolge). PRINT VALS würde den Fehler "Nonsense in Basic" provozieren.

VAL hat den größeren praktischen Nährwert. Was kann man mit diesem Sprachelement alles anstellen? Nun eine ganze Menge z.B. die direkte Formeleingabe als Rechenvorschrift. VAL erledigt nicht nur direkte Berechnungen, sondern arbeitet auch Variablen und Festwerte ab:

LET IS = "A \* 5 + SQR B"

Mit der Zuweisungsform sind auch Iterationen sogar bei numeric arrays zulässig:

LET AS = "": LET SUM = 0 : DIM A(5) => mit Werten gefüllt...

FOR N = 1 TO 5: LET AS = AS + "A(" + STR\$ N + "): NEXT N  
LET SUM = VAL AS: PRINT "SUMME = "; SUM

VAL ist schließlich die einzige Chance, die erste Stelle einer Zahl als Wert direkt abzufragen:

LET STARTZIFFER = VAL (STR\$ ZAHL)(1)

Demgegenüber kann VALS etwas anderes - die Adressierung nach Feldnamen. Das dürfte ein einmaliges Feature sein, ist es doch in Basic ansonsten unmöglich, Namen zu qualifizieren. Die folgende Demonstration druckt je nach Eingabenummer entweder String AS, BS oder CS an:

LET AS="HIER DER ERSTE": LET BS="ICH BIN NR 2": LET CS="NR 3"  
LET XS="ABC": INPUT "STRINGNR ?";N: IF NOT N AND N>3 THEN STOP  
LET ZS= VALS (XS(N)+"\$"): PRINT ZS

Ähnliche Dinge kann man mit DATA anstellen, das Input gleichsam aus dem Basicprogramm liefert.

Input von Programm - DATA

Mit DATA läßt sich nicht allein ein Festwertspeicher im Programm selber installieren, auch die Benutzung mit READ als Zeiger bietet viele Möglichkeiten. Wie sonst nur bei der freien Zeichenkette können in der DATA-Tabelle Werte beliebigen Typs gesammelt werden. Will man sich so etwas wie einen Formelspeicher anlegen, können in DATA ein oder mehrere Variablen benutzt werden. Auch Arrays und anderen Datentypen

lassen sich einbringen und sogar mischen. Zuerst der Formelspeicher:

```
DATA A+1, A*2, A^2, A/2
```

```
LET A = 8: INPUT "STELLE ?": S: IF NOT S OR S>4 THEN STOP
FOR N = 1 TO S: READ A: NEXT N
PRINT "A BEI FORMEL ";S;" = ";A
```

Nun die Typenmischung und der "umgedrehte" Array:

```
DIM A(3): LET A(1)=123: LET A(2)= 234: LET A(3)= 345
DATA 3, A(3), A(2), A(1), "ENDE"
FOR N=1 TO 5: IF N=5 THEN READ XS:PRINT XS: STOP
READ X: IF N=1 THEN PRINT X;" ELEMENTE"
IF N>1 THEN PRINT "ELEMENT";N-1;" = ";X
NEXT N
```

Schließlich kann DATA eingesetzt werden, um eine der vielen ON .. GOTO-Surrogate zu stricken:

```
DATA 10,23,44,231,32
```

```
->Sprungqualifizierer in X
FOR N=1 TO X: READ SPRUNG: NEXT N
GOTO SPRUNG
```

Als nächstes wenden wir uns den Abfragen und Vergleichen zu. Wie bereits mehrmals angedeutet, ist die Verwendung SINCLAIRscher (logischer) Operatoren ein weites Feld für ausgefeilte Programmieretechnik. Insbesondere die IF-losen Abfragen werden uns zuerst beschäftigen. Nach ein wenig Systematik kommen wir dann zu einer Reihe arbeitserleichternder Tricks. ,

#### Abfragen und Vergleiche

##### Die Booleschen Operatoren

Die meisten Abfragen komplizierterer Art arbeiten mit der Verknüpfung von AND, OR und haben das Entscheidungskriterium gleich, größer nebst jeweiliger Negation. Der Bedingungstest endet dann mit dem Vergleichsergebnis "wahr" oder "falsch". Das sind sozusagen die 'Urzustände' der Logik. Da SINCLAIR-Basic dem Wahrheitstest die Wertigkeit 1 = true und 0 = false zuordnet, kann man auch von sog. Booleschen Operatoren sprechen. Ihr Einsatz macht das übliche IF in vielen Fällen entbehrlich. Man braucht nur das Vergleichsergebnis an die THEN-Aktion zu koppeln, um IF-lose Abfragen zu realisieren.

Eine Besonderheit des SINCLAIR-Basic ist es, den Bedingungstest

auch für Zahlen direkt anzuwenden. IF ZAHL steuert den JA-Zweig an, wenn ZAHL <> Null ist; umgekehrt wird THEN aktiv, falls mit IF NOT ZAHL auf Null getestet wurde. Soviel zu den Voraussetzungen Boolescher Operatoren.

Die praktische Bedeutung reicht äußerst weit; es können nämlich 'Wahrheitstests' für alle Datentypen und deren Kombination angestellt werden. Bei relativ wenig Regeln lassen sich durch Boolesche Operation

- gleiche Datentypen mit einander verknüpfen
- die ungleichen Brüder String + Zahl mischen
- jede Zahl und jedes Zeichen auf Null bzw. Leer testen.

Das waren die Funktionen, nun die Regeln für logische Operatoren. Es sind nur wenige.

X AND Y ergibt X, wenn Y true oder Y <> 0  
ergibt 0, wenn Y false oder Y = 0  
Ergo - "True" bestätigt X

X OR Y ergibt 1, wenn Y true oder Y <> 0  
ergibt X, wenn Y false oder Y <> 0  
Ergo - "False" bestätigt X

Für Zeichen und Typmischungen ist nur der AND - Vergleich zulässig:

XS AND YS ergibt (Leerstringtest!) XS, wenn YS <> ""  
ergibt "", wenn YS = ""

XS AND Y ergibt XS, wenn Y true oder Y <> 0  
ergibt "", wenn Y false oder Y = 0

Das Ganze kann noch mit NOT kombiniert werden und negiert dann die Operatoren. Ergänzt wird die Logik durch den IF - Test; damit hat man das Set Boolescher Operatoren komplett:

IF X THEN... -> Aktion, wenn X <> 0  
IF NOT X THEN... -> Aktion, wenn X = 0

IF XS THEN... -> Aktion, wenn XS <> ""  
IF NOT XS THEN... -> Aktion, wenn XS = ""

Natürlich lassen sich IF-Direktabfrage und Operatoren miteinander kombinieren, Operatoren verketteten und beliebig lange Ausdrücke bilden. Die Sache wird dann aber so kompliziert, daß man sie nur mit einer Wahrheitstabelle durchschaut. Und noch was - logische Operatoren kosten Laufzeit. - Wie kann man damit arbeiten?

Wir werden eine ganze Reihe von Anwendungen bringen, das meiste findet der Leser im Programmteil dieses Buches. Es lassen sich summarisch drei Einsatzschwerpunkte für mehr praktisch orientierte Programmierer ausmachen:

# 1. Direktverarbeitung des Vergleichsergebnisses

```
IF X THEN ..
LET X = NOT X (Null/Eins-Umschalter)
```

## 2. Bestätigung von Rechengrößen

```
LET X = X * (A = B) + Y * (A <> B)
GOTO START + 100 * (IS=1) + 220 * (IS=2)
```

## 3. 'Verifikator' mit AND/OR

```
INPUT ("TEXT" AND T); IS
LET N = N OR NOT N (Überlaufcheck auf Null)
```

Wichtig sind noch die Wertigkeiten bei kombinierten Ausdrücken. Es gibt nämlich Prioritäten - NOT vor AND vor OR. Letztere arbeiten abhängig von ihrer Stellung im Ausdruck. Eine Klammerung, wie sie hier steht, ist also entbehrlich:  
IF X>5 OR (Y>5 AND Z<3) THEN...

Bei 'expressions' ist hingegen die Rechnung vom AND/OR-Operator zu trennen:

```
GOTO (100 AND A<10) + (200 AND A>5)
```

Gleichwertig mit AND ist das gestaffelte IF, um der echten Bedingungsprüfung auch mal das Wort zu reden:

```
PRINT (AS AND A<10 AND Z=1) ist gleichbedeutend mit
IF A<10 THEN IF Z=1 THEN PRINT AS
```

Es gibt noch einen Nachteil IF-loser Abfragen (!), mit dem wir uns anhand konkreter Programmhilfen beschäftigen werden: Fürs komplette Abprüfen, also auch des Nein-Zweigs, muß die Bedingung 'negativ' wiederholt werden. Das wird deutlich an einem Beispiel; zuerst die IF-Formulierung als "Klartext":

```
IF A<10 THEN GOTO 100
GOTO 222
```

Nun als Boolescher Operator

```
GOTO (100 AND A<10) + (222 AND A>5)
```

Was wiederum vereinfacht werden kann

```
GOTO 100 + (122 AND A>5)
bzw. GOTO 100 + 122 * (A>5)
```

## IF-loose Abfragen

In loser Folge sollen praktische Anwendungsfälle behandelt werden, um zu zeigen, was man alles (ohne IF und damit oft kürzer und kompakter) mit Booleschen Operatoren anstellen kann. Es ist ratsam, diesen Abschnitt insgesamt durchzuarbeiten, sonst

Fällt das "Programmverständnis" der Kapitel II - IV schwer. Falls nötig werden wir die IF-Alternative als "Klartext" mit bringen.

## Prompts (Anzeigen)

IF-loose Abfragen haben viele Vorteile auf ihrer Seite. Das wird bei INPUTs deutlich, wo der Normalweg so aussehen wird:

```
INPUT "WAHL ?"; I : IF I<>1 AND I<>2 AND I<>6 THEN STOP
```

```
IF I = 6 THEN PRINT "FIRMA" ...
IF I = 2 THEN PRINT "NAME" ...
IF I = 1 THEN PRINT "ANSCHRIFT" ...
```

Nun kürzer und als 'sicherer' Stringinput:

```
INPUT "WAHL ?"; IS
PRINT ("FIRMA" AND IS="6"); ("NAME" AND
IS="2"); ("ANSCHRIFT" AND IS="1")
```

Damit werden gleich vier Vorteile gleichzeitig wahrgenommen:

1. Sicherere Eingabe (durch String IS)
2. Leerausgabe bei unsinnigem Input
3. nur eine Zeile Länge
3. einfachere Lösung

## Schalter

Schalter sind Markposten, die im Programm Auskunft über einen Zustand oder ein (vorheriges) Ergebnis geben. Der Möglichkeiten gibt es viele, z.B. ob ein Up- oder Down- oder mit Fehler endete. Man kann das Ergebnis mit Sprung und Nachricht direkt verbinden:

```
LET OK = 1
GOSUB 100 < bei Fehler wird OK = 0 >
IF NOT OK THEN GOSUB FEHLER
PRINT "VERARBEITUNG "; ("NICHT " AND NOT OK); "OK"
```

Ein/Ausschalter 'alternieren', sprich wechseln bei jedem Durchlauf von 0 auf 1 und umgekehrt. Sie sind recht nützlich, etwa um Texte abwechselnd auf "Bright" zu setzen (s. HEIMBUDGET):

```
LET BR = 0
FOR N = 1 TO 12
PRINT BRIGHT BR; M$(N);
LET BR = NOT BR
NEXT N
```

## Zuweisungen

Wertbesetzungen lassen sich genauso 'schalten'. Die Bedingungsprüfung aktiviert den Operanden und weist ihn einer Variablen zu. - Gesetzt den Fall, wir wollen eine

Statistik manchen für die Sparten "Spiele", "Utility", "Compiler" und "Anwendung". Der User gibt den ersten Buchstaben und noch so was wie die Cassettennummer ein. Die Anzahl N wird jeweils in Datenarray D(1..4) aufsummiert. Konventionell ist die Wertzuweisung so programmiert:

```
INPUT "SPARTE ?";SS;" ANZAHL ?";N
```

```
IF SS = "S" THEN LET D(1) = D(1) + N
IF SS = "U" THEN LET D(2) = D(2) + N
IF SS = "C" THEN LET D(3) = D(3) + N
IF SS = "A" THEN LET D(4) = D(4) + N
```

Nun mit Booleschen Operatoren, die die Einzelwerte per Index "anknipsen":

```
LET I = (SS="S") + 2*(SS="U") + 3*(SS="C") + 4*(SS="A")
IF I THEN LET D(I) = D(I) + N
```

Selbstverständlich kann die "Logik" auch bei Sprüngen angewandt werden. Allerdings ist das Geschwindigkeitsargument nicht von der Hand zu weisen. Wenn es bei Iterationen um viele Durchläufe geht, ist die FOR-NEXT-Schleife zweifellos schneller, welcher der folgende Abschnitt gewidmet ist. Ansonsten kann man bei SINCLAIR's Logik überspitzt sagen - mit "Boole" ist kein Ding unmöglich..

#### Sprünge und Schleifen

##### Berechnete Sprungziele

Statt lange IF-Ketten aufzubauen, kann man Sprünge als zweithäufigstem Basic-Befehl auch berechnen. Fast immer gibt die sgn. Destination (= Zielzeile) etwas her, um sie direkt mit dem Vergleich zu verbinden. Schließlich werden kluge Heimcomputeristen ihr Programm bereits in 'Blöcken' aufbauen, die sich leicht adressieren lassen. Der direkt berechnete Sprung greift auf die Programmnummerierung etwa so zurück:

```
< Zeile 50 ANFANG, Zeile 100 VERARBEITUNG, Zeile 150 ENDE >
```

```
CLS: PRINT "'M E N U E '""1-INIT""2-VERARB""3-SCHLUSS"
INPUT "WAHL ->NR ";N
IF N > 0 AND N < 4 THEN GOSUB 50*N
GOTO..
```

Schwerer ist es bei 'krummen' Zeilennummern. Erste Idee - eine Sprungtabelle muß her. Um beim Beispiel zu bleiben, Einrichten

des Arrays und Abfrage würde dann so aussehen:

```
DIM S(3): LET S(1) = 50: LET S(2) = 100: LET S(3) = 150
```

```
IF N > 0 AND N < 4 THEN GOSUB S(N)
```

Die Idee der Tabelle läßt sich variieren. Warum eigentlich ein numeric array? Eingedenk der Tatsache, daß in SINCLAIR-Basic nichts so flexibel ist wie ein "offener" String, betreiben wir etwas Werttransformation. Bei Sprungzielen im Codebereich 0 - 255 kann direkt auf den ad hoc-String qualifiziert zugegriffen werden

```
.. GOSUB CODE ("2d"+chr$(150))(N)
```

Bei größeren Zeilenzielen ist entweder ein Offset hinzuzuschieben oder aber man arbeitet mit dem bekannt vielseitigen VAL. Wenn die Zeilennummern also 100, 200 und 275 lauten, ist das kein Beinbruch. Der richtige VAL-Sprung (und damit auch ON N GOTO - Ersatz) ist so herzustellen:

< Memo: Es werden dreistellige Zeilenadressen vorausgesetzt, sonst Längenmodifikation >

```
.. GOSUB VAL "100200075"(N*3-2 TO N*3)
```

Nun, ein Sprung ist eine einmalige, nicht nach fester Reihenfolge wiederkehrende Sache. Was macht man bei Iterationen? Da wird solchermaßen Selbstgestricktes kaum sinnvoll sein. Hier spielt der Faktor Zeit die entscheidende Rolle. Lohnt es sich, Schleifen (mit GOTO) selbst zu verwalten? Nein, der Basic-Befehl FOR..NEXT ist in jedem Fall schneller. Was kann dabei trickreich programmiert werden? Welche Schleifenvariationen jenseits des Üblichen gibt es auf dem Weg voller Sprachausnutzung?

##### Schleifenvariationen

Eine Besonderheit des SINCLAIR ist es, daß das System keinen Stack (= Adress-Stapel) bemüht, um mit FOR NEXT zu arbeiten. Das gestattet den "unsauberen" Ausstieg aus der Schleife. Auch die Verwendung, ja sogar Veränderung der Laufvariablen ist zulässig, wenngleich Vorsicht geboten ist (Die Stepgröße hat nach Start der Schleife keinen Einfluß mehr auf die Schrittweite; sie kann bedenkenlos weiterverwendet werden). Zuerst aber ein "sauberer" Ausstieg, ehe wir die Variationen diskutieren. Grundanliegen geordneten Verlassens ist, nach Erreichen des Iterationsziels weitere Leerläufe zu vermeiden. Nehmen wir einen einfachen Suchprozess, eine Tabelle (AS(ENDE)) wird mit dem Suchwort (BS) verglichen. Da N bei "nicht gefunden" immer auf Begrenzer + 1 steht, kann Laufvariable N bei "gefunden" einfach auf > E + 1 gesetzt werden. So hat man einerseits den "Ausstieg" erreicht, andererseits ein Kriterium für "FOUND" geschaffen; das



ist besser, als ein Merkmal zu setzen und/oder die Schleife weiter laufen zu lassen...

Die Sache hat so allgemein formuliert einen Haken, man weiß nachher nicht, wo der Suchbegriff gefunden wurde. Die Information ist fürs Weiterarbeiten oft von großer Wichtigkeit. Lösung - entweder wird ein "Merker" bemüht oder man "mißbraucht" die Laufvariable N hierfür. Das ist in folgender Miniroutine geschehen.

```
FOR N = 1 TO ENDE
IF AS(N) = BS THEN LET N = ENDE + 1 + N
NEXT N: LET N = N - ENDE - 1
IF N THEN PRINT BS; " FOUND AT ";N
```

SINCLAIR-Basic läßt wie gesagt den unsauberen Ausstieg aus FOR NEXT zu; er ist in folgenden kaschiert enthalten. Eigentlich verbirgt sich dahinter ein GOTO etwa so

```
IF AS(N) = BS THEN GOTO ... <FOUND>
```

Unsere Routine erledigt das ohne GOTO indirekt gleichermaßen.

```
FOR N = 1 TO ENDE
IF AS(N) <> BS THEN NEXT N
IF NOT (ENDE + 1 - N) THEN PRINT "FOUND AT";N
```

Hätte man nicht den Vergleich Suchargument - Array am Ende wiederholen können? Falsch gedacht, wenn "nicht gefunden" steht N auf ENDE + 1, also einem nicht definierten Arrayelement. Die fatale Folge wäre ein 'subscript wrong' und unprogrammierter Stop. Was schlaue Leute auf die Idee gebracht hat, alle Suchtabellen mit einem Element mehr als nötig zu definieren! Dann allerdings funktioniert dieses:

```
... IF BS = AS(N) THEN PRINT "FOUND AT ";N
```

Eine andere Sache ist die generelle Möglichkeit, die Laufvariable zu manipulieren. Wenn man bei Update das bereits "entwischte" Element halten will, kann durchaus in der FOR NEXT - Schleife stehen

```
... LET N = N - 1: NEXT N
```

Daß auch Laufvariablen untereinander in geschachtelten Schleifen Aufsetzpunkte abgeben können, ist wohl klar (vgl. Kapitel IV, Sortverfahren). Sehr zupass wird dem gewieften Heimprogrammierer die relative Dimensionierung von Schleifen kommen; beispielsweise

```
FOR N = A TO B STEP 32
FOR M = N TO N + 32
```

```
..
NEXT M
NEXT N
```

Fällt dem Leser nichts auf? Was sollen die Einrückungen? Das Bild erinnert an strukturierte Hochsprachen wie PASCAL. Das Abgesetzte soll die Lesbarkeit erhöhen, die verstellte Logik ist durch Einrückung kenntlich gemacht. Wir haben alle Listings im Buch so wiedergegeben - unser Betrag zur Strukturierung. Da wir beim Thema sind, kann man in Basic nicht sign. logische Konstrukte - da nicht in der Semantik enthalten - nachbilden? Man kann durchaus; die beiden grundlegenden, DO WHILE und REPEAT UNTIL, sind mit FOR NEXT tatsächlich zu rekonstruieren. Es geht auch mit GOTO, ist aber weniger schnell und widerspricht dem Prinzip zeilenloser Programmierung. (Des dritte Konstrukt - die CASE-Unterscheidung - ist in etwa mit dem ON GOTO vergleichbar). Zeigen wir abschließend, wie man mit der FOR NEXT-Schleife so was wie "Struktur" ins Basic bekommt.

DO WHILE (als kopfgesteuerte Schleife)

FOR N = 0 TO 1	< Bestandteile >
LET N = NOT (BEDINGUNG)	DO WHILE
IF (BEDINGUNG) THEN	(BEDINGUNG)
(AKTION)	.
NEXT N	WEND

Nun das andere Konstrukt, die fußgesteuerte Schleife (Memo: sie wird mindestens einmal durchlaufen):

REPEAT UNTIL	REPEAT
FOR N = 0 TO 1	.
(AKTION)	.
LET N = (ABR.BED.)	UNTIL (ABBRUCH-
NEXT N	BEDINGUNG)

Mit dieser das eigentliche Basic überschreitenden Schleifenkonstruktion sei das Gebiet einzelner Befehlsgruppen verlassen. Zum Ausklang des Abschnitts noch ein paar Tips zur optimierten Programmierung, ehe eine besondere Basic-Erweiterung das Kapitel beschließt.

## Programmoptimierung

### Speicher sparen

Weniger Programmaufwand und schnellere Laufzeiten sind sich in der Regel ausschließende Gegensätze. Nur ganz vereinzelt kann beides zugleich realisiert werden. Die folgenden summarischen Tipps sind daher auch unter dem Aspekt zu sehen, daß dadurch ein Programm langsamer wird. Was besonders für Zahlenersatz gilt..

#### 1. Füllstoff raus

Man entferne alle REMs (manche Utilities besitzen dafür eine REMKILL-Funktion), unnütze Optik, Abstandszeilen und erklärende Prints. Möglichst viele Statements pro Zeile sparen nicht nur Zeilennummern, sondern bekommen auch der Programmschnelligkeit.

#### 2. Kurze Namen

Variablenamen sind je kürzer, je besser. Am besten arbeitet man mit so wenig wie möglichen numerischen Variablen; Mehrfachverwendung spart Platz, vieles läßt sich mit logischen Vergleichen direkt erledigen..

#### 3. Zahlen meiden

Zahlen sind Speicherfresser, am meisten geht das ins Memory bei numeric arrays. Wenn schon Zahlen, dann nicht als Programmkonstanten im Listing, sondern durch 'expressions' (z.B. statt LET X = 1 -> LET X = SGN PI oder LET X = VAL "1") ersetzen. Oft kommt man mit Codes oder Stringzahlen zurecht (z.B. PRINT AT 1,32;"Spalte"; 32 wird ersetzt durch PRINT AT VAL "1",CODE " ";;"Spalte 32").

#### 4. Rechenaufwand klein halten

Viel Boolesche Logik entlastet den Computer von aufwendiger Recherei, manche Klammern oder Zahlenberechnungen sind vermeidbar (z.B. Let (C\*5) + (D/1000) geht a) ohne Klammer und ist b) durch Multiplikation machbar -> ..+ D\*1E-3)

#### 5. Textökonomie betreiben

Texte sollten, falls gleichlautend und wiederkehrend im Programm, in einer Zeichenkette gesammelt werden. Man kann in single strings vieles voll variabel ablegen. Wie man so Sprungziele, Menutexte etc. verwendet, wurde schon gebracht; Programm KUERZEL zeigt weiter hinten, wie man durch Pointer das Ganze gut verwaltet. Markierer, Drucksteuer codes und anderes können mit eingebunden werden.

### Speeding up (Schnellermachen)

Sicher, Basic ist im allgemeinen langsam (als Interpreterversion), SINCLAIR Basic im Besonderen. Ein Grund mehr für clevere Meisprogrammierer, auf Geschwindigkeit zu achten. Die folgenden Maßnahmen sind zumindest bei der

"Hochglanzversion" eines Programms anwendbar; wenn man schon den Aspekt 'speed' nicht dauernd im Hinterkopf hat. Jeder Tip bringt allein nicht viel, aber es läppert sich mit der Zeit (!).

#### 1. Häufige Module und Variablen an den Anfang des Programms.

#### 2. Wo es geht FOR NEXT statt gleichertiger GOTOs.

#### 3. Statements kompakt halten - also viele Befehle in eine Zeile, Prints mit Druckkomma und TAB zusammenfassen, direkte Indizierungen (statt 100 mal LET X = V(I + 1) den Index vorweg mit + 1 definieren), Variablen aufeinander beziehen statt viel zu rechnen usw.

#### 4. Extensive Funktionen meiden - DEF FN und FN-Bezüge kosten enorm Zeit, Gleiches gilt für symbolische Adressierung (eher was für Anfänger -> GOSUB LESEN, GOTO STEUERUNG) oder berechnete Sprünge (wie GOTO 100+EINGABE); man springe lieber exakt die 'Arbeitszeile' an (nicht das erklärende REM).

#### 5. Schließlich - weithin unbekannt - RANTOP auf tatsächlichen Programmfumfang herabsetzen..

Auf dem Weg zur Hochsprache

Maschinencode ist nicht jedermanns Sache

Bei aller Flexibilität, über die SINCLAIR-BASIC verfügt, über kurz oder lang wächst der Wunsch nach mehr. Beispielsweise wird mancher Spectrum-Fan wünschen, es gäbe einen FILL-Befehl zum Ausfüllen ganzer Flächen oder ein RENUMBERING. AMSTRAD/SCHNEIDER und andere Heimcomputer verfügen über solches. Möchte man sich derartige Wünsche erfüllen, bleibt nur der Weg der Maschinencodierung. Was konkret bedeutet, - um mehr BASIC zu bekommen, muß man in Niederungen der Elementaroperationen herab steigen. Maschinencode ist nicht allein SINCLAIR's Produkt, sondern prozessorabhängig. Der Z80 A des Spectrum hat eine eigene Mnemonik, die sich aus ca. 700 Befehlskürzeln zusammensetzt.

Einzelroutinen bleiben Stückwerk

Es bedarf umfangreicher Systemkenntnisse, viel Erfahrung und großer Disziplin, in sgn. Assembler zu programmieren. Möchte man zudem aus Arbeitsvereinfachung 'vorprogrammierte' ROM-Routinen seiner Maschine nutzen, ist erhebliches Wissen, was "den SINCLAIR im Innersten zusammenhält", vonnöten. Maschinencode - Routinen werden denn auch allerorten in Zeitschriften und darauf ausgerichteter Literatur angeboten. Wer sich nicht schämt, unbesonnen fremdes Gedankengut zu nutzen - oft mit "bugs" (Fehlern) behaftet - , kommt auf seine Kosten. Die Sache hat nur einen Pferdefuß, von den Kosten und der Mühe des Eingabens abgesehen: die verschiedenen Routinen passen in den seltensten Fällen zueinander. Nur wenige MC-Programme sind allozierbar, benutzen also nur relative Adressen, und können überall hin im Hauptspeicher geladen werden. Besser ist folglich eine Lösung aus einem Guss. Natürlich gibt es auch solches. Da werden "im Paket" ganze Tool-, Screen- und Grafik-Kits angeboten, bei denen diese Funktionen zusammengefaßt sind. Aber das alles ist im Grunde, wenn man eine umfassende Erweiterung seines BASIC anstrebt, Stückwerk. Vereinzelt gibt es zwar größere Spracherweiterungen, die lassen jedoch fast immer Wünsche offen. Ist es folglich vermessene, nach "mehr BASIC aus einem Guß" zu fragen? Keineswegs.

Wer das SINCLAIR-BASIC auf den Level einer (strukturierten) Hochsprache heben möchte, sei auf eine Erweiterung aus England verwiesen - B E I A B A S I C. Das Erzeugnis ist auch in Deutschland erhältlich, die neueste Version 3.0 1987 kaum noch Wünsche offen. Das gilt insbesondere für PASCAL ähnliche Kontrollstrukturen wie DO WHILE, DO UNTIL, DO..UNTIL LOOP, IF..THEN ..ELSE, ON..Statement, Statement (= CASE-Äquivalent).

Die Basic - Spracherweiterung wird als Maschinencode eingeladen und belegt etwa 19 K RAM Speicher. Mit vielen neuen, erweiterten Befehlen wird der Spectrum weit über seine normalen Grenzen hinaus geführt. Kurzum - mit BETABASIC ist der SINCLAIR anderen Heimcomputern haushoch überlegen.

Die wichtigsten Schwerpunkte

Ein Außenseiter zeigt, was möglich ist

Da es Anliegen dieses Buches ist, auch Spracherweiterungen zum SINCLAIR-BASIC aufzuzeigen, sei auf BETABASIC näher eingegangen (Übrigens - das neue Basic kann außer über den deutschen Vertrieb auch direkt in England für 15.50 Pfund bestellt werden; wer eine ältere Version 1.2 oder 1.8 besitzt, bekommt fürs Upgrade einen Rabatt. Adresse: BETASOFT, 92 Oxford Road, Moseley, Birmingham, B13 9 5D).

Die Schwerpunkte der zahlreichen Hilfen, Befehlsweiterungen und Unterstützungsfunktionen lassen sich wie folgt klassifizieren. Damit ist im folgenden keineswegs alles, was BETABASIC kann, abgehandelt. Aber bei allein 88 Seiten eines vorzüglichen, mit vielen Beispielen angereicherten Manuals muß die Zusammenfassung notgedrungen schwerpunktartig bleiben.

Strukturierung

Angenehm von der Optik fällt zunächst die Wahl eines LIST - Formats, das der Programmstruktur angepaßt ist, auf. Jedes Statement erhält eine eigene Zeile, FOR-NEXT-Schleifen werden eingerückt ebenso wie IF-THEN-Statements etc. Wichtiger ist die Blockstruktur, wie sie PASCAL, C und anderen Hochsprachen bieten. Die sgn. Blockorientierung wird neben anderem durch Prozeduren erreicht, wobei neben den Referenzparametern lokale, d.h. vom Hauptprogramm unabhängige, Variable notwendige Voraussetzung sind. BETABASIC realisiert dies mit einfachem Prozeduraufruf. Kontrollstrukturen wurden vollständig integriert; Befehle wie DO WHILE, DO UNTIL, DO .. LOOP UNTIL, ON .. (entspricht CASE), IF THEN ELSE u.a. ermöglicht ein BASIC ohne GOTO! BETABASIC kommt dem entgegen, indem bei Listendruck eine Option ohne Zeilennummer gewählt werden kann.

Grafik

Zunächst enthält die Kassette neben den üblichen Scrolls, Rollis, Inscreen-Bewegungen etc. ein separates LOGO alikes Paket mit der bekannten TURTLE-Grafik. Ein Extra (im Preis enthalten), das gänzlich von BASIC unabhängig macht und einfachste Grafikprogrammierung ermöglicht. Nun zu den Basicbefehlen: einsems "Spitze" ist das WINDOW-Feature. Bis zu 128 (!) Fenster können angelegt, verwaltet und sogar in Überlappung auf den Schirm gebracht werden. Jedes Fenster

erhält unabhängig Farbattribute, Schriftgröße und Koordinatengrenzen. Es kann einfach angesprochen, gecleart und gescrollt werden, da eigene Controlcodes geschaffen wurden. Letztere lassen sich auch in normalem Text verwenden, wie auch jeder String pixelgenau platzierbar ist...

Das ist noch lange nicht alles. Screenteile sind in Strings abgespeicherbar, Bildschirmteile im Hauptspeicher ableg- und abrufbar. Figuren können mit FILL gefärbt, Farbattribute ohne Neuzeichnen des Screens geändert werden. Das weiteren wurden die vorhandenen Befehle (DRAW, OVER, PRINT mit USING etc.) in nicht unerheblichen Maße erweitert.

#### Toolkit-Funktionen

Neben dem verbesserten Editing fällt besonders angenehm die schon angesprochene Wahl diverser LIST-Formate auf. Man fragt sich, wieso nicht eher jemand auf das saubere Einrücken, abgesetzt von der Zeilennummer, oder den flashing cursor verfallen ist. Was woanders laut gepriesen wird - wie AUTO- und RENUMBERING mit COPY- und DELETE -, ist in BETABASIC Selbstverständlichkeit. Demgegenüber sind die zahlreichen Referenzen, die abgerufen werden können, bemerkenswert. Man kann eine Liste aller, nur der numerischen oder der String-Variablen anwählen. Jede mit aktuellem Wert und Zeile des Vorkommens. Dazu tritt die TRACE-Funktion. Sie ermöglicht bei Tests Zeilenandruck - als Nummer oder in Klartext -, Variablenausgabe und anderes mehr. Es wird dazu nach Erreichen der Programmzeile eine selbstdefinierte Routine angesteuert, die entsprechend "bestückbar" ist. Sogar single stepping (mit PAUSE 0) ist möglich.

#### Arbeitsvereinfachungen

Unter die Rubrik Arbeitsvereinfachungen fallen Dinge wie JOIN und SPLIT von Substatements (innerhalb und zwischen verschiedenen Zeilen). Auch das Durchsuchen, Ersetzen und Modifizieren von Variablen bzw. Inhalten ist kein Problem. Weiter gehts mit wichtigen Maßnahmen, die die Basic-Arbeit vereinfachen, zum Beispiel beim Datenhandling.

Die Fülle der Möglichkeiten, die sich im Zusammenhang mit Strings und Arrays ergibt, kann nur angetippt werden. Da ist einmal die INSTRING-Suche, gekoppelt mit Ersetzung oder Löschung des Inhalts. Umbesetzungen per BETABASIC-Befehl, Zusammenfügen und Verkürzen von Strings lassen die Verarbeitung von Daten zum programmatischen Kinderspiel werden. Arrays sind leicht zu sortieren (200 Elemente in weniger als 1 Sekunde!) - wahlweise auf- oder absteigend. Weiter kann der Variablenbereich eines Programms insgesamt separat gespeichert und rückgeladen werden; ein Vorteil bei Programmen, die mit unterschiedlicher Datenausstattung gefahren werden sollen. Man kann sogar - wenn man es will - den gesamten Programmbereich im RAM hin- und herschieben (wie bei sgn. RAM-Disks) oder als Code auf Band/microdrive abspeichern.

Wer es schließlich leid ist, bei Listings oder Datenprint an Sinclairs 32 Zeichen pro Zeile gebunden zu sein, wählt die passende Zeichengröße (mit CSIZE) aus; entweder bis 64 Z/Z (wie bei TASWORD) oder beliebig groß.

#### Beurteilung

##### Weniger Speicher und ein paar "Bugs"

Man fragt sich, ob es bei alledem nicht auch Nachteile von BETABASIC gibt. Es gibt sie, aber es sind wenige.

Gravierendster ist der hohe Speicherbedarf, der auf Kosten des RAM-Bereichs geht. Mit Nullzeile ist rd. 19 K des kostbaren RAM verloren; eine ROM-Lösung wäre zweifellos besser gewesen. Damit im Gefolge ist zu beachten - ab Adresse 47271 ist mit Version 3.0 von BETABASIC alles dicht. Eigene MC-Routinen müssen neu ausgerichtet werden, DEVPAC und Ähnliches geht nicht mehr.

Wer nicht unter BETABASIC geschriebene Programme mit der Spracherweiterung warten will, kann Ungewohntes erleben. Strings oder Funktionen ändern im Listing ihr Aussehen (ohne die Lauffähigkeit einzuschränken): aus MS wird MEMORY, aus IS wird TIME uaf. Das Handbuch erkennt dies an - wie auch eventuelle Schwierigkeiten mit Printsteuerzeichen -, und gibt Hilfestellung. - Noch ein Merkposten bei Nicht-BETABASIC's: Programme sind zur Basicerweiterung hinzu zu mergen. Was bei mehrfachem Tun Zeilenwirrwarr verursachen kann.

Kein Zeilenwirrwarr aber Farb- und Printchaos kann das Arbeiten mit überlappenden Windows bringen. Man muß beim Positionieren schon höflich aufpassen, um ein sauberes Bild zu erzielen. Ein falsches Bild ergibt sich vom System auch ungewollt bei LIST und LIST DATA, d.h. Andruck aller Variablen mit Wert. Die Listzeilen sind fehlerhaft, auf dem ZX-Drucker streikt BETABASIC bereits nach der ersten Zeile; ein echter Bug also.. Ob es sich um einen solchen bei ALTER Variable handelt, ist fraglich; immerhin weist das Manual darauf hin, daß mit Ändern des Variablenamens keine Löschung der alten Programmgröße verbunden ist. Eine Sache, die als Nachteil zu betrachten ist.

#### Zusammenfassung

Mit BETABASIC werden Wünsche wahr, welche die Frage nach Grenzen des "Beginners all purpose symbolic instruction codes" aufwerfen. BETABASIC hebt das Sinclair-Basic auf den Level einer modernen blockorientierten Hochsprache. Es macht aus dem Spectrum eine neue Maschine mit ungeahnten Möglichkeiten (Beispiel: KEYIN bringt ein Programm dazu, sich selber fortzuschreiben...). Entscheidendes Handicap ist lediglich der Speicherbedarf von fast 19 K RAM; eine ROM-Lösung wäre besser.

gewesen. Zwar kann der Mangel an verfügbarem Speicher gemildert werden (z.B. durch programmierten Ersatz aller Zahlen im Programm mit VAL "..."- s. Manual, Seite 15), wird jedoch schwerlich durch die leistungsstarken neuen Befehle aufgehoben.

Die Sprachmächtigkeit von BETABASIC droht anfangs zu erschlagen; am einkängigsten sind noch die Elemente einer PASCAL ähnlichen Strukturierung. Viele nützliche Hilfen machen das Arbeiten mit der Erweiterung leicht. Man sollte auch bedenken, daß man sich mit zunehmender Gewöhnung immer mehr vom normalen BASIC-Standard entfernt.

Insgesamt ist BETABASIC fast eine neue Sprache für den Spectrum. Mit ihr schlägt er die Konkurrenz um Längen.

## Kapitel II: Grafik

### ZAHLENDARSTELLUNG

#### Monatstage optisch aufbereitet (KALENDER)

##### Aufgabenstellung

In diesem Kapitel werden wir uns mit den grafischen Möglichkeiten des SINCLAIR beschäftigen - angefangen von Blockgrafik und einfacher Zahlendarstellung bis hin zu ausgefallenen Bildschirmtricks. Grundlage für Grafik und Ausnutzung der Hochofflösung (HiRes abgekürzt) ist es, Daten optisch zu aufzubereiten. Grafik heißt, vereinfacht ausgedrückt, Zahlen in Strecken umzuformen. Von Spielereien und "Kunst" abgesehen liegt Kurven, Skalen, Flächen etc. immer eine Vorschrift, meist ein Algorithmus, zugrunde. Das bedeutet Programmarbeit, trotz bequemer BASIC-Befehle wie DRAW, CIRCLE u.ä. Wenden wir uns zuerst dem Einfachsten zu - der Verwendung von Blockzeichen und der Platzierung von Zahlen auf dem Bildschirm. Die Grafik bei einem einfachen Monatskalender zeigt sich in der Gestaltung des "Rahmens" und richtig positionierten Monatstagen. Sie sollen vertikal fortlaufend erscheinen und gemäß den Wochentagen (von Montag bis Sonntag) korrekt eingetragen sein. Ein bescheidenes Ziel, aber der vom Computer stammende Kalender ist gewiß nicht ohne Reiz.. Um die Monatstage nach dem betreffenden Jahr stimmig auszurichten, sind zwei Aufgaben zu lösen, die der mathematische "Vorlauf" von Programm KALENDER erledigt:

1. die Monatsendtage (28/29/30/31 Tage) sind zu berechnen - Problem Schaltjahr
2. für die Wochentage ist der Jahresstart maßgebend, danach müssen sie korrekt fortgeschrieben werden

Was darüber hinaus zu tun bleibt, ist die Darstellung - u.z.

- a) die Aufbereitung des Monatsrahmens (Beschriftung, "Optik")
- b) Ausrichtung der Tage mit (invers) hervorgehobenen Sonntagen

##### Lösungsbestandteile

Erläutern wir die Lösung, wie sie im Folgenden Programm KALENDER bringt. Gehen wir die einzelnen Teile durch.

Die Initialisierung (Zeile 10 - 48) fragt zunächst "unzeitgemäße" Jahreszahlen ab und errechnet Schaltjahre auf der Basis 1984 (letztes Schaltjahr). Die Fortrechnung der Monatsenden geht von 28 Tagen aus (Zeile 46); darauf werden die

Distanzen zum Monatsdatum ausgerichtet. Bis auf den Februar - soweit kein Schaltjahr - kommt die Differenz 2 oder 3 in Monatsstring MS. Der Januar bekommt eine 3 (31 Tage = Basis 28 + 3), der April mit 30 Tagen eine 2 (Basis 28 + 2) und so fort. Eine Schleife (in Zeile 48) errechnet und speichert die Monatszahlen als Codes im MS. Dort stehen sie im direkten Zugriff zur Abfrage bereit.

Vorbereitet wird der Monatsprint mit Einsetzen des Startwochentages eines Jahres (von User). Von da an zählt KALENDER die Wochentage fort, um sie für einen bestimmten Monat bzw. das ganze Jahr auszudrucken. Was passiert im einzelnen hierzu?

Wird ein bestimmter Monat angewählt (s. Zeile 53), arbeitet die folgende Zweifachschleife erst die Monate von Jahrbeginn ab; im Innern (Zeile 58 ff.) werden die Wochentage fortgeführt. Vorher ist für die Druckoperationen Spaltenzähler SP jeweils auf Anfang gesetzt worden. Ist der Wahlmonat erreicht, wird der Jahreswochentag (mit "Sonntagsüberlauf") fortgezählt und spaltengerecht gedruckt. Möglich macht es eine trickreiche Zeilenpositionierung: Wochentagszähler JA dient gleichzeitig als Zeilenzähler. Ist ein Montag erreicht, beginnt der Spaltendruck wieder von oben. Durch die vertikalen Monatstage kommt es zur Kolonnenbildung, wie man bei Kalendern gewohnt.

Das übrige Layout (Zeile 200 - 210) wird in einem Unterprogramm generiert. Die Aufbereitung besorgt eine Schleife (FOR NEXT mit N), welche Rahmen und Wochentage nebst Monatsnamen erzeugt. String IS enthält dazu erst den "Balken", dann die Wochtagskürzel. Sie werden - einmal geladen - nach Andruck abgehakt (Zeile 204); das ist die einfachste Methode, einen String in gleichen "Stücken" ohne viel Rechnerlei abzuverarbeiten. Solange der gewählte Monat (noch) nicht da ist, wird im Upward lediglich mit READ (Zeile 201) der DATA-Pointer vorgeschoben. Die DATA's am Schluß von KALENDER (Zeile 220) bergen die Monatsbezeichnungen als Konstanten. - Abschließend die Screencopy eines Kalenderblatts, das Programm und etwas Programmbeschreibung.

AUGUST		1985			
MO		5	12	19	26
DI		6	13	20	27
MI		7	14	21	28
DO	1	8	15	22	29
FR	2	9	16	23	30
SA	3	10	17	24	31
SO	4	11	18	25	

```

6 REM *****
7
10 REM *****INITIALISIEREN*****
20 LET JA=0
30 REM MONATSENDETAG
40 INPUT "JAHR";JAHR
41 IF JAHR<1900 OR JAHR>2000 THEN
42   GOTO 40
43 LET Z=ABS(1904-JAHR)
44 LET Z=Z/4=INT(Z/4)
45 INPUT "MS=";"STR$ Z+"32323323
46 FOR N=1 TO 12
47   LET MS(N)=CHR$(VAL MS(N)+
48     (N-1)*7)
49 NEXT N
50 REM MONATSPRINT
51 IF JA=0 THEN INPUT ("WOCHEN
TAG 1..JAN..1..JAHR);"HILFE:1=
MO,2=DI,3=MI,4=DO,5=FR,6=SA
52 IF JA<1 OR JA>7 THEN LET JA
=0
53 INPUT "JAHRMONAT=ZAHLE=GAN
ZES JAHR";JAHR
54 RESTORE 12 THEN LET UM=0
55 FOR M=1 TO 12
56   CLS
57   FOR N=1 TO CODE MS(M)
58     IF UM=M OR UM=0 THEN PRINT
59     AT 4+JA*2,SP+(N-10);IN
VERSE (JA=7);N
60     LET JA=JA+1
61     LET SP=SP+4
62   NEXT N
63   SUB 200
64   NEXT M
65 INPUT "FOLGEJAHR ? JA=J";IS
66 IF IS="J" THEN LET JAHR=JAHR
67   CLS
68   GOTO 42
69 REM *** E N D E *****
70
100 REM >>UNTERPROGRAMM<<<
101
190 REM LAYOUT
200 IF UM=0 THEN GOTO 202
201 IF UM<M THEN READ TS
202 LET TS=" "
PRINT AT 0,0;TS;AT 4,0;TS;A
T 20,0;TS;AT 5,5;" "
LET TS="MODIMIDOPR$ASO"
203 FOR N=1 TO 19
204   PRINT AT N,0;" ";AT N,31;"
NEXT N
205 FOR N=6 TO 19 STEP 2
206   PRINT AT N,2;TS( TO 2);"
207   AT N+1,5;" ";
208   TS=TS(3 TO )
NEXT N
209 READ TS
210 PRINT AT 2,3;TS;TAB 24;JAHR
211 INPUT "HARDCOPY ? U=JA/RETU
RN=NEIN/STOP";IS
212 IF IS="J" THEN COPY
213 IF IS="S" THEN STOP
214 RETURN

```



```
220 DATA "JANUAR", "FEBRUAR", "MARZ", "APRIL", "MAY", "JUNI", "JULI", "AUGUST", "SEPTEMBER", "OKTOBER", "NOVEMBER", "DEZEMBER"
```

#### Programmbeschreibung

#### Programmaufbau:

##### 1. Hauptteil

Zeile 10-48 Initialisieren - Bestimmen Schaltjahr + Monatstage  
50-68 Monatsprint - Ausrichten Monatsanfang, Drucksteuerung und Andruck Monatsdatum

##### 2. Upros

Zeile 200 Layout 'Kalenderblatt' - Zeichen Rahmen, Beschriftung Wochentage + Monatsbezeichnung  
210 ten Wochentage + Monatsbezeichnung  
220 DATA-Festwerte = Monatsbezeichnungen

#### Variablenliste

JAHV Vorgebejahr  
JA Wochentag des 1. Tages von JAHV  
N Schleifenvariable (Druck + Monatstageszähler)  
M Zähler Monate  
SP Spaltenzähler  
WM Wahlmonat  
Z Zählvariable  
  
IS User-Input  
MS Monatstage (als Codes)  
IS Druckstring

#### Skulengrafik

##### Zahlen in HiRes

HiRes bedeutet beim SINCLAIR Über 255 x 175 Pixels auf dem Bildschirm zu gebieten. Nach der "primitiven" Blockgrafik - wie bei ZX 81 - wollen wir das volle Spectrum hoch auflösende Grafik entfalten. Neben dem Umdenken von "ausgetauschten" Spalten/Zeilen wie bei PRINT AT ist der Screen auf die höhere Auflösung auszurichten.

Hauptproblem bei Zahlen auf dem Screen ist, Werte in Strecken

umzurechnen. Damit das Diagramm als getrautes Abbild von Werten in den gegebenen Rahmen paßt, ist ein Darstellungsmaßstab zu gewinnen. - Die Horizontale bereitet die geringsten Schwierigkeiten; es sind lediglich Abstände für die Werteanzahl einzuteilen und ein paar Bezeichner anzubringen. Bei Zeitreihen stehen auf der X-Koordinate etwa Monate, Quartale, Jahre oder sonstige Data. Mittels VERTIKAL-Schrift (s. Kapitel III) kann die Achse gedreht und so Platz für Kurven unbegrenzter Länge geschaffen werden. Wählt man die herkömmliche Form, kann kostbarer Pixelbereich "in Y" durch Abkürzungen auf der X-Achse geschaffen werden. Die Y-Achse ist schwerer zu beherrschen. Hier spielt sich das eigentliche Feld der Grafikerzeugung ab. Verbleibende X-Koordinaten engen den Raum ein, umfangreiche Wertumrechnungen sind vorzunehmen, Maßstäbe sind zu entwickeln und alles ist gegen Überlauf zu sichern. Das wird uns in folgenden beschäftigen; zeigen wir erst einmal, wie einfache Skulengrafik auf Screen und Papier zu bringen ist.

Wir bleiben bei Monaten und gehen von einer simplen Umsatzstatistik in Skulenform aus. Was nicht so schwierig wie Fläche (sgr. Tortengrafik) ist, aber schon anspruchsvoller als Kurven auf dem Bildschirm. Immerhin enthält Programm "Skulengrafik" schon so interessante Dinge wie Jahreszahlen und automatische Justierung nach 'range' (= Wertebereich zwischen einem Minimum + Maximum). Wegen der Skulenbreite sind nur die Monate Januar bis Oktober dargestellt, wählt man kleinere X-Abstände, geht ein volles Jahr auf den Bildschirm. Aber gehen wir der Reihe nach vor.

#### Kernproblem Darstellungsmaßstab

Kernproblem ist die Eichung der Vertikalen (= Y-Achse). Die genaue Erörterung der Problematik findet sich in MEINBUDGET; hier soll lediglich der Weg zur "Skule" dargelegt werden. Wir halten uns an die Maxime - Schritt für Schritt vom Einfachen zum Anspruchsvollen.

Die Umrechnung von tatsächlichem Wert zur Pixelstrecke beinhaltet drei Maßnahmen:

1. Abprüfen der 'range' auf Höchst/Tiefstwerte (bei wiederkehrenden Daten können sie am Kopf des Wertearrays abgespeichert werden)
2. Umrechnen der Distanzen zum Maxi/Minimum als Darstellungsfaktor (etwa 10 Werteeinheiten ergeben 7 Pixel)
3. Generieren der Y-Werte (als Vertikalstriche) von Koordinatenursprung aus und Fortschreiben der Horizontalachse

Zusätzlich sind die "Skulen" voll auszumalen und ggf. mit Farbe zu versehen. Weitere Funktionen wie SAVE/LOAD etc. vervollständigen ein solches Grafikprogramm. - Wie nun löst

"Säulengrafik" das aufgezeigte Kernproblem ?

Als erstes sind zwei Felder MIN und MAX mit umgekehrten Maximal- bzw. Minimalwerte zu belegen (vgl. Zeile 122). So kommen immer echte Höchst/Tiefstwerte bei Durchgang aller Daten herein. Letztere werden entweder direkt oder per LOAD eingelesen (s. Zeile 130 - 132). Nun kann der Darstellungsmaßstab für eine Einheit errechnet werden. Konkret heißt das im Programm (s. Zeile 135):  
Konstante f = Differenz Max-Min geteilt durch 151 Pixel (Plotstrecke).

Die Darstellung im Koordinatenbereich beginnt mit dem Aufsetzen auf der X-Achse. Der Säulenstrich wird bis zum aktuellen Wertrepräsentanten hoch gezogen (vgl. Zeile 436 ff). Den wieder liefert Funktion FN g(). Wie ?  
Aktueller Arraywert a(i + 1,x) abzüglich min ergibt die relative Distanz. Sie mal Darstellungsfaktor f erbringt die aktuelle Pixelzahl, welche von der X-Achse zu ziehen ist. Das ist alles..

Natürlich gehört zu "Säulengrafik" noch mehr, z.B. richtige Menusteuerung und optisch ansprechende Ausgabe der Zahlen mit Vergleichswert (Vorjahr). Bei der Grafik selber ist zu beachten, daß aktuelle Monatswerte immer als gefüllter Balken erscheinen, während der Vergleichswert als Schatten daneben steht. Das Ganze ist wie bei richtigen PC's aufgezogen. Mehr an Optik - außer noch "higher resolution" - können die auch nicht bieten. - Abschließend das Programm selber und die notwendigen Angaben zum Lesen desselben.

```

10 REM Säulengrafik
12
14 DEF FN g() = 4 + (a(i+1,x) - min)
16 *f
17 LET d$ = "JANFEBMARZAPRMAIJUNJ
18   URAGUSEPOTK"
19 REM Beispiel f. 10 Monate
20
21 REM Mini-Menu
22 CLS
23 PRINT "Säulengrafik"
24 PRINT "OPTIONEN: 1-PAPER
25   2-Werte eingeben/lade
26   3-Programme/Werte Save
27   4-Grafik generieren" (Re
28   turn=STOP)
29
30 INPUT "Bitte waehlen.."; LI
31 IF LI < 0 AND LI > 5 THEN G
32 O SUB 100 * UAL 1
33 IF LI < 0 THEN GO TO 18
34 STOP

```

```

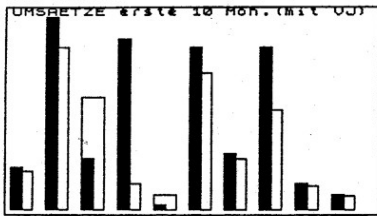
90 REM ***
100 INPUT "Anzahl (Doppel-)Wert
110 IF z < 1 OR z > 10 THEN GO TO 1
110 INPUT "Werte laden/angeben
111 IF b$ = "e" OR b$ = "l" THEN GO
112 TO 120
112 IF b$ = "l" OR b$ = "l" THEN IN
113 PUT "Ladenname array"; b$
114 IF b$ < " " THEN LOAD "a"; 1
115 DATA a()
116 GO TO 122
117 RETURN
118 DIM a(z,2)
119 LET c$ = ""
120 LET min = 999999
121 LET max = 0
122 FOR i = 1 TO z
123   FOR x = 1 TO 2 THEN LET w = a(i,x)
124   IF w < min THEN LET min = w
125   IF w > max THEN LET max = w
126 NEXT x
127 IF w < 0 THEN GO TO 125
128 LET a(i,x) = w
129 IF w > max THEN LET max = w
130 IF w < min THEN LET min = w
131
132 LET x = 2 OR x = 2
133 IF x = 2 THEN GO TO 125
134 NEXT i
135 LET f = 151 / (max - min)
136 RETURN
137
138 REM ***
139
210 INPUT "Save Programm/Array
211 IF b$ < "p" AND b$ < "p" AND
212 b$ < "a" AND b$ < "a" THEN GO
213 TO 210
214 INPUT "Name ?"; c$
215 IF c$ = "" THEN GO TO 220
216 IF b$ = "p" OR b$ = "p" THEN SA
217 VE "a"; 1; c$ LINE 10
218 RETURN
219 SAVE "a"; 1; c$ DATA a()
220 RETURN
221
300 REM ***
301
310 INPUT "Anzahl Werte geladen
311 /eingespeichert", "Wenn nicht=0
312 "; z
313 IF NOT z THEN RETURN
314 CLS
315 PRINT "Monat Umsatz Ugt,Uo
316 "
317
318 FOR i = 0 TO z-1
319   PRINT d$(i*3+1 TO 3*(i+1))
320
321 PRINT TAB 7; a(i+1,1); TAB 1
322   INVERSE 1; a(i+1,2)
323 NEXT i
324 INPUT "copy ?"; b$
325 IF b$ = "y" OR b$ = "y" THEN CO
326 PY
327 RETURN
328
400 REM *** Generiere die

```

```

410 CLS
PRINT "UMSATZE erste 10 M
412 PLOT 4.8
DRAW 0.5
DRAW 0.5
DRAW 0.5
DRAW 0.5
DRAW 0.5
DRAW 0.5
DRAW 0.5
DRAW 0.5
DRAW 0.5
DRAW 0.5
414 FOR i=0 TO z-1
416 FOR j=0 TO z-1
418 PRINT INVERSE x;AT 21,w;d$
420 NEXT j
422 NEXT i
424 REM DRAW-Umrechnen d.Werte
430 FOR i=0 TO z-1
432 FOR j=0 TO z-1
434 PLOT j,12
436 IF s(i+1,x) THEN DRAW 0,f
438 NEXT j
440 NEXT i
442 INPUT AT 0,0;"ScreenCOPY?="
444 IF s(1,0) THEN RETURN
446 IF c$="" THEN RETURN
448 IF c$="N" THEN RETURN
450 RETURN

```



Monat	Umsatz	Uel.Vorj.
JAN	0.0	0.0
FEB	0.0	0.0
MAR	0.0	0.0
APR	0.0	0.0
MAY	0.0	0.0
JUN	0.0	0.0
JUL	0.0	0.0
AUG	0.0	0.0
SEP	0.0	0.0
OCT	0.0	0.0

Angaben zum Programm

## Aufbau

Zeile  
10-17 Initialisieren  
20-26 MiniMenu  
100-140 Werteinlesen (per User oder LOAD)  
Array besetzen  
Min/Max bestimmen  
Darstellungsmaßstab festlegen  
200-242 SAVE Programm o. Array (MicroDrive)  
300-350 Tabellenendruck Datenarray  
400-424 Generieren Grafik:  
- Rahmen + Beschriftung zeichnen  
- PLOT Ausgangspunkt + DRAW d. Distanz  
pro Monat (jeweils f. akt. u. Vergleichswert)  
- Option ScreenCopy

## Variablen

s(z,2) Datenarray (z=Monate, 2 entsprechend  
f. akt. u. Vergleichswert)  
f Darstellungsmassstab  
i,j Zählvariablen  
min Minimum der Werte  
max Maximum der Werte  
x Umschalter f. akt./Vergleichswert  
w Inputwert  
z Anzahl Monate (max. 10)

b\$ Userwahl bei LOAD/SAVE  
c\$ Ladenname  
d\$ Monatskurztext  
is Menuwahl

## Funktion

FN g() Wertumrechnung Array-Wert in Plotpunkte

## BILDSCHIRMARBEIT

### Screenbewegungen

### Inscreen Copy

Mit PLOT und Draw sind die Grafikmöglichkeiten des SINCLAIR bei weitem nicht ausgeschöpft. Schon an Befehlen gibt es mehr - etwa CIRCLE und POINT. Wenden wir uns weiteren Dingen zu, die BASIC und System bieten; zeigen wir, wie man versteckte Fähigkeiten der Maschine aufspürt - angefangen von einfachen Bewegungen auf

dem Bildschirm bis zur Screen-Manipulation per Maschinencode. Letztere naturgemäß als Basic-Ergänzung nur, zur Einbindung ins Programm bestimmt. - Im zweiten Teil dieses Abschnitts werden wir gar ein komplettes Utility (= Hilfsprogramm) zur bequemen Bildschirmmalerei entwickeln. Es greift auf einige Routinen, die zuvor kommen, zurück. Fangen wir mit relativ einfacher "Bildschirmarbeit" an.

Oft ist unklar, was eigentlich mit dem Basic-Statement POINT anzustellen ist. POINT liefert die Information, ob ein Pixel an gegebener Koordinate vorhanden ist oder nicht. POINT ist so was wie eine boolesche Variable (wie in PASCAL und anderen Hochsprachen bekannt). Lautet die Screenuntersuchung JA (TRUE), so wird der Wert 1 rückgeliefert; ist kein Pixel gesetzt, schickt das Ergebnis den Wert 0 (FALSE). Es kann in SINCLAIR-Basic direkt mit IF abgefragt werden, wobei die verkürzte Form des "logischen" Vergleichs gewählt werden kann:

```
IF POINT (N,M) THEN PRINT "PIXEL BEI ";N;",";M
```

Wozu ist eine solche Information gut? Nun doch wohl, wenn man von einer bestimmten Stelle eine Kopie anfertigen will. Man braucht das Vergleichsergebnis nämlich bloß mit dem PLOT an anderer Stelle zu verbinden!

Zum Kopieren eines Screenbereichs ist folglich die Ursprungskoordinate pixelweise abzufragen und die Zielkoordinaten in den PLOT-Befehl einzusetzen. Natürlich wird man beide Bereiche synchron fortzählen, damit nichts verloren geht. Am einfachsten und sichersten geht das, indem mit einer Verschiebedistanz gearbeitet wird. Die folgende Routine geht so vor und ermöglicht auf einfache Weise eine "inscreen copy". Da in Basic sind an die Geschwindigkeit keine großen Erwartungen zu stellen.

Bekannt sein müssen Spalte/Zeile der zu kopierenden Box, sowie die Zielkoordinaten. Die Routine rechnet die Angaben um, wodurch die notwendige Verschiebung gewährleistet ist. Die erforderlichen Variablen sind:

Anfangsecke Box XA, YA; gegenüber liegende Eckpunkte XE,YE; Zielkoordinaten - entspricht der Anfangsecke - XZ,YZ (s. Zeile 22)

Im Programm ist ein Testtext direkt installiert (s. Zeile 26 - das Overprinten hat nichts zu bedeuten). Nach dem Fehlercheck wird (in Zeile 35) die Verschiebedistanz errechnet, ehe die Plotschleife (Zeile 40) loslegt. Ansonsten dürfte der Ablauf klar sein. Zur Demonstration ist eine Hardcopy des Ergebnisses mit angegeben.

```
10 REM INSCREEN- COPY
12
20 REM TESTSET
21 REM AUSGANGS- + ZIELKOORDIN
22 ATEN
23 LET XA=80
24 LET YA=70
25 LET XE=190
26 LET YE=140
27 LET XZ=30
28 LET YZ=60
```

```
25
26 PRINT AT 1,10;"INSCREEN COP
Y:AT 2,10;"ist in Basic"
T:0;10;"sehr langsam";CHR$ 6;OU
27
28
29 REM FEHLERCHECK
30 IF XA>XE OR YA>YE OR (XZ+
(XE-XA))>255 OR (YZ-(YA-YE)
)>0 THEN PRINT "FEHLER"
31 STOP
32
33 REM VERSCHIEBEDISTANZ
34
35 LET X=XA-XZ
36 LET Y=YA-YZ
37 REM COPY-PLOT
38
39 FOR M=YA TO YE STEP -1
40 FOR N=XA TO XE
41 IF POINT (N,M) THEN PLOT
N-X,M-Y
42 NEXT N
43 NEXT M
```

```
INSCREEN COPY
ist in Basic
sehr langsam

INSCREEN COPY
ist in Basic
sehr langsam
```

Doch gehen wir zu raffinierteren und vor allem schnelleren Screenbewegungen über. Dazu müssen wir einen kleinen Excursus ins System machen; das Ergebnis sind einige kurze Maschinencodes.

Screen Switch

Um Bildschirminhalte bzw. Teile davon zu verschieben, ist es nützlich, sich die Organisation des sgn. Display Files vor Augen zu halten. Als notwendige Angaben sei die Aufteilung des Bildschirmbereichs kurz wiedergegeben (Man werfe eine Blick in Handbuch, Stichwort SCREENS). - Der Screen ist in drei Bereiche aufgeteilt, wobei die Attribute getrennt hintereinander liegen. Der SCREENS-Befehl speichert so den gesamten Bildschirm ab und entspricht

SAVE (LOAD) "BILD" 16384,6912

Verzichtet man auf Farbe u.ä., bedarf es weniger Bytes, um alles abzulegen.

SAVE (LOAD) "BILD 0" 16384,6144

Damit wird das "nackte" Display File von Adresse 16384 - 22527 incl. genommen. Man kann einen weiteren Schritt zur Teil-Screen-Bewegung gehen, indem die unterschiedlichen Adressen der drei Teile benutzt werden. Was besonders interessant ist, die Bereiche können im RAM (geschützt) gehalten und blitzschnell hervorgeholt werden. Man besten geht das in MC, wobei nur wenige Bytes Code erforderlich sind. Auch Überlagerungen, Hervorheben verborgener Help-Seiten etc. ist möglich. Zuerst die erforderlichen Adressen:

Bereich	Adresse Screen	Attributadresse
TOP - Zeile 0 - 7	16384 - 18431	22528 - 22783
MID - Zeile 8 - 15	18432 - 20479	22784 - 23039
LOW - Zeile 16 - 23	20480 - 22527	23040 - 23295

Selbstverständlich lassen sich alle Teile getrennt speichern, verschieben und wieder hervorholen; wodurch schöne Überlagerungseffekte entstehen - etwa mit

SAVE "BILD 1" 16384,4896

Dadurch wird der obere und mittlere Teil gespeichert (ohne Attribute). Das reicht für Help-Seiten meist völlig, weshalb wir es für die folgenden MC-Routinen ausgewählt haben.

Um die Eingabe zu erleichtern und BASIC nicht ganz zu vergessen, ist das folgende Listing jeweils mit einem Hexloader versehen. Hexcodes sind besser, als mit vielen DATA-Zeilen zu arbeiten. Warum? Nun Codes in Dezimalform, die es einzupoken gilt, haben den Bereich 0 - 255, sprich maximal 3 x Tippen plus ggf. das Komma. In "hexa" sind nur 2 Zeichen (0-9 und a-f) einzugeben, die Umrechnung kann der Rechner besorgen. Faulheit im Umgang mit dem Computer macht erfinderisch... Gegen Fehleingaben kann man sich mit einer Prüfsumme (checksum) absichern, wie im 2. Teil der MC-Routinen noch zu sehen sein wird.

```

10 REM HEXLOAD (OHNE CHECKSUM)
12
13 DEF FN F(M)=CODE C*(M)-(48+(CODE C*(M)>57)*39)
15 INPUT "DATA-ZEILE ? ";N
20 INPUT "START F.MC ? ";START
22
25 RESTORE N
30 READ C*
```

```

35 FOR B=1 TO LEN C* STEP 2
40 LET E=FN F(B)*16+FN F(B+1)
44
```

```

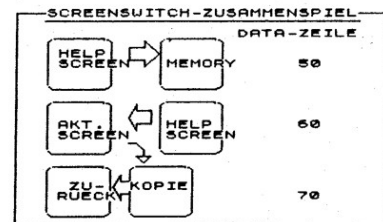
45 POKE START,E
LET START=START+1
NEXT B
46 PRINT "CODE GELADEN => DATA-ZEILE ";N
47 REM PARTIELLER SCREENSWITCH
48
49 REM LADEN HIDDEN SCREEN IN MEMORY
50 DATA "2100401100000100100db0c9"
58
59 REM akt.SCREEN <-> HelpSCREEN
60 DATA "2100401100000100100db02100f01100400100100db0c9"
68
69 REM switchback akt.SCREEN
70 DATA "2100001100400100100db0c9"
```

Wie arbeiten die ScreenSwitch-Routinen zusammen?

Die drei MC's bewirken aufeinander abgestimmt folgendes:

- laden des Bildschirms (obere 2/3 = Zeile 0-15) in den RAM-Speicher (Platz für 2 x 4096 Byte mit CLEAR am RAM-Ende schaffen!)
- Austausch des aktuellen Screens mit dem abgelegten (Help->Bildschirm, ersterer bleibt verborgen erhalten)
- Rückladen des aktuellen Screens, um weiter zu arbeiten (seine Kopie bleibt temporär vorhanden)

Der Vorgang wird vielleicht deutlicher, wenn man das folgende Schaubild betrachtet.



Nach "Installation" der Routinen wird man überwiegend mit den

49

Haupteigenschaften vor, ehe wir uns dem Lösungsweg widmen.

## Features

Das Draw-Utility soll Hilfe beim Zeichnen geben. Das heißt mehr, als ein paar Striche zu malen, vielmehr

- erzeugen von Strich, Box, Kreis
- löschen von Einzelpunkten oder mit breitem "Strich"
- Screenmanipulation per Scroll oder Inscreen Copy
- Bildschirme Laden und speichern
- Texte incl. Blockgrafik beliebig platzieren
- einfache, schnelle Bedienung mittels Status/Command-Zeile

Gerade letzter Punkt wird oft vernachlässigt. In DRAWUTIL turnt der User mit Tastendruck von Option zu Option; entweder mit Buchstabeninput bei "Anwahl" oder mit Space bei "Ebenenwechsel". Das sgn. Menu ist kein Tableau, das zwischendurch eingeblendet wird, sondern eine Steuerleiste im Inputbereich (Zeile 23 + 24). Durch Ausnutzen des Microdrive-Befehls PRINT # 0 bzw. # 1 bleibt (in Verbindung mit PAUSE 0 und INKEYS) die Steuerung immer sichtbar. Mit der Space-Taste kommt man von einer Funktion zurück auf die Hauptmenüleiste.

Auch die Cursor-Bewegungen wurden vereinfacht: alle Richtungen werden ohne "Shift" angesteuert. Außerdem ist Cursor kein unscharfes Zeichen, sondern ein präzises Fadenkreuz. Man kann zu Beginn eines jeden Menüzeiges zwei Geschwindigkeiten vorwählen; also einmal langsam, dann wieder schnell (4-fache Geschwindigkeit) auf dem Bild herumfahren. Mit "Space" (4-fache) eine Option aus. Diese (Draw, Un/plot, Circle, Scroll, Text) sind zudem recht flott - trotz Basic. Wie kommt das?

Nun einmal wurde die schnelle INKEYS-Abfrage eingesetzt, zum anderen ist vieles in SINCLAIR-typischer Vergleichslogik realisiert. Zeigen wir dazu ein paar Kniffe auf:

(Zeile 303 + 304) Die gesamte Cursorsteuerung incl. Überlauf-sicherung wird in nur zwei Zeilen erledigt

(Zeile 228) Auch die Berechnung der Verschiebedistanz plus Begrenzungscheck kommt mit OR-Vergleich statt vieler IF's zustande

(Zeile 28) Der Sprungtrick basiert auf der Errechnung des Buchstabencodes als direktes GOTO-Ziel

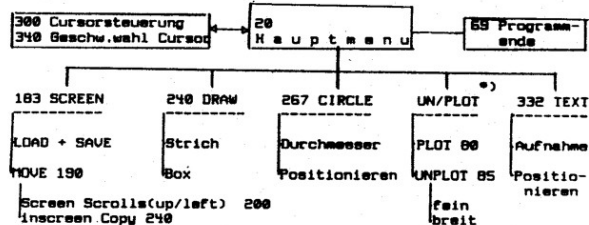
(Zeile 342) Hier ist der Variableninput angewandt; die Inputantwort S = Slow oder F = Fast ist bereits mit einer vorbelegten Variable (s. Zeile 17) vorbereitet. Gibt der User S oder F ein, erhält G (wie Geschwindigkeit) als Wert 1 bzw. 5

(Zeile 23 + 332) Die aufgerufene ROM-Routine setzt User-inputs jeweils von Groß- auf Kleinbuchstaben um (und umgekehrt); da für Text Kleinschreibung sein soll, wird der Menu-Buchstabe danach wieder auf "groß" gesetzt.

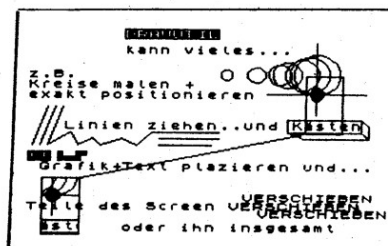
Genug der Tricks, wenden wir uns wieder dem Programm im Ganzen zu. Zeigen wir die Funktionen und ihr Zusammenspiel, ehe der Rest der Dokumentation folgt.

## Funktionen

Statt vieler Wort sei als erstes das Schema der Hauptfunktionen und ihre logische Untergliederung gebracht (Die Zahlen stellen die Startprogrammzeilen dar).



\*) Die Funktionen sind in die Cursor-Routine (2.300) integriert





- Start mit RUN, um die MC's zu laden; Restart ins Programm mit GOTO 20
- Rückspiegung auf die Steuerleiste und eine höhere Ebene mit SPACE, dito um zwischen durch den Cursor zu deaktivieren
- Cursor-Geschwindigkeit wird bei den Haupttexten vorgewählt und bleibt gesetzt; notfalls kann man mit UN/PLOT zwischen durch umschalten (Achtung: Antwort nur S oder G, sonst Systemstop !!)
- Plot geht pixelfein, Unplot löscht fein und breit (8 pixels)
- Kopieren innerhalb des Bildschirms bedarf der Festlegung Ausgangs- und Zielkoordinaten; der Bereich wird als Box kurz angezeigt, ehe die inscreen copy beginnt
- Text kann Grafikzeichen umfassen; die Pixelposition wird in Zeilen/Spalten (für PRINT AT) umgerechnet
- LOAD/SAVE des Screens sind in Microdrive-Syntax (eine Änderung für Tape ist einfach)

```

5 REM *****
6 UTILITY 2. BEARBEITUNG
7 D.BILDSCHIRM (SONST)
8 FUNKTIONEN, VER. SCHREIBEN (SONST)
9 DRUCKTEXT() INT (175-Y)/0.5
10 DRUCKTEXT() INT (X/8)+5
11 REM LADEN MC-ROUTINEN F.SCREEN
12 LOAD "M" SCROLL-LEFT
13 LOAD "M" ADDRESS, 32800
14 REM ADDRESS, 32800 LEFT CODE
15 PRINT AT 1; AT 0,0; FLASH 1;"
16 * FLASH 0;" OPTIONS/PRAMS="
17 * CURSORSTART="ENTER"
18 PAUSE 200
19
20 REM INITIALISIEREN
21 CLEAR
22 G=0
23 G=7
24
25 ORDRU 1 1024 7
26 ORDRU 2 256 7
27 ORDRU 3 16 7
28 CLEAR X=128
29
30 REM *****
31
32 PRINT "ALLOTAT 0,0;" SCREEN 1; "R
33 ALLOCATE OPTION 0; TEXTEND=
34 ;BUCHSTABE (SONST CURSOR)
35
36
37 LET S$="INKEY$
38 IF S$="" THEN RANDOMIZE US

```

[illegible]

```

221 LET S$="B"
222 OVER 1
223 DO SUB 322
224 DO SUB 322
225 IF INKEY$="" THEN OVER 0
226 RETURN
227 IF INKEY$="P" OR S$="U" THEN LE
228 LET X=XE+Y
229 LET Y=XE-Y
230 IF ((X(XE-XA))>250) OR ((Y
231 (YA-YE))>10) THEN PRINT #0;
232 "1. KOORD.FEHLER =";
233 PAUSE 10
234 GO TO 228
235 FOR MYA TO YE STEP -1
236 FOR N=XA TO XE
237 IF POINT(N,M) THEN PLOT
238 N-(X-X),M-(Y-Y)
239 NEXT N
240 RETURN
241 IF INKEY$="" THEN DRAW X-XA,Y-
242 YA
243 IF INKEY$="B" THEN DRAW X-XA,0
244 IF INKEY$="Y" THEN DRAW X-XA,Y-
245 IF INKEY$="X" THEN DRAW X-XA,0
246 IF INKEY$="A" THEN DRAW X-XA,Y-
247 RETURN
248 INPUT AT 0,0;"EINGABE"
249 IF INKEY$="" THEN RANDOMIZE USR
250 431
251 RETURN
252 PRINT #1;AT 1,0;"TEXTPOSIT."
253 DO SUB 300
254 IF LEN I$(31-FN S()) THEN
255 I$=I$ TO 31-FN S()
256 PRINT AT FN Z(),FN S()
257 PRINT I$
258 IF Y-YE*(Y)=12
259 GO TO 332
260 PRINT #0;AT 1,0;"CURSOR"
261 PAUSE 0
262 LET S$=INKEY$
263 RETURN
264
265 PLOT XA,YA THEN DRAW X-XA,Y-
266 YA
267 RETURN
268 IF INKEY$="B" THEN DRAW X-XA,0
269 IF INKEY$="Y" THEN DRAW X-XA,Y-
270 IF INKEY$="X" THEN DRAW X-XA,0
271 IF INKEY$="A" THEN DRAW X-XA,Y-
272 RETURN
273 INPUT AT 0,0;"EINGABE"
274 IF INKEY$="" THEN RANDOMIZE USR
275 431
276 RETURN
277 PRINT #1;AT 1,0;"TEXTPOSIT."
278 DO SUB 300
279 IF LEN I$(31-FN S()) THEN
280 I$=I$ TO 31-FN S()
281 PRINT AT FN Z(),FN S()
282 PRINT I$
283 IF Y-YE*(Y)=12
284 GO TO 332
285 PRINT #0;AT 1,0;"CURSOR"
286 PAUSE 0
287 LET S$=INKEY$
288 RETURN
289
290 OVER 1
291 DO SUB 322
292 DO SUB 322
293 IF INKEY$="" THEN OVER 0
294 RETURN
295 IF INKEY$="P" OR S$="U" THEN LE
296 LET X=XE+Y
297 LET Y=XE-Y
298 IF ((X(XE-XA))>250) OR ((Y
299 (YA-YE))>10) THEN PRINT #0;
300 "1. KOORD.FEHLER =";
301 PAUSE 10
302 GO TO 228
303 FOR MYA TO YE STEP -1
304 FOR N=XA TO XE
305 IF POINT(N,M) THEN PLOT
306 N-(X-X),M-(Y-Y)
307 NEXT N
308 RETURN
309 IF INKEY$="" THEN DRAW X-XA,Y-
310 YA
311 IF INKEY$="B" THEN DRAW X-XA,0
312 IF INKEY$="Y" THEN DRAW X-XA,Y-
313 IF INKEY$="X" THEN DRAW X-XA,0
314 IF INKEY$="A" THEN DRAW X-XA,Y-
315 RETURN
316 INPUT AT 0,0;"EINGABE"
317 IF INKEY$="" THEN RANDOMIZE USR
318 431
319 RETURN
320 PRINT #1;AT 1,0;"TEXTPOSIT."
321 DO SUB 300
322 IF LEN I$(31-FN S()) THEN
323 I$=I$ TO 31-FN S()
324 PRINT AT FN Z(),FN S()
325 PRINT I$
326 IF Y-YE*(Y)=12
327 GO TO 332
328 PRINT #0;AT 1,0;"CURSOR"
329 PAUSE 0
330 LET S$=INKEY$
331 RETURN
332
333 PLOT XA,YA THEN DRAW X-XA,Y-
334 YA
335 RETURN
336 IF INKEY$="B" THEN DRAW X-XA,0
337 IF INKEY$="Y" THEN DRAW X-XA,Y-
338 IF INKEY$="X" THEN DRAW X-XA,0
339 IF INKEY$="A" THEN DRAW X-XA,Y-
340 RETURN
341 INPUT AT 0,0;"EINGABE"
342 IF INKEY$="" THEN RANDOMIZE USR
343 431
344 RETURN
345 PRINT #1;AT 1,0;"TEXTPOSIT."
346 DO SUB 300
347 IF LEN I$(31-FN S()) THEN
348 I$=I$ TO 31-FN S()
349 PRINT AT FN Z(),FN S()
350 PRINT I$
351 IF Y-YE*(Y)=12
352 GO TO 332
353 PRINT #0;AT 1,0;"CURSOR"
354 PAUSE 0
355 LET S$=INKEY$
356 RETURN
357
358 OVER 1
359 DO SUB 322
360 DO SUB 322
361 IF INKEY$="" THEN OVER 0
362 RETURN
363 IF INKEY$="P" OR S$="U" THEN LE
364 LET X=XE+Y
365 LET Y=XE-Y
366 IF ((X(XE-XA))>250) OR ((Y
367 (YA-YE))>10) THEN PRINT #0;
368 "1. KOORD.FEHLER =";
369 PAUSE 10
370 GO TO 228
371 FOR MYA TO YE STEP -1
372 FOR N=XA TO XE
373 IF POINT(N,M) THEN PLOT
374 N-(X-X),M-(Y-Y)
375 NEXT N
376 RETURN
377 IF INKEY$="" THEN DRAW X-XA,Y-
378 YA
379 IF INKEY$="B" THEN DRAW X-XA,0
380 IF INKEY$="Y" THEN DRAW X-XA,Y-
381 IF INKEY$="X" THEN DRAW X-XA,0
382 IF INKEY$="A" THEN DRAW X-XA,Y-
383 RETURN
384 INPUT AT 0,0;"EINGABE"
385 IF INKEY$="" THEN RANDOMIZE USR
386 431
387 RETURN
388 PRINT #1;AT 1,0;"TEXTPOSIT."
389 DO SUB 300
390 IF LEN I$(31-FN S()) THEN
391 I$=I$ TO 31-FN S()
392 PRINT AT FN Z(),FN S()
393 PRINT I$
394 IF Y-YE*(Y)=12
395 GO TO 332
396 PRINT #0;AT 1,0;"CURSOR"
397 PAUSE 0
398 LET S$=INKEY$
399 RETURN
400
401 PLOT XA,YA THEN DRAW X-XA,Y-
402 YA
403 RETURN
404 IF INKEY$="B" THEN DRAW X-XA,0
405 IF INKEY$="Y" THEN DRAW X-XA,Y-
406 IF INKEY$="X" THEN DRAW X-XA,0
407 IF INKEY$="A" THEN DRAW X-XA,Y-
408 RETURN
409 INPUT AT 0,0;"EINGABE"
410 IF INKEY$="" THEN RANDOMIZE USR
411 431
412 RETURN
413 PRINT #1;AT 1,0;"TEXTPOSIT."
414 DO SUB 300
415 IF LEN I$(31-FN S()) THEN
416 I$=I$ TO 31-FN S()
417 PRINT AT FN Z(),FN S()
418 PRINT I$
419 IF Y-YE*(Y)=12
420 GO TO 332
421 PRINT #0;AT 1,0;"CURSOR"
422 PAUSE 0
423 LET S$=INKEY$
424 RETURN
425
426 OVER 1
427 DO SUB 322
428 DO SUB 322
429 IF INKEY$="" THEN OVER 0
430 RETURN
431 IF INKEY$="P" OR S$="U" THEN LE
432 LET X=XE+Y
433 LET Y=XE-Y
434 IF ((X(XE-XA))>250) OR ((Y
435 (YA-YE))>10) THEN PRINT #0;
436 "1. KOORD.FEHLER =";
437 PAUSE 10
438 GO TO 228
439 FOR MYA TO YE STEP -1
440 FOR N=XA TO XE
441 IF POINT(N,M) THEN PLOT
442 N-(X-X),M-(Y-Y)
443 NEXT N
444 RETURN
445 IF INKEY$="" THEN DRAW X-XA,Y-
446 YA
447 IF INKEY$="B" THEN DRAW X-XA,0
448 IF INKEY$="Y" THEN DRAW X-XA,Y-
449 IF INKEY$="X" THEN DRAW X-XA,0
450 IF INKEY$="A" THEN DRAW X-XA,Y-
451 RETURN
452 INPUT AT 0,0;"EINGABE"
453 IF INKEY$="" THEN RANDOMIZE USR
454 431
455 RETURN
456 PRINT #1;AT 1,0;"TEXTPOSIT."
457 DO SUB 300
458 IF LEN I$(31-FN S()) THEN
459 I$=I$ TO 31-FN S()
460 PRINT AT FN Z(),FN S()
461 PRINT I$
462 IF Y-YE*(Y)=12
463 GO TO 332
464 PRINT #0;AT 1,0;"CURSOR"
465 PAUSE 0
466 LET S$=INKEY$
467 RETURN
468
469 PLOT XA,YA THEN DRAW X-XA,Y-
470 YA
471 RETURN
472 IF INKEY$="B" THEN DRAW X-XA,0
473 IF INKEY$="Y" THEN DRAW X-XA,Y-
474 IF INKEY$="X" THEN DRAW X-XA,0
475 IF INKEY$="A" THEN DRAW X-XA,Y-
476 RETURN
477 INPUT AT 0,0;"EINGABE"
478 IF INKEY$="" THEN RANDOMIZE USR
479 431
480 RETURN
481 PRINT #1;AT 1,0;"TEXTPOSIT."
482 DO SUB 300
483 IF LEN I$(31-FN S()) THEN
484 I$=I$ TO 31-FN S()
485 PRINT AT FN Z(),FN S()
486 PRINT I$
487 IF Y-YE*(Y)=12
488 GO TO 332
489 PRINT #0;AT 1,0;"CURSOR"
490 PAUSE 0
491 LET S$=INKEY$
492 RETURN
493
494 OVER 1
495 DO SUB 322
496 DO SUB 322
497 IF INKEY$="" THEN OVER 0
498 RETURN
499 IF INKEY$="P" OR S$="U" THEN LE
500 LET X=XE+Y
501 LET Y=XE-Y
502 IF ((X(XE-XA))>250) OR ((Y
503 (YA-YE))>10) THEN PRINT #0;
504 "1. KOORD.FEHLER =";
505 PAUSE 10
506 GO TO 228
507 FOR MYA TO YE STEP -1
508 FOR N=XA TO XE
509 IF POINT(N,M) THEN PLOT
510 N-(X-X),M-(Y-Y)
511 NEXT N
512 RETURN
513 IF INKEY$="" THEN DRAW X-XA,Y-
514 YA
515 IF INKEY$="B" THEN DRAW X-XA,0
516 IF INKEY$="Y" THEN DRAW X-XA,Y-
517 IF INKEY$="X" THEN DRAW X-XA,0
518 IF INKEY$="A" THEN DRAW X-XA,Y-
519 RETURN
520 INPUT AT 0,0;"EINGABE"
521 IF INKEY$="" THEN RANDOMIZE USR
522 431
523 RETURN
524 PRINT #1;AT 1,0;"TEXTPOSIT."
525 DO SUB 300
526 IF LEN I$(31-FN S()) THEN
527 I$=I$ TO 31-FN S()
528 PRINT AT FN Z(),FN S()
529 PRINT I$
530 IF Y-YE*(Y)=12
531 GO TO 332
532 PRINT #0;AT 1,0;"CURSOR"
533 PAUSE 0
534 LET S$=INKEY$
535 RETURN
536
537 PLOT XA,YA THEN DRAW X-XA,Y-
538 YA
539 RETURN
540 IF INKEY$="B" THEN DRAW X-XA,0
541 IF INKEY$="Y" THEN DRAW X-XA,Y-
542 IF INKEY$="X" THEN DRAW X-XA,0
543 IF INKEY$="A" THEN DRAW X-XA,Y-
544 RETURN
545 INPUT AT 0,0;"EINGABE"
546 IF INKEY$="" THEN RANDOMIZE USR
547 431
548 RETURN
549 PRINT #1;AT 1,0;"TEXTPOSIT."
550 DO SUB 300
551 IF LEN I$(31-FN S()) THEN
552 I$=I$ TO 31-FN S()
553 PRINT AT FN Z(),FN S()
554 PRINT I$
555 IF Y-YE*(Y)=12
556 GO TO 332
557 PRINT #0;AT 1,0;"CURSOR"
558 PAUSE 0
559 LET S$=INKEY$
560 RETURN
561
562 OVER 1
563 DO SUB 322
564 DO SUB 322
565 IF INKEY$="" THEN OVER 0
566 RETURN
567 IF INKEY$="P" OR S$="U" THEN LE
568 LET X=XE+Y
569 LET Y=XE-Y
570 IF ((X(XE-XA))>250) OR ((Y
571 (YA-YE))>10) THEN PRINT #0;
572 "1. KOORD.FEHLER =";
573 PAUSE 10
574 GO TO 228
575 FOR MYA TO YE STEP -1
576 FOR N=XA TO XE
577 IF POINT(N,M) THEN PLOT
578 N-(X-X),M-(Y-Y)
579 NEXT N
580 RETURN
581 IF INKEY$="" THEN DRAW X-XA,Y-
582 YA
583 IF INKEY$="B" THEN DRAW X-XA,0
584 IF INKEY$="Y" THEN DRAW X-XA,Y-
585 IF INKEY$="X" THEN DRAW X-XA,0
586 IF INKEY$="A" THEN DRAW X-XA,Y-
587 RETURN
588 INPUT AT 0,0;"EINGABE"
589 IF INKEY$="" THEN RANDOMIZE USR
590 431
591 RETURN
592 PRINT #1;AT 1,0;"TEXTPOSIT."
593 DO SUB 300
594 IF LEN I$(31-FN S()) THEN
595 I$=I$ TO 31-FN S()
596 PRINT AT FN Z(),FN S()
597 PRINT I$
598 IF Y-YE*(Y)=12
599 GO TO 332
600 PRINT #0;AT 1,0;"CURSOR"
601 PAUSE 0
602 LET S$=INKEY$
603 RETURN
604
605 PLOT XA,YA THEN DRAW X-XA,Y-
606 YA
607 RETURN
608 IF INKEY$="B" THEN DRAW X-XA,0
609 IF INKEY$="Y" THEN DRAW X-XA,Y-
610 IF INKEY$="X" THEN DRAW X-XA,0
611 IF INKEY$="A" THEN DRAW X-XA,Y-
612 RETURN
613 INPUT AT 0,0;"EINGABE"
614 IF INKEY$="" THEN RANDOMIZE USR
615 431
616 RETURN
617 PRINT #1;AT 1,0;"TEXTPOSIT."
618 DO SUB 300
619 IF LEN I$(31-FN S()) THEN
620 I$=I$ TO 31-FN S()
621 PRINT AT FN Z(),FN S()
622 PRINT I$
623 IF Y-YE*(Y)=12
624 GO TO 332
625 PRINT #0;AT 1,0;"CURSOR"
626 PAUSE 0
627 LET S$=INKEY$
628 RETURN
629
630 OVER 1
631 DO SUB 322
632 DO SUB 322
633 IF INKEY$="" THEN OVER 0
634 RETURN
635 IF INKEY$="P" OR S$="U" THEN LE
636 LET X=XE+Y
637 LET Y=XE-Y
638 IF ((X(XE-XA))>250) OR ((Y
639 (YA-YE))>10) THEN PRINT #0;
640 "1. KOORD.FEHLER =";
641 PAUSE 10
642 GO TO 228
643 FOR MYA TO YE STEP -1
644 FOR N=XA TO XE
645 IF POINT(N,M) THEN PLOT
646 N-(X-X),M-(Y-Y)
647 NEXT N
648 RETURN
649 IF INKEY$="" THEN DRAW X-XA,Y-
650 YA
651 IF INKEY$="B" THEN DRAW X-XA,0
652 IF INKEY$="Y" THEN DRAW X-XA,Y-
653 IF INKEY$="X" THEN DRAW X-XA,0
654 IF INKEY$="A" THEN DRAW X-XA,Y-
655 RETURN
656 INPUT AT 0,0;"EINGABE"
657 IF INKEY$="" THEN RANDOMIZE USR
658 431
659 RETURN
660 PRINT #1;AT 1,0;"TEXTPOSIT."
661 DO SUB 300
662 IF LEN I$(31-FN S()) THEN
663 I$=I$ TO 31-FN S()
664 PRINT AT FN Z(),FN S()
665 PRINT I$
666 IF Y-YE*(Y)=12
667 GO TO 332
668 PRINT #0;AT 1,0;"CURSOR"
669 PAUSE 0
670 LET S$=INKEY$
671 RETURN
672
673 PLOT XA,YA THEN DRAW X-XA,Y-
674 YA
675 RETURN
676 IF INKEY$="B" THEN DRAW X-XA,0
677 IF INKEY$="Y" THEN DRAW X-XA,Y-
678 IF INKEY$="X" THEN DRAW X-XA,0
679 IF INKEY$="A" THEN DRAW X-XA,Y-
680 RETURN
681 INPUT AT 0,0;"EINGABE"
682 IF INKEY$="" THEN RANDOMIZE USR
683 431
684 RETURN
685 PRINT #1;AT 1,0;"TEXTPOSIT."
686 DO SUB 300
687 IF LEN I$(31-FN S()) THEN
688 I$=I$ TO 31-FN S()
689 PRINT AT FN Z(),FN S()
690 PRINT I$
691 IF Y-YE*(Y)=12
692 GO TO 332
693 PRINT #0;AT 1,0;"CURSOR"
694 PAUSE 0
695 LET S$=INKEY$
696 RETURN
697
698 OVER 1
699 DO SUB 322
700 DO SUB 322
701 IF INKEY$="" THEN OVER 0
702 RETURN
703 IF INKEY$="P" OR S$="U" THEN LE
704 LET X=XE+Y
705 LET Y=XE-Y
706 IF ((X(XE-XA))>250) OR ((Y
707 (YA-YE))>10) THEN PRINT #0;
708 "1. KOORD.FEHLER =";
709 PAUSE 10
710 GO TO 228
711 FOR MYA TO YE STEP -1
712 FOR N=XA TO XE
713 IF POINT(N,M) THEN PLOT
714 N-(X-X),M-(Y-Y)
715 NEXT N
716 RETURN
717 IF INKEY$="" THEN DRAW X-XA,Y-
718 YA
719 IF INKEY$="B" THEN DRAW X-XA,0
720 IF INKEY$="Y" THEN DRAW X-XA,Y-
721 IF INKEY$="X" THEN DRAW X-XA,0
722 IF INKEY$="A" THEN DRAW X-XA,Y-
723 RETURN
724 INPUT AT 0,0;"EINGABE"
725 IF INKEY$="" THEN RANDOMIZE USR
726 431
727 RETURN
728 PRINT #1;AT 1,0;"TEXTPOSIT."
729 DO SUB 300
730 IF LEN I$(31-FN S()) THEN
731 I$=I$ TO 31-FN S()
732 PRINT AT FN Z(),FN S()
733 PRINT I$
734 IF Y-YE*(Y)=12
735 GO TO 332
736 PRINT #0;AT 1,0;"CURSOR"
737 PAUSE 0
738 LET S$=INKEY$
739 RETURN
740
741 PLOT XA,YA THEN DRAW X-XA,Y-
742 YA
743 RETURN
744 IF INKEY$="B" THEN DRAW X-XA,0
745 IF INKEY$="Y" THEN DRAW X-XA,Y-
746 IF INKEY$="X" THEN DRAW X-XA,0
747 IF INKEY$="A" THEN DRAW X-XA,Y-
748 RETURN
749 INPUT AT 0,0;"EINGABE"
750 IF INKEY$="" THEN RANDOMIZE USR
751 431
752 RETURN
753 PRINT #1;AT 1,0;"TEXTPOSIT."
754 DO SUB 300
755 IF LEN I$(31-FN S()) THEN
756 I$=I$ TO 31-FN S()
757 PRINT AT FN Z(),FN S()
758 PRINT I$
759 IF Y-YE*(Y)=12
760 GO TO 332
761 PRINT #0;AT 1,0;"CURSOR"
762 PAUSE 0
763 LET S$=INKEY$
764 RETURN
765
766 OVER 1
767 DO SUB 322
768 DO SUB 322
769 IF INKEY$="" THEN OVER 0
770 RETURN
771 IF INKEY$="P" OR S$="U" THEN LE
772 LET X=XE+Y
773 LET Y=XE-Y
774 IF ((X(XE-XA))>250) OR ((Y
775 (YA-YE))>10) THEN PRINT #0;
776 "1. KOORD.FEHLER =";
777 PAUSE 10
778 GO TO 228
779 FOR MYA TO YE STEP -1
780 FOR N=XA TO XE
781 IF POINT(N,M) THEN PLOT
782 N-(X-X),M-(Y-Y)
783 NEXT N
784 RETURN
785 IF INKEY$="" THEN DRAW X-XA,Y-
786 YA
787 IF INKEY$="B" THEN DRAW X-XA,0
788 IF INKEY$="Y" THEN DRAW X-XA,Y-
789 IF INKEY$="X" THEN DRAW X-XA,0
790 IF INKEY$="A" THEN DRAW X-XA,Y-
791 RETURN
792 INPUT AT 0,0;"EINGABE"
793 IF INKEY$="" THEN RANDOMIZE USR
794 431
795 RETURN
796 PRINT #1;AT 1,0;"TEXTPOSIT."
797 DO SUB 300
798 IF LEN I$(31-FN S()) THEN
799 I$=I$ TO 31-FN S()
800 PRINT AT FN Z(),FN S()
801 PRINT I$
802 IF Y-YE*(Y)=12
803 GO TO 332
804 PRINT #0;AT 1,0;"CURSOR"
805 PAUSE 0
806 LET S$=INKEY$
807 RETURN
808
809 PLOT XA,YA THEN DRAW X-XA,Y-
810 YA
811 RETURN
812 IF INKEY$="B" THEN DRAW X-XA,0
813 IF INKEY$="Y" THEN DRAW X-XA,Y-
814 IF INKEY$="X" THEN DRAW X-XA,0
815 IF INKEY$="A" THEN DRAW X-XA,Y-
816 RETURN
817 INPUT AT 0,0;"EINGABE"
818 IF INKEY$="" THEN RANDOMIZE USR
819 431
820 RETURN
821 PRINT #1;AT 1,0;"TEXTPOSIT."
822 DO SUB 300
823 IF LEN I$(31-FN S()) THEN
824 I$=I$ TO 31-FN S()
825 PRINT AT FN Z(),FN S()
826 PRINT I$
827 IF Y-YE*(Y)=12
828 GO TO 332
829 PRINT #0;AT 1,0;"CURSOR"
830 PAUSE 0
831 LET S$=INKEY$
832 RETURN
833
834 OVER 1
835 DO SUB 322
836 DO SUB 322
837 IF INKEY$="" THEN OVER 0
838 RETURN
839 IF INKEY$="P" OR S$="U" THEN LE
840 LET X=XE+Y
841 LET Y=XE-Y
842 IF ((X(XE-XA))>250) OR ((Y
843 (YA-YE))>10) THEN PRINT #0;
844 "1. KOORD.FEHLER =";
845 PAUSE 10
846 GO TO 228
847 FOR MYA TO YE STEP -1
848 FOR N=XA TO XE
849 IF POINT(N,M) THEN PLOT
850 N-(X-X),M-(Y-Y)
851 NEXT N
852 RETURN
853 IF INKEY$="" THEN DRAW X-XA,Y-
854 YA
855 IF INKEY$="B" THEN DRAW X-XA,0
856 IF INKEY$="Y" THEN DRAW X-XA,Y-
857 IF INKEY$="X" THEN DRAW X-XA,0
858 IF INKEY$="A" THEN DRAW X-XA,Y-
859 RETURN
860 INPUT AT 0,0;"EINGABE"
861 IF INKEY$="" THEN RANDOMIZE USR
862 431
863 RETURN
864 PRINT #1;AT 1,0;"TEXTPOSIT."
865 DO SUB 300
866 IF LEN I$(31-FN S()) THEN
867 I$=I$ TO 31-FN S()
868 PRINT AT FN Z(),FN S()
869 PRINT I$
870 IF Y-YE*(Y)=12
871 GO TO 332
872 PRINT #0;AT 1,0;"CURSOR"
873 PAUSE 0
874 LET S$=INKEY$
875 RETURN
876
877 PLOT XA,YA THEN DRAW X-XA,Y-
878 YA
879 RETURN
880 IF INKEY$="B" THEN DRAW X-XA,0
881 IF INKEY$="Y" THEN DRAW X-XA,Y-
882 IF INKEY$="X" THEN DRAW X-XA,0
883 IF INKEY$="A" THEN DRAW X-XA,Y-
884 RETURN
885 INPUT AT 0,0;"EINGABE"
886 IF INKEY$="" THEN RANDOMIZE USR
887 431
888 RETURN
889 PRINT #1;AT 1,0;"TEXTPOSIT."
890 DO SUB 300
891 IF LEN I$(31-FN S()) THEN
892 I$=I$ TO 31-FN S()
893 PRINT AT FN Z(),FN S()
894 PRINT I$
895 IF Y-YE*(Y)=12
896 GO TO 332
897 PRINT #0;AT 1,0;"CURSOR"
898 PAUSE 0
899 LET S$=INKEY$
900 RETURN
901
902 OVER 1
903 DO SUB 322
904 DO SUB 322
905 IF INKEY$="" THEN OVER 0
906 RETURN
907 IF INKEY$="P" OR S$="U" THEN LE
908 LET X=XE+Y
909 LET Y=XE-Y
910 IF ((X(XE-XA))>250) OR ((Y
911 (YA-YE))>10) THEN PRINT #0;
912 "1. KOORD.FEHLER =";
913 PAUSE 10
914 GO TO 228
915 FOR MYA TO YE STEP -1
916 FOR N=XA TO XE
917 IF POINT(N,M) THEN PLOT
918 N-(X-X),M-(Y-Y)
919 NEXT N
920 RETURN
921 IF INKEY$="" THEN DRAW X-XA,Y-
922 YA
923 IF INKEY$="B" THEN DRAW X-XA,0
924 IF INKEY$="Y" THEN DRAW X-XA,Y-
925 IF INKEY$="X" THEN DRAW X-XA,0
926 IF INKEY$="A" THEN DRAW X-XA,Y-
927 RETURN
928 INPUT AT 0,0;"EINGABE"
929 IF INKEY$="" THEN RANDOMIZE USR
930 431
931 RETURN
932 PRINT #1;AT 1,0;"TEXTPOSIT."
933 DO SUB 300
934 IF LEN I$(31-FN S()) THEN
935 I$=I$ TO 31-FN S()
936 PRINT AT FN Z(),FN S()
937 PRINT I$
938 IF Y-YE*(Y)=12
939 GO TO 332
940 PRINT #0;AT 1,0;"CURSOR"
941 PAUSE 0
942 LET S$=INKEY$
943 RETURN
944
945 PLOT XA,YA THEN DRAW X-XA,Y-
946 YA
947 RETURN
948 IF INKEY$="B" THEN DRAW X-XA,0
949 IF INKEY$="Y" THEN DRAW X-XA,Y-
950 IF INKEY$="X" THEN DRAW X-XA,0
951 IF INKEY$="A" THEN DRAW X-XA,Y-
952 RETURN
953 INPUT AT 0,0;"EINGABE"
954 IF INKEY$="" THEN RANDOMIZE USR
955 431
956 RETURN
957 PRINT #1;AT 1,0;"TEXTPOSIT."
958 DO SUB 300
959 IF LEN I$(31-FN S()) THEN
960 I$=I$ TO 31-FN S()
961 PRINT AT FN Z(),FN S()
962 PRINT I$
963 IF Y-YE*(Y)=12
964 GO TO 332
965 PRINT #0;AT 1,0;"CURSOR"
966 PAUSE 0
967 LET S$=INKEY$
968 RETURN
969
970 OVER 1
971 DO SUB 322
972 DO SUB 322
973 IF INKEY$="" THEN OVER 0
974 RETURN
975 IF INKEY$="P" OR S$="U" THEN LE
976 LET X=XE+Y
977 LET Y=XE-Y
978 IF ((X(XE-XA))>250) OR ((Y
979 (YA-YE))>10) THEN PRINT #0;
980 "1. KOORD.FEHLER =";
981 PAUSE 10
982 GO TO 228
983 FOR MYA TO YE STEP -1
984 FOR N=XA TO XE
985 IF POINT(N,M) THEN PLOT
986 N-(X-X),M-(Y-Y)
987 NEXT N
988 RETURN
989 IF INKEY$="" THEN DRAW X-XA,Y-
990 YA
991 IF INKEY$="B" THEN DRAW X-XA,0
992 IF INKEY$="Y" THEN DRAW X-XA,Y-
993 IF INKEY$="X" THEN DRAW X-XA,0
994 IF INKEY$="A" THEN DRAW X-XA,Y-
995 RETURN
996 INPUT AT 0,0;"EINGABE"
997 IF INKEY$="" THEN RANDOMIZE USR
998 431
999 RETURN
1000 PRINT #1;AT 1,0;"TEXTPOSIT."
1001 DO SUB 300
1002 IF LEN I$(31-FN S()) THEN
1003 I$=I$ TO 31-FN S()
1004 PRINT AT FN Z(),FN S()
1005 PRINT I$
1006 IF Y-YE*(Y)=12
1007 GO TO 332
1008 PRINT #0;AT 1,0;"CURSOR"
1009 PAUSE 0
1010 LET S$=INKEY$
1011 RETURN
1012
1013 PLOT XA,YA THEN DRAW X-XA,Y-
1014 YA
1015 RETURN
1016 IF INKEY$="B" THEN DRAW X-XA,0
1017 IF INKEY$="Y" THEN DRAW X-XA,Y-
1018 IF INKEY$="X" THEN DRAW X-XA,0
1019 IF INKEY$="A" THEN DRAW X-XA,Y-
1020 RETURN
1021 INPUT AT 0,0;"EINGABE"
1022 IF INKEY$="" THEN RANDOMIZE USR
1023 431
1024 RETURN
1025 PRINT #1;AT 1,0;"TEXTPOSIT."
1026 DO SUB 300
1027 IF LEN I$(31-FN S()) THEN
1028 I$=I$ TO 31-FN S()
1029 PRINT AT FN Z(),FN S()
1030 PRINT I$
1031 IF Y-YE*(Y)=12
1032 GO TO 332
1033 PRINT #0;AT 1,0;"CURSOR"
1034 PAUSE 0
1035 LET S$=INKEY$
1036 RETURN
1037
1038 OVER 1
1039 DO SUB 322
1040 DO SUB 322
1041 IF INKEY$="" THEN OVER 0
1042 RETURN
1043 IF INKEY$="P" OR S$="U" THEN LE
1044 LET X=XE+Y
1045 LET Y=XE-Y
1046 IF ((X(XE-XA))>250) OR ((Y
1047 (YA-YE))>10) THEN PRINT #0;
1048 "1. KOORD.FEHLER =";
1049 PAUSE 10
1050 GO TO 228
1051 FOR MYA TO YE STEP -1
1052 FOR N=XA TO XE
1053 IF POINT(N,M) THEN PLOT
1054 N-(X-X),M-(Y-Y)
1055 NEXT N
1056 RETURN
1057 IF INKEY$="" THEN DRAW X-XA,Y-
1058 YA
1059 IF INKEY$="B" THEN DRAW X-XA,0
1060 IF INKEY$="Y" THEN DRAW X-XA,Y-
1061 IF INKEY$="X" THEN DRAW X-XA,0
1062 IF INKEY$="A" THEN DRAW X-XA,Y-
1063 RETURN
1064 INPUT AT 0,0;"EINGABE"
1065 IF INKEY$="" THEN RANDOMIZE USR
1066 431
1067 RETURN
1068 PRINT #1;AT 1,0;"TEXTPOSIT."
1069 DO SUB 300
1070 IF LEN I$(31-FN S()) THEN
1071 I$=I$ TO 31-FN S()
1072 PRINT AT FN Z(),FN S()
1073 PRINT I$
1074 IF Y-YE*(Y)=12
1075 GO TO 332
1076 PRINT #0;AT 1,0;"CURSOR"
1077 PAUSE 0
1078 LET S$=INKEY$
1079 RETURN
1080
1081 PLOT XA,YA THEN DRAW X-XA,Y-
1082 YA
1083 RETURN
1084 IF INKEY$="B" THEN DRAW X-XA,0
1085 IF INKEY$="Y" THEN DRAW X-XA,Y-
1086 IF INKEY$="X" THEN DRAW X-XA,0
1087 IF INKEY$="A" THEN DRAW X-XA,Y-
1088 RETURN
1089 INPUT AT 0,0;"EINGABE"
1090 IF INKEY$="" THEN RANDOMIZE USR
1091 431
1092 RETURN
1093 PRINT #1;AT 1,0;"TEXTPOSIT."
1094 DO SUB 300
1095 IF LEN I$(31-FN S()) THEN
1096 I$=I$ TO 31-FN S()
1097 PRINT AT FN Z(),FN S()
1098 PRINT I$
1099 IF Y-YE*(Y)=12
1100 GO TO 332
1101 PRINT #0;AT 1,0;"CURSOR"
1102 PAUSE 0
1103 LET S$=INKEY$
1104 RETURN
1105
1106 OVER 1
1107 DO SUB 322
1108 DO SUB 322
1109 IF INKEY$="" THEN OVER 0
1110 RETURN
1111 IF INKEY$="P" OR S$="U" THEN LE
1112 LET X=XE+Y
1113 LET Y=XE-Y
1114 IF ((X(XE-XA))>250) OR ((Y
1115 (YA-YE))>10) THEN PRINT #0;
1116 "1. KOORD.FEHLER =";
1117 PAUSE 10
1118 GO TO 228
1119 FOR MYA TO YE STEP -1
1120 FOR N=XA TO XE
1121 IF POINT(N,M) THEN PLOT
1122 N-(X-X),M-(Y-Y)
1123 NEXT N
1124 RETURN
1125 IF INKEY$="" THEN DRAW X-XA,Y-
1126 YA
1127 IF INKEY$="B" THEN DRAW X-XA,0
1128 IF INKEY$="Y" THEN DRAW X-XA,Y-
1129 IF INKEY$="X" THEN DRAW X-XA,0
1130 IF INKEY$="A" THEN DRAW X-XA,Y-
1131 RETURN
1132 INPUT AT 0,0;"EINGABE"
1133 IF INKEY$="" THEN RANDOMIZE USR
1134 431
1135 RETURN
1136 PRINT #1;AT 1,0;"TEXTPOSIT."
1137 DO SUB 300
1138 IF LEN I$(31-FN S()) THEN
1139 I$=I$ TO 31-FN S()
1140 PRINT AT FN Z(),FN S()
1141 PRINT I$
1142 IF Y-YE*(Y)=12
1143 GO TO 332
1144 PRINT #0;AT 1,0;"CURSOR"
1145 PAUSE 0
1146 LET S$=INKEY$
1147 RETURN
1148
1149 PLOT XA,YA THEN DRAW X-XA,Y-
1150 YA
1151 RETURN
1152 IF INKEY$="B" THEN DRAW X-XA,0
1153 IF INKEY$="Y" THEN DRAW X-XA,Y-
1154 IF INKEY$="X" THEN DRAW X-XA,0
1155 IF INKEY$="A" THEN DRAW X-XA,Y-
1156 RETURN
1157 INPUT AT 0,0;"EINGABE"
1158 IF INKEY$="" THEN RANDOMIZE USR
1159 431
1160 RETURN
1161 PRINT #1;AT 1,0;"TEXTPOSIT."
1162 DO SUB 300
1163 IF LEN I$(31-FN S()) THEN
1164 I$=I$ TO 31-FN S()
1165 PRINT AT FN Z(),FN S()
1166 PRINT I$
1167 IF Y-YE*(Y)=12
1168 GO TO 332
1169 PRINT #0;AT 1,0;"CURSOR"
1170 PAUSE 0
1171 LET S$=INKEY$
1172 RETURN
1173
1174 OVER 1
1175 DO SUB 322
1176 DO SUB 322
1177 IF INKEY$="" THEN OVER 0
1178 RETURN
1179 IF INKEY$="P" OR S$="U" THEN LE
1180 LET X=XE+Y
1181 LET Y=XE-Y
1182 IF ((X(XE-XA))>250) OR ((Y
1183 (YA-YE))>10) THEN PRINT #0;
1184 "1. KOORD.FEHLER =";
1185 PAUSE 10
1186 GO TO 228
1187 FOR MYA TO YE STEP -1
1188 FOR N=XA TO XE
1189 IF POINT(N,M) THEN PLOT
1190 N-(X-X),M-(Y-Y)
1191 NEXT N
1192 RETURN
1193 IF INKEY$="" THEN DRAW X-XA,Y-
1194 YA
1195 IF INKEY$="B" THEN DRAW X-XA,0
1196 IF INKEY$="Y" THEN DRAW X-XA,Y-
1197 IF INKEY$="X" THEN DRAW X-XA,0
1198 IF INKEY$="A" THEN DRAW X-XA,Y-
1199 RETURN
1200 INPUT AT 0,0;"EINGABE"
1201 IF INKEY$="" THEN RANDOMIZE USR
1202 431
1203 RETURN
1204 PRINT #1;AT 1,0;"TEXTPOSIT."
1205 DO SUB 300
1206 IF LEN I$(31-FN S()) THEN
1207 I$=I$ TO 31-FN S()
1208 PRINT AT FN Z(),FN S()
1209 PRINT I$
1210 IF Y-YE*(Y)=12
1211 GO TO 332
1212 PRINT #0;AT 1,0;"CURSOR"
1213 PAUSE 0
1214 LET S$=INKEY$
1215 RETURN
1216
1217 PLOT XA,YA THEN DRAW X-XA,Y-
1218 YA
1219 RETURN
1220 IF INKEY$="B" THEN DRAW X-XA,0
1221 IF INKEY$="Y" THEN DRAW X-XA,Y-
1222 IF INKEY$="X" THEN DRAW X-XA,0
1223 IF INKEY$="A" THEN DRAW X-XA,Y-
1224 RETURN
1225 INPUT AT 0,0;"EINGABE"
1226 IF INKEY$="" THEN RANDOMIZE USR
1227 431
1228 RETURN
1229 PRINT #1;AT 1,0;"TEXTPOSIT."
1230 DO SUB 300
1231 IF LEN I$(31-FN S()) THEN
1232 I$=I$ TO 31-FN S()
1233 PRINT AT FN Z(),FN S()
1234 PRINT I$
1235 IF Y-YE*(Y)=12
1236 GO TO 332
1237 PRINT #0;AT 1,0;"CURSOR"
1238 PAUSE 0
1239 LET S$=INKEY$
1240 RETURN
1241
1242 OVER 1
1243 DO SUB 322
1244 DO SUB 322
1245 IF INKEY$="" THEN OVER 0
1246 RETURN
1247 IF INKEY$="P" OR S$="U" THEN LE
1248 LET X=XE+Y
1249 LET Y=XE-Y
1250 IF ((X(XE-XA))>250) OR ((Y
1251
```

Ansteuern Scroll (up + left)  
 Ansteuern Copy-Funktion  
 • Bereich abgrenzen (Z. 240)  
 • Verschiebekoordinaten festlegen  
 • Pixelplot (Z. 236)

Zeile 84->332  
 TEXT Position festlegen  
 Texteingabe  
 Textprint

Zeile 85->300  
 UNPLOT in "Cursor" integriert

Zeile 168 DRAW-Endabfrage (auch bei falscher  
 Optionwahl)

#### U p r o ' s

Zeile 183 - 237 SCREEN-Funktion  
 Zeile 240 - 252 DRAW + BOX-Funktion  
 Zeile 267 - 268 CIRCLE-Funktion  
 Zeile 300 - 328 CURSOR  
 darin enthalten Plot-Abfrage (Zeile 324) +  
 Unplot-Abfrage (Zeile 325-326)  
 Zeile 332 - 340 TEXT-Funktion  
 Zeile 3342 - 344 Abfrage Cursor-Geschwindigkeit

#### Variablen

F,S Konstanten für Cursorgeschwindigkeit  
 (S=slow, F=fast)  
 G gewählte Cursorgeschwindigkeit  
 M Circle-Durchmesser  
 X,Y Plotkoordinate aktuell  
 XA,XE,YA,YE temporäre Zwischenkoordinaten (Verschieben, Box)  
 SS Userinput Menu

#### Funktionen

FN Z() Umrechnung Plotpunkt  
 FN S() zu Print Position  
 (Zeile/Spalte)

#### MC-Aufrufe

Zeile 23 + 332 => USR 4317 : Umschalten auf Klein/Großbuchstaben  
 (ROM-Routine)  
 Zeile 203 => USR 32700: Scroll left

Zeile 204 => USR 32600: Scroll up

#### MEMO:

\*\*\* Alle Schaubilder, Schema etc. in diesem Buch wurden mit \*\*\*  
 \*\*\* DRAWUTIL erstellt. \*\*\*

#### SCREEN-MANIPULATIONEN

Viele Daten auf dem Bildschirm mit HEIMBUDGET

#### Grafik voll ausgeschöpft

In folgenden Teil des Grafik-Kapitels zeigt der SINCLAIR, was in ihm steckt. Es stellt mit Programm HEIMBUDGET den Übergang zur Datenverarbeitung dar. Anders ausgedrückt - Daten ohne Grafik ist wie ein Maler ohne Staffelei. Erst in "Optik" umgesetzt werden Zahlen sprechend und verständlich. Bekanntlich werden 80 % aller Informationen übers Auge wahrgenommen. Schöpfen wir daher die grafischen Möglichkeiten unserer Maschine voll aus, zeigen wir, was und wie man vieles auf den Screen bringt. Die Manipulationen des Bildschirms laufen dabei alle in Basic, wozu der bekannt schnelle DRAW des SINCLAIR beiträgt. Auch werden unbekannte ROM-Routinen (per RANDOMIZE USR) benutzt, die so schöne Dinge wie "Windows" (= Bildschirmfenster) und partielle Scrolls ermöglichen. - Gleich vorweg, HEIMBUDGET ist nicht einfach, daher ist viel Aufwand für lückenlose Dokumentation betrieben worden. Dafür bietet das Programm auch ein Menge. Um nur ein paar Stichworte zum Thema Heimdatenverwaltung und Grafik "at its best" vorab zu bringen:

- + automatische Skalierung von Positionen (horizontal + vertikal)
- + Skulpturgrafik mit Summenbalken vorab
- + Hell/Dunkelsteuerung bei Kurven (zum besseren Lesen der Spaltenpositionen)
- + Mischung von Kurven und Zahlenprint in zwei "Fenstern"
- + gleichzeitige Darstellung zweier Kurven nach Userwahl getrennt oder "zusammengeschoben"
- + Statusbericht mit Wertebelegung (zgn. Monatsposten)
- + leichtes Update mit der Möglichkeit, Werte "zellenweise" vor- und rückzutragen
- + einfache Werteingabe durch vorgegebene Bezeichner

Gerade die Anwendung einer speziellen Window-Technik bringt neue Effekte. Sie wird erreicht durch die Möglichkeit, mit bestimmten ROM-Routinen einen Teil-Screen-Scroll herbei zu führen. Auch das Löschen eines Screen-Bereichs ist so zu erreichen. Diese Screen-Manipulationen werden genau beschrieben; vor allem - alles geht recht rasch in Basic! - Wenden wir uns so neugierig

geworden dem Programm zu. Die folgenden Zeilen führen schrittweise an HEIMBUDGET heran; zum Schluß des Kapitels "Grafik" sind alle Screenstricks (in HEIMBUDGET) noch genauer erklärt.

Was kann das Programm ?

HEIMBUDGET ermöglicht die Verwaltung, Aktualisierung und Darstellung beliebiger Posten einer Einnahmen-/Ausgabenrechnung. Es ist zunächst für den Hausgebrauch gedacht, kann jedoch auch für begrenzte kommerzielle Anwendungen benutzt werden. Wegen der freien Vorgabe von Bezeichnungen ist die Grenze weit gezogen, sie reicht von simpler Haushaltskontrolle bis zur Bilanzanalyse, von kleingewerblichen Einnahmen/Ausgaben bis zur Deckungsbeitragsrechnung unterschiedlicher Artikelgruppen... Da alle Eingaben entweder in Monats- oder Jahresübersichten aufbereitet werden, ist eine gute visuelle Kontrolle vorhanden. Sie reicht von der Abfrage aktueller Status bis zur kombinierten Grafik beliebiger "Etatposten". Durch trickreiche Verwendung von ROM-Scroll-Routinen ist es möglich, mehrere "Fenster" auf dem Bildschirm zu eröffnen. Auch die Mischung aus Zahlen / Diagrammen wurde verwirklicht. Beim Menüzeig "Jahresübersicht" werden sogar zwei verschiedene Diagramme zusammengeschoben, um die dazugehörigen Werte auf dem unteren Bildschirm mit darzustellen. Bei allen dem wird Bedienungskomfort groß geschrieben:

Neben Selbstverständlichkeiten wie automatischer Skalierung aller Wertepoints, stellingerechter Tabulierung bei Zahlen (incl. Rundung) und maßstabgerechter Grafik wurde dem sgn. Benutzerhandling verstärkt Rechnung getragen. Dazu ein paar Beispiele:

- bei "Anwahl" von Einzelposten (s. Option EINZELNENTWICKLUNG, Menüast JAHRESÜBERSICHT) wird dem Benutzer die Skala aller Bezeichner im INPUT-Bereich präsentiert; mittels Cursor-Tasten führt er die Leiste entlang, bis er sein Wahl getroffen hat.
- Da häufig einzelne Werte übers Jahr gleich bleiben, kann der User - ebenfalls unter CursorControl - eingegebene Monatszahlen vor- oder auch zurücktragen. Miete, Versicherungsbeiträge, Ratenzahlungen werden in die angezeigten Monatsfelder kopiert. Ein Feature, wie man es sonst nur bei anspruchsvollen SPREADSHEETs findet..
- Wird Grafik mit zugehörigen Werten gemischt (s. MONATSÜBERSICHT), kann es vorkommen, daß der untere Bildschirmbereich nicht ausreicht. Zu dem Zweck wurde ein Scroll des unteren "windows" realisiert. Damit der Benutzer synchrone Werte und Darstellung verfolgen kann (beim lower screen scroll), gibt es eine extra Pausensteuerung: Scroll solange keine Taste gedrückt ist, Pause wenn dies geschieht, und wieder Scroll, falls erneut eine Taste betätigt wird.
- Um bei der MONATSÜBERSICHT weitere Informationen präsent zu haben, ist - neben der Prozentaufschlüsselung aller Positionen

- im oberen Grafikteil ein "Balken" eingebaut. Er zeigt die Summe der Einnahmen und die der - so die Regel - geringeren Ausgaben. Und was passiert, wenn letztere einmal die Einnahmen übersteigen? Dann wird automatisch der Ausgabenteil des Balkens von oben geschwärtzt. Man weiß auf einen Blick, welchen Anteil die Ausgaben eines Monats ausmachen und ob sie die Einnahmen übersteigen.

Auf eins wurde jedoch bei aller Grafik verzichtet - auf den Einsatz von Farbe; aus einem einfachen Grund - nur die wenigsten Spectrum-Besitzer genießen den Luxus eines Farbmonitors. Zudem ist die Unsitte zu beobachten, mit viel Farbspielerei von programmtechnisch schwacher Lösung abzulenken... Das Programm HEIMBUDGET wählt einen zweckmäßigeren Weg, Informationen sauber und unterscheidbar darzustellen - durch gezielten Einsatz der Hell/Dunkelsteuerung (mit BRIGHT). Die Monatskolonnen sind abwechselnd normal und "bright" gehalten. So ist leicht zu verfolgen, zu welchem Monat ein Plotwert gehört.

Wie ist das Programm aufgebaut ?

Die übliche Menüsteuerung zeigt als Hauptliste des Programms:

```
INITIALISIEREN
UPDATE MONAT
AKT.UPDATE-STATUS
MONATSÜBERSICHT
fern - SAVE, LOAD und ENDE
```

INITIALISIEREN dient der Besetzung der Params, Datenarrays werden dimensioniert und mit Erstwerten besetzt, sowie das Programm mit den nötigen Initialisierungen versehen. (Wohin zu diesen und den anderen Menüzeilen enthält die genaue Programmbeschreibung)

UPDATE ist der Eingabe monatlicher Postenzahlen vorbehalten. Nach Anwahl des gewünschten Monats werden Neuwerte eingegeben oder ggf. vorhandene bestätigt. Es ist also möglich, auf einen updateten Monat zurück zu schalten und Korrekturen gezielt vorzunehmen. Der Arbeitserleichterung dient die Option "Vor/Rücktag", die dem schnellen Wertauffüllen anderer Monate dient. Es ist zu beachten, daß sie das Update voraussetzt, d. h. vorgetragene Monatspositionen werden erst "endgültig", wenn ein Updatetdurchgang durchgeführt wird. (Selbstverständlich reicht es, das im Schnelldurchgang mit ENTER - Bestätigung aller Zahlen zu erledigen.)

AKTUELLER STATUS schafft ein Layout aller Jahrespositionen, die bereits mit Werten besetzt sind. Es dient der schnellen Übersicht, um festzustellen, wie weit man bereits updatet hat. MONATSÜBERSICHT ist wie die JAHRESÜBERSICHT von der Grafik bestimmt. Im oberen Screenbereich steht neben dem erwähnten "Einn./Ausgabe-Balken" der Balken für jeden Einzelposten. Natürlich erfolgt die Ausrichtung der Säulen automatisch, wenige Posten bringen breite, viele schmale Säulen. Synchron mit dem Wertepoint erscheinen die Monatszahlen im unteren Bildschirmbereich. Man kann der Scroll abstoppen und wieder in Gang setzen, wie bereits geschildert. Wichtig ist, daß unbegrenzt Einzelpositionen dargestellt werden können.

Bekanntlich gibt es keine Beschränkung dabei - außer der des Speichers...Die Zahlen werden tabuliert und gerundet, jeder Wert hat zu Vergleichszwecken die %-Angabe (jeweils bezogen auf Summe Einnahmen bzw. Ausgaben).

JAHRESUEBERSICHT heißt der wohl aussagekräftigste Teil des Programms. Nach vorhandenen Monatssummen kann der User wählen zwischen "Summen-" und "Einzelentwicklung". Bei erster werden drei Diagramme auf zwei Screens gleichzeitig gebracht: 1. Entwicklung aller Monatssummen 'Einnahmen' - upper screen 2. Saldo von Einnahmen/Ausgaben übers Jahr - Grundlinie upper screen

3. Entwicklung Ausgaben der Monate (gem. Hell-Dunkel-Leiste) - lower screen

Möchte man die dazu gehörenden Summen eines Monats wissen, wird Grafik 1 und 3 zu einer einzigen im oberen Bildschirm zusammengeschoben, um "unten" für Zahlen Platz zu machen. Einnahmen haben das Zeichen +, Ausgaben sind durch - ausgewiesen.

Noch interessanter ist die Option "Einzelentwicklung". Zunächst wird per Cursor ein Posten aus der Bezeichnerliste gewählt, er kommt in den oberen Ausschnitt; dann ist der zweite Etappen dran, der "unten" hinkommt. Skalierung und Beschriftung richtet sich nach dem jeweiligen Maximum, sodaß immer eine maßstabgetreue Plot-Darstellung vorliegt. Der Nutzen eines solchen Verfahrens ist naheliegend - man kann kreuz und quer Postenvergleiche anstellen. Bei vollem Screen wird das ganze Bild einfach hochgerollt und neu gemalt..

Die Menüzeile LOAD,SAVE und ENDE brauchen nicht weiter behandelt zu werden. Eine Anregung dazu: statt das gesamte Programm zu speichern, kann man statt dessen - bei sonst gleichen Bezeichnern, Postenzahl und Jahr - den Datensatz N(13,N+3) direkt abspeichern. Mit schnellem Microdrive hat man dann eine ganze Menge Daten im direkten Zugriff.

Im Folgenden sind für den engagierten Spectrumprogrammierer, der es ganz genau wissen will, noch offene Details nachgetragen.

#### Programmbeschreibung

#### Module

(Nummer = Programmzeile)

130 - 145 INITIALISIEREN  
132-137 Params:  
Jahr, Anzahl Posten (Einnahmen, Ausgaben)  
138-145 Dimensionieren + Erstbesetzung:  
Anlegen Daten- + Textarray  
Festlegen Bezeichnung Einzelposten  
Aufbau Monatsnamenstring

50 - 69 UPDATE  
52- 65 Eingabe Monatszahlen:

Auswahl Monat  
Andruck vorh. Positionen  
Besetzen gem. User-Input  
Bildung Monatssumme bzw. -saldo  
66 - 69 Vor/Rücktag von Monatszahlen:  
Cursorsteuerung  
Monatsanzeige und Wertfortschreibung

20 - 44 AKI.UPDATE-STATUS  
20 - 24 Layout:  
Scroll, Kopf, Einnahmen/Ausgabentrennung  
25 - 27 Andruck Besetzung:  
Hell/Dunkelsteuerung, Besetzungsscheck, Rahmen-plot

70 - 79 MONATSUEBERSICHT  
71 - 79 Aufbau Grafik upper screen:  
Festlegen Einnahmen/Ausgabenpanel (Maximum, Darstellungsfaktor, Balkendiagramm)  
Wertplot  
77 - 79 Anliest Monatszahlen lower screen:  
Scroll Text + Absolutwerte  
Prozentzahlen je Position

80 - 99 JAHRESUEBERSICHT  
85 - 91 Summenentwicklung:  
85-88 Grafik Monatssummen + -saldo  
88-91 Grafik mit Absolutzahlen

92 - 99 Einzelentwicklung  
92-95 Postenwahl per Cursor  
96-97 Bestimmen Darstellungsdetails  
98-99 Skalenbeschriftung + Plot

#### UPRO's

32 - 32 Statusbericht Kopf  
34 - 34 Hell/Dunkelsteuerung der Bestzungsfelder  
36 - 37 Wertplot Jahressummen  
38 - 39 Errrechnen Monatssummen u. Max f. Skalierung  
40 - 44 Aufbau Layout Jahressummen + Bright-Control f. Summenfelder  
66 - 69 Monatsvor- bzw. -rücktrag  
150-154 Monatsabfrage

#### Funktionen

FN IS(X) Tabulieren der gerundeten Zahl X mittels ausgerichteter 'leading spaces'  
FN MS(X) Konversion Monatszahl in Monatsbezeichnung aus MS-String

#### Variablenliste

# Array's

N(13,N+3) Datenarray Monats- u. Jahressummen je Posten wie folgt:  
 N(1 bis 12,...) Monatspostenbeträge Jan-Dez  
 N(13,...) Jahressumme(je Posten)  
 N(...,1 bis N) Einzelposten je Monat  
 N(...,N+1) Summe Einnahmen je Monat  
 N(...,N+2) Summe Ausgaben je Monat  
 N(...,N+3) Monatsaldo Einn./Ausg.

IS(N+3,15) Bezeichner eines jeden Postens(15 Stellen lang)

## Stringfelder

MS Monatsnamen (alle 12 Monate = Länge 12\*9)  
 JS Jahr  
 IS Userinput

## Numerische Felder

A Zahl der "Ausgaben"-Positionen  
 E Zahl der "Einnahmen"-Positionen  
 F Bright On/Off-Schalter; auch für Skalierungs-  
 zwischenwerte verwandt  
 FAK Darstellungsfaktor f. Wertepilot  
 M akt. Monat  
 MAX Maximum der betreffenden Wert-Range  
 N Anzahl aller Positionen (= E + A)  
 O Schalter "oben" bei kombinierter Grafik  
 (s. JAHRESUEBERSICHT)  
 X,Y allgemeine Laufvariablen  
 Z Zählvariable

```

10 REM MEINENBEREICH
11
12 GO TO 100
13
14 DEF FN T$(X) = " " (TO
15   B-LEN STR$(INT X))+STR$ I
16   NT (ABS X+.5)
17 DEF FN M$(X) = M$((X-1)*9+1 T
18   O X*9)
19 REM AKTUELLER STATUS
20
21 LET IS="EINNAHMEN"
22 GO SUB 32
23 PRINT
24 GO SUB 34
25 FOR X=1 TO N
26   IF SCREEN$(21,0) <> " " THE
27     N GO SUB 32

```

```

24 IF X=E+1 THEN LET IS="AUSGA
25   BEN
26   GO SUB 34
27   PRINT TAB 8;T$(X);TAB 18;
28   LET F=1
29   FOR Y=1 TO 12
30     IF N(Y,X) THEN LET IS=" "
31     PRINT BRIGHT F;IS;
32     LET F=NOT F
33   NEXT Y
34   PRINT
35   GO SUB 34
36   PRINT
37   PRINT
38   PRINT
39   PRINT
40   PRINT
41   PRINT
42   PRINT
43   PRINT
44   PRINT
45   PRINT
46   PRINT
47   PRINT
48   PRINT
49   PRINT
50   PRINT
51   PRINT
52   PRINT
53   PRINT
54   PRINT
55   PRINT
56   PRINT
57   PRINT
58   PRINT
59   PRINT
60   PRINT
61   PRINT
62   PRINT
63   PRINT
64   PRINT
65   PRINT
66   PRINT
67   PRINT
68   PRINT
69   PRINT
70   PRINT
71   PRINT
72   PRINT
73   PRINT
74   PRINT
75   PRINT
76   PRINT
77   PRINT
78   PRINT
79   PRINT
80   PRINT
81   PRINT
82   PRINT
83   PRINT
84   PRINT
85   PRINT
86   PRINT
87   PRINT
88   PRINT
89   PRINT
90   PRINT
91   PRINT
92   PRINT
93   PRINT
94   PRINT
95   PRINT
96   PRINT
97   PRINT
98   PRINT
99   PRINT

```

```

32 RANDOMIZE USR 3330
33 PRINT TAB 4;"MONAT";TAB 12;"JAHRESUEBERSICHT"
34 PRINT TAB 4;"POSITIONEN";TAB 12;"H O N"
35 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
36 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
37 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
38 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
39 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
40 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
41 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
42 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
43 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
44 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
45 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
46 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
47 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
48 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
49 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
50 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
51 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
52 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
53 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
54 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
55 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
56 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
57 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
58 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
59 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
60 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
61 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
62 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
63 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
64 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
65 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
66 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
67 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
68 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
69 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
70 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
71 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
72 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
73 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
74 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
75 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
76 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
77 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
78 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
79 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
80 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
81 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
82 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
83 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
84 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
85 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
86 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
87 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
88 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
89 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
90 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
91 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
92 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
93 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
94 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
95 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
96 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
97 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
98 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"
99 PRINT TAB 4;"JAHRESUEBERSICHT";TAB 12;"JAHRESUEBERSICHT"

```

## Datenverwaltung

Im Folgenden soll konkret gezeigt werden, wie man effektiver die Grafik programmiert. Oft steht der User oder fortgeschrittene Programmierer ratlos vor den Erzeugnissen fremder Kunst. Erklären wir also nicht allein das Gebrachte, sondern zeigen genauer das "Wie" und "Warum". Es geht darum, möglichst viel Erläuterung zu geben, quasi als Hilfe zum Selbstprogrammieren. Das Programm HEIMBUDGET verbindet zwei Dinge miteinander, welche den Schwerpunkt dieses Abschnitts ausmachen - Datenverwaltung per Array und wirkungsvolle Grafik zur Veranschaulichung. In einem dritten Punkt werden eine Reihe sonstiger Programmniffe anhand des Listings erläutert. Die Daten finden sich in einem Array, der nach Useranforderung mit DIM N(13,N+3) angelegt wurde. Die Tabelle enthält also volle Floating-Point-Zahlen d.h. jede Zahl braucht 5 Bytes. Natürlich hätte man auch auf Stringzahlen gehen können, was jedoch wenig sinnvoll gewesen wäre. Schließlich sollen auch DM-Nachkommastellen möglich sein. Etwas anderes wurde aber fürs bessere Handling getan: statt Einnahmen, Ausgaben und Summenfelder in getrennten Arrays unterzubringen, sind sie im N(13,N+3)-Array enthalten. Die Datengruppierung ist dafür in Variable E und A festgehalten. Warum? Wer verschiedene Jahre mit HEIMBUDGET fahren will, kann den gesamten Array - sprich Datenbereich - mit einfachem LOAD herein holen. Die Aufteilung wird deutlich, stellt man die beiden Möglichkeiten gegenüber.

```
1.Möglichkeit: Einnahmen DIM N(12,E)
                Ausgaben DIM N(12,A)

                Summe Einnahmen (pro Monat) DIM S(12)
                Summe Ausgaben (pro Monat) DIM T(12)
                Saldo Monatssummen DIM Z(12)

                Jahressummen (als 13.Monat) DIM J(E + A + 3)
```

2.(realisierte) Möglichkeit, wobei N=E+A:

```
DIM N(13,N+3)
```

Zur Erinnerung sei erwähnt, daß gem. SINCLAIR-BASIC eine Variable gleichen Namens (hier N) sowohl für einen Numeric Array, als auch als Einzelvariable wie auch als Laufvariable benutzt werden kann.

## Grafikeinsatz

Zunächst als Leckerbissen vorweg die bisher unbeobachtete Verwendung von ROM-Routinen des Spectrums. Bekanntlich kommt der Aufruf von "vorfabrizierten" Maschinencoderoutinen nicht nur der

Schnelligkeit eines Basic-Programms zugute, er vereinfacht auch die Programmierung. - Der Einsprung in die Scroll Routine ermöglicht folgendes:

```
RANDOMIZE USR 3330
```

scrollt den Gesamtscreen weg (effektvoller als ein simples CLS, wengleich letzteres zur Print-Neupositionierung nützlich ist).

```
RANDOMIZE USR 3583
```

scrollt den unteren Bildschirm - genauer die letzten 12 Zeilen - um eine Zeile hoch; dadurch ist eine neuartige Window-Technik anwendbar, wenn wechselseitig der upper (Zeile 1 - 10) oder lower screen (Zeile 11 - 22) belegt wird. Letzterer kann auch getrennt gelöscht werden mit

```
RANDOMIZE USR 3652
```

Bleibt schließlich noch der Scroll des Gesamtscreens um eine Zeile, was mit

```
RANDOMIZE USR 3280
```

bewerkstelligt wird.

Nun noch ein paar Tricks zur Grafik im allgemeinen, wie in HEIMBUDGET anzutreffen; die Listingzeile ist jeweils zwecks besserer Orientierung mit angegeben.

Statt "Balken" mit viel Plot und Draw voll auszumalen, wurde der Weg einer trickreichen Drawverwendung beschritten. Memo: Plot ist langsamer als Draw. Daher wird nur der Anfangspunkt geplottet, dann erfolgt die Diagrammerzeugung allein mit Draw. Wieso das? Für die Darstellung reicht völlig, relative Werte (= Distanz zum Skalennullpunkt) zu verwenden, DRAW bietet sich geradezu an. Das Verfahren des Umrechnens von absoluten in relative Werte ist einfach; es läuft in drei Schritten ab. Das Folgende hat übrigens Gültigkeit für alle Diagrammgrafik und wurde deshalb genauer wiedergegeben (vgl. Zeile 36):

1.Festlegen Darstellungsbereich (vertikal)

```
Im Programm - Nullpunkt 95, obere Begrenzung 175
-> Bereich 80 Pixel
```

2.Ermittel Wertemaximum

```
Im Programm - Variable MAX, die in einer Schleife
mitbearbeitet wird (vgl. Zeile 38)
```

3.Festlegen Darstellungsfaktor

```
Im Programm - Variable FAK, sie ist gleich MAX/80
```

Nun ist es leicht, die aktuellen Werte zu plotten. Durch einfache Multiplikation bekommt man den Plotpunkt als relativen Distanzwert. Im Programm (s. Zeile 74 oder 36):



LET F = FAK \* AKT. WERT

Da beim Spectrum die Vertikalposition ("Zeile") von 'oben' gerechnet wird, muss lediglich der grafische Begrenzungswert mit berücksichtigt werden. Man kann natürlich auch vom unteren Rand ausgehen (wie in Zeile 77 geschehen). Folglich lautet die endgültige "Formel":

PLOT Y, FAK \* akt. Wert + 95      bzw.  
PLOT Y, 175 - FAK \* akt. Wert

Mit Draw ist die Umrechnung von Anfangsplotpunkt noch einfacher. Da brauchen nur die Pixelabstände "übersetzt" zu werden. Also kommt lediglich die Differenz zum Vorwert zum Tragen. Wenn er X ist, der Nachfolger X1, lautet der Drawbefehl (beim "Balkenabstand" Y1):

DRAW Y1, (X - X1) \* FAK

Natürlich wird man noch ein wenig Feinarbeit ins Runden und Ähnliches investieren. Auch kann man die Darstellungsspanne zwischen Minimum und Maximum der Werte voreinstellen. Dann werden die Kurvenabstände größer, das Diagramm wird 'gezackter'.

Weitere Programmricks

Zeile 34 - Ein EIN-/AUS-Schalter für Bright wird durch die NOT-Logik erreicht. Damit kann eine Variable im Durchlauf abwechselnd an- und abgeschaltet werden. Sie erhält quasi den Status des (PASCAL-)Typs boolean:  
LET F = 1: ... LET F = NOT F

Zeile 38 - Erst- und Folgebesetzung von Werten ließe sich (umständlich) durch Löschen mit nachfolgender Initialisierung erledigen. Was etwa so aussehen würde:  
FOR X = 1 TO N: LET N(13, X) = 0      <Löschen>  
FOR Y = 1 TO 12: LET N(13, Y) = N(13, Y) + N(X, Y) <Neu  
NEXT Y: NEXT X      Besetzen>

Es geht aber auch in einem "Abwasch", wie das Listing zeigt; man muß lediglich die Abfrage auf 'erster Durchgang' einbauen. Das geschieht mit Sinclair-typischem Logikvergleich, der aufgelöst so aussieht:  
IF X = 1 THEN LET N(13, Y) = N(X, Y)  
IF X > 1 THEN LET N(13, Y) = N(13, Y) + N(X, Y)

Zeile 62 - Die Summation der Einnahmeposten liefert ein weiteres Beispiel vielseitiger 'Logik-Schalter'. Insgesamt sind vier Bedingungen abzufragen, welche das Programm ohne ein einziges IF bewältigt:

1. beim ersten Durchgang sollen die 'Einnahmen' ohne Addition besetzt werden (=Löschen + Neubesetzen in einem Zug)

2. bei allen weiteren 'Einnahmedurchgängen' \* ist hingegen fort zu summieren  
3. ist der erste Ausgabeposten erreicht (d.h. X=E), muß analog das erste Ausgabefeld 'erstbelegt' werden  
4. bei den weiteren 'Ausgabedurchgängen' ist gleichermaßen (siehe 2.) fortzusummieren

Eine IF-like Programmierung würde so aussehen:  
IF X = 1 THEN LET N(M, N+1) = N(M, X): GOTO (A)  
IF X <= E THEN LET N(M, N+1) = N(M, N+1) + N(M, X)  
IF X = E + 1 THEN LET N(M, N+2) = N(M, X): GOTO (A)  
IF X > E THEN LET N(M, N+2) = N(M, N+2) + N(M, X)

(A) Anspringzeile zur Programmfortsetzung

Derselbe Effekt wird in Zeile 62 in nur einem Statement erreicht und zwar ohne IF! Man versuche, sich die Logik klar- und zunutze zu machen...  
(Die Cursorsteuerung in Zeile 68 + 95 funktioniert übrigens ganz analog)

Zeile 79 - Die Pausensteuerung ist benutzerfreundlich angelegt. Folgendes wird erreicht:

- solange keine Taste betätigt ist, wird gescrollt
- bei erstem Tastendruck hält das Scrolling an
- wird erneut irgend eine Taste gedrückt, geht das Scrolling weiter

Man hätte das auch so realisieren können:  
IF INKEYS = "" THEN PAUSE 0  
IF INKEYS <> "" THEN PAUSE 20

Statt dessen wurde die "Logik" direkt in die Pausenfunktion eingebaut.

Auf weitere Erläuterungen sei an dieser Stelle verzichtet; man führe sich das Listing zu Gemüte und - vor allem - experimentiere selber ein wenig.

# Kapitel III: ZEICHEN & TEXT

## ZEICHEN AUF DEM BILDSCHIRM

### Manipulation des 'Character Sets'

#### Die Darstellung von Zeichen

Zeichen werden beim SINCLAIR (und anderen Heimcomputern) punktchenweise als Matrix generiert. Jeder Buchstabe, jedes Sonderzeichen und jede Zahl entstehen aus einer 8 x 8 - Matrix. Die Besetzung der Pixel (= schwarzer Punkt höchster Auflösung auf dem Bildschirm) wird von der Information gesteuert, die sich hinter dem Character-Code des Zeichens verbirgt. Der größte Teil darstellbarer Zeichen ist nach der ASCII-Konvention definiert; Buchstabe "I" etwa hat beim Spectrum den Code 84. Dahinter jedoch verbirgt sich mehr fürs Betriebssystem. Der Code besagt nur, wenn dies Zeichen kommt, springe zu einer Tabelle, welche die Zeilen- und Spaltenbesetzung für die Darstellung liefert. Im sign. Character Set finden sich für jedes Pixel die notwendigen Informationen, nach welchen das System blitzschnell die Zeichendarstellung zusammenbaut.

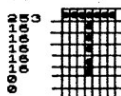
Da sich in einem Byte durch Ausschöpfen aller Bit-Kombinationen die Spaltenbesetzung einer Zeile (der Druckmatrix) gewinnen läßt, braucht das Character Set genau 8 Bytes für die Generierung eines Zeichens; das hat nämlich 8 Zeilen, die spaltenweise mit "Punkten" gefüllt werden. Um die Bitbesetzung, welche die Ein-Bytezahl liefert, zu verstehen, seien die Stellenwertigkeiten wiedergegeben.

128	64	32	16	8	4	2	1
1	2	3	4	5	6	7	8

Die Wertbestimmung erfolgt - wie man sehen kann - in Form von 2er Potenzen. Ist zur "Schwärzung" ein Punkt erforderlich, wird die Wertigkeit hinzuaddiert.

Da meist unten und rechts ein Rand bleibt, ist 253 der höchste Bitwert. Die acht Bytes für das große I werden durch Fortzählen einzelner Wertigkeiten wie angezeigt bestimmt.

Derjenige, welcher schon mal mit selbst definierten Zeichen gearbeitet hat, weiß, daß SINCLAIR (außerst einfach (mit BIN) die Besetzung der Zeichennmatrix ermöglicht. Normalerweise braucht man sich um die Entstehung von Screen-Zeichen keine Gedanken zu machen, das besorgt der Rechner. Anders sieht es aus, wenn Zeichen manipuliert oder ein neuer Zeichensatz gene-



riert werden soll.

Halten wir fest: Bei Zeichen hat jedes Pixel eine Wertigkeit, die sich nach 2er Potenzen (für die Bitsetzung) richtet; pro Zeichen sind 8 Bytes (= 8 Zeilen x 8 Pixels pro Spalte) erforderlich.

#### Die Adressierung

Das Character Set ist dem Rechner bekannt, er weiß, wo er seine "Matrix" findet. Systemvariable CHARS (s. Manual, Kapitel 25) sagt, ab welcher Adresse der Zeichensatz beginnt. Um mit eigenen Zeichensätzen zu arbeiten, muß noch ein wenig Adressrechnung betrieben werden. CHARS sagt, wo das x.te Zeichen mit seinen 8 Bytes (-256) beginnt. Dazu ist der Code des Zeichens zu nehmen; das erste ASCII-Zeichen "Space" beginnt also mit Code 32. 32 \* 8 Bytestellen sind sozusagen der Nullpunkt für ein konkretes Zeichen. Den CHARSTART und das erste Byte (CHAR) eines in Stringvariable IS gespeicherten Zeichens bekommt man daher so:

```
(CHARS ist gleich Adresse 23606 + 23607)
CHARSTART = PEEK 23606 + 256*PEEK 23607
CHAR = 8 * CODE IS + CHARSTART
```

Die Bitkombination läßt sich zum Zweck der Manipulation aus jedem der folgenden 8 Bytes herausfiltern. Befassen wir uns zunächst mit der Gewinnung und Veränderung des Zeichensatzes, ehe wir die Generierung eines neuen beschreiben.

Da CHARS eine veränderbare Adresse ist, kann der Zugriff auf selbstgeschaffene Sets dem System aufgezwungen werden. Sinnvollerweise wird man den eigenen Zeichensatz in einen geschützten RAM-Bereich legen, um ihn zu sichern.

Am einfachsten ist die Manipulation, wenn auf den bestehenden Zeichensatz zurück gegriffen wird. Zuerst werden wir diesen Weg wählen, um mittels Pixelplot ein paar interessante Effekte zu erzielen. Der Aufwand ist am geringsten bei der Vergrößerung bzw. Verzerrung von Schrift. Das geht ohne umfangreiche DATA-Tipperei und Einpoken neuer Zeichenbytes.

Im ersten Beispiel bedienen wir uns gar des simplen PRINT-Statements; wie erzeugt man Riesenbuchstaben, BIG PRINT genannt ?

#### BIG PRINT oder die Erzeugung vo Riesenlettern

BIG PRINT ist die Stammroutine für die folgenden Programme. Sie wird nur wenig modifiziert; wer BIG PRINT "gefressen" hat, kann vielerlei verblüffende Effekte auf dem Bildschirm erreichen.

Zuerst wird mittels des Zeichenbytes die Pixelzeile für je 8 Punkte angewählt. In der äußeren Schleife (mit Variable Y) werden innerhalb des abzuarbeitenden Teststrings (Variable N) diese 8 Zeilenbytes abgefragt. Die eingebettete innere Schleife (mit Variable Z) erschließt die Besetzung der Spalten einer

Zeile. Wie geschieht das ?

Man fragt einfach die 2er Potenzen ab. Ist der Bytewert (als Stellenwertigkeit) größer als sie, ist ein Pixel von links nach rechts gesehen vorhanden. Ansonsten bleibt die Matrix an der Stelle leer. Anschließend wird der Bytewert um die "Wertigkeit" vermindert und mit der Restgröße weitergerechnet.

In der Routine erfolgt die Umsetzung von "besetzt" und "unbesetzt" durch Ausgabe eines schwarzen Vierecks, wie es als Grafikzeichen zur Verfügung steht. Man hätte beim Ausprinten auch eine aufwendige PRINT AT - Steuerung praktizieren können, worauf verzichtet wurde. Das PRINT AT in Zeile 31 ist jedoch notwendig, weil nach Zeilende auf die nächste Druckzeile vorgeschoben werden muß (Nach Abarbeiten der Z-Schleife steht Z auf -1, der Folgebefehl entspricht mit -Z = +1). Die Spaltenposition ergibt sich aus der Buchstabenanzahl von IS; da um Faktor 8 je Pixel vergrößert wird, ist N\*8 einzusetzen..

```

10 REM 8-fach Vergrößerung v
11 ON Buchstaben
12 LET start=10
13 LET is="LOOK"
14 FOR n=0 TO LEN is-1
15   LET zeile=start
16   PRINT AT zeile,n*8;PEEK 23
17   FOR v=0 TO 7
18     LET x=PEEK (char*v)
19     FOR z=7 TO 0 STEP -1
20       IF (x-2+z)>0 THEN PRI
21         NT "█";
22       GOTO TX=X-2+z
23     PRINT " ";
24   NEXT z
25   LET zeile=zeile-Z
26   PRINT AT zeile,n*8;
27 NEXT v
28 NEXT n

```

Look

Wenn die Riesenbuchstaben zu groß sind, kann auch mit einem anderen Faktor arbeiten. Allerdings muß dann pixelgenau positioniert werden. Bei der nächsten Routine wird denn auch geplottet. (Würde man keinerlei Vergrößerung wählen, würden die Buchstaben Pixel für Pixel gemalt.)

Wählt man Faktor 2 (Zweifach-Vergrößerung), wird ein doppelter Pixelabstand gemacht (s. Zeile 30 + 45). Pro Buchstabe werden also nicht 8, sondern 16 Pixel benötigt. Es wird um den Kernpixel herumgeplottet, sodaß eine 2 x 2 - Matrix entsteht. Variable Sp und Zei geben die Kernposition bei der Allokation ab. Man beachte, daß beim PLOT im Gegensatz zum PRINT Spalte und Zeile vertauscht sind - so will es das SINCLAIR-Basic halt..

```

10 REM Buchstabenvergrößerung 2-fach
11
12
13
14 REM Startzeile Zeile 10
15 LET start=10
16 LET is="Test 2 fach"
17 REM Test 17 in 9
18 LET zeile=175-start*8
19 LET charstart=PEEK 23606+25
20 LET char=PEEK 23607
21
22
23
24
25 FOR n=0 TO LEN is-1
26   LET ts=is(n+1)
27   LET sp=1,zei
28   LET sp=n*16
29   LET char=8*CODE ts+charstart
30   FOR v=0 TO 7
31     FOR z=7 TO 0 STEP -1
32       IF (x-2+z)>0 THEN PLOT sp,
33         ze
34       PLOT sp,zei+1
35       PLOT sp,zei-1
36       PLOT sp,zei+1
37       PLOT sp,zei-1
38       LET sp=sp+2
39     NEXT z
40     LET ze=zei-2
41     LET sp=sp-16
42   NEXT v
43 NEXT n

```

Test 2 fach

Weitere Manipulationen

Mit der angelistete Ploteroutine läßt sich noch mehr anstellen. Als nächstes gehen wir einmal in die Breite. Was ist dazu zu tun ?

Zunächst vereinfachen wir dem pixelrechnenden SINCLAIR die Arbeit, indem bei Leerzeichen gleich die Spalte (s. Zeile 30) fortgezählt und nichts geplottet wird. Weiter verbessern wir das Ganze, indem wir es gegen "out of screen" versichern (s. Zeile 50). Nun zur Schriftverbreiterung.

Dazu wird ein Breitenfaktor b eingeführt; er wäre bei Nichtverbreiterung Null (s. Zeile 33). Da Spalte Sp mit 1 fortgezählt wird, käme nichts hinzu. Die Breitschrift wird erzeugt, indem bei jedem zweiten Pixel dieses nochmals geplottet wird (s. Zeile 40). Besetzt man jeden Spaltenwert zweifach, würde die Schrift noch breiter; so nimmt sie nicht soviel Platz weg.

Um den alternierenden Plot zu erreichen, wird b als Ein/Aus-Schalter geführt (vgl. Zeile 43). Dadurch wird etwas wie "enlarged condensed" bei Normalpapierdruckern realisiert. Man experimentiere ruhig ein wenig mit der folgenden Routine.

```

10 REM BREITSCHRIFT
11
12
13
14 INPUT "startzeile";start
15 LET SP=0
16 INPUT "22-25 Zeichen/Zeile";i
17 LET zeile=175-start*8
18 LET charstart=PEEK 23606+25
19 LET char=PEEK 23607
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

stark hoch + gross BREIT

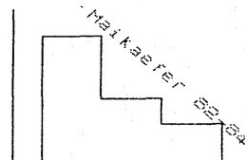
Das Programmchen enthält noch eine hübsche "Starterleichterung": Bei Eingabe von Startzeile/-spalte erscheint die Position direkt auf dem Screen. Man hat die Möglichkeit der Korrektur. Erst wenn alles klar ist, wird der Beschriftungstext abgearbeitet.

```
10 REM PROGRAMM 100101  
11  
12  
14 INPUT "Startzeile ?";zeile  
16 INPUT "Startspalte ?";spalt
```

```

17 PRINT "AT zeile,spalte, OVER
18 INPUT "Zeile: ";zeile
19 IF zeile<1 THEN RETURN":1
20 PRINT CHR$(8); OVER 1; "
21 IF zeile<1 THEN GO TO 14
22 INPUT "Spalte: ";sp
23 LET zeile:=15-zeile*8
24 LET sp:=sp*spalte*8
25 GOSUB 23607
26 *PEEK 23607
27
28 LET zeil:=zeile
29
30 FOR n=0 TO CODE is-1
31   LET char=8*CODE is+charsta
32   FOR y=0 TO 7
33     LET x=PEEK (char+y)
34     FOR z=0 TO 7
35       IF (x-z)*z=1 THEN
36         THEN PLOT sp,zeil
37         LET x=x-2
38         LET sp=sp+1
39         LET zeil=zeil-1
40
41       LET zeil=zeil+7
42       sp=sp-9
43     NEXT y
44     LET sp=sp+15
45     zeil=zeil+1
46   NEXT n
47   PRINT
48   GOTO 23

```



> Skalen können koordinatengerecht beschriftet werden

> Für Diagramme wird wertvoller Plotraum gewonnen (vertikal 255  
Pixels !)

- > Funktionen lassen sich "open ended" (immer weiter nach unten fortgeschrieben) darstellen

> Der Bildschirm bekommt ein angenehmes 32 Zeilen-Format

> Durch Hardcopies auf Normalpapierdrucker lassen sich bei mehreren Screens beliebige Zeilenbreiten realisieren (Ausdruck von Text in "Schüben")

Aber beschäftigen wir uns erst mit der langsamen Pixellösung, ehe wir einen eigenen Vertikalschrift-Zeichensatz generieren.

HALLO SINGULAR-  
HANS! Ich bin  
verheiratet und  
lebe in der  
Stadt. Bei der  
Arbeit bin ich  
aber sehr  
ruhig. Jeder  
Tag ist für mich  
ein Abenteuer.  
Ich gehe

80

Wie ist vorzugehen ?

Die einzelnen Schritte zur Lösung von Vertikalschrift sind wie folgt:

- Für das Gesagte findet der Leser zuerst eine Generierungsroutine, die auf der bisherigen "Potenzrechnung" aufbaut. Da der gesamte ASCII-Bereich "umgelegt" wird, hat der Rechner viel zu tun: 96 Buchstaben a Z Bytes ergibt 768 Acht-Bit-Positionen; wozu 8 Wertigkeiten pro Pixel zu errechnen sind, was 6144 Arithmetikoperationen ausmacht. Kein Wunder also, wenn die Generierung des Vertikal-Sets ca. 15 Minuten in Anspruch nimmt. Das Ende der Kaffeepause wird mit ein paar Bytes SIGDOLLART...:

81

```

38 FOR n=1 TO 5
  PRINT "n=";n;";"
  NEXT n
PRINT "n=";n;";"
NEXT n
PRINT FLASH 1;"Generierung
beendet"
40 REM Umlagen des Char.Sets m
1 POKE 23606,88;POKE 23607,
251;ZURUECK a) POKE 23606,0
;POKE 23607,60)

```

Wie die Bemerkung in Zeile 40 besagt, ist das Abspeichern des Character Sets sehr zu empfehlen. Mit der angegebenen Adresse läuft das so:

SAVE "UCHAR" CODE 64500,768

Um nun an die Vertikalschrift zu kommen, muß Systemvariable CHARS auf Adresse 64500 "umgebogen" werden:

POKE 23606,88; POKE 23607,251

Ab sofort erscheinen Buchstaben "gedreht", also auch alle Systemnachrichten, Listingzeilen usw. Wie kann man nun mit der neuen Schrift arbeiten? Wie erreicht man einen "sauberen" Ausdruck von Text?

Arbeiten mit UCHAR (Vertikalcharacters)

Um in den Genuß eines völlig neuen Screen-Gefühls zu kommen, sind einige Vorbereitungen zu treffen. Ehe man den Fernseher kippt und unvertraute 32 Zeilen mit 22 Spalten betrachtet, soll das Druckprogramm erklärt werden. Es gibt zudem ein paar Hilfen, so z.B. Text in mehreren Schüben auszudrucken.

Zuerst ist das RAM mit CLEAR 64599 herabzusetzen, um UCHAR einzulesen. Das geschieht mit LOAD UCHAR, CODE. Nach Verbiegen des Char-Vektors (s.o.) kann dann ein vorhandener Text vertikal geprintet werden. Die Printroutine beinhaltet zwei Vorschläge:

1. Anzeige eines vorhandenen Textes (in AS) mit eigener (auf "vertikal" umgestellten) Input-Aufforderung. - Der erste Teil des Printprogramms richtet Spalten und Zeilen neu aus. Da der Textstring AS nicht einfach - mit PRINT AS - angedruckt werden kann, ist er spaltenweise von oben nach unten zu "kippen". Die Routine verkraftet zudem (s. Zeilen 16 - 34) auch einen Seitenwechsel, falls der Bildschirm voll ist.

2. Ausdruck eines Textes mittels einer eigenen Tabelle (quasi als Zwischenspeicher). Zeile 36 - 54 füllt zum schubweisen Print von zwei Screensseiten nach aufwendiger Umsetzung in eine Tabelle. Sie kann danach ohne weiteren Aufwand beliebig oft angedruckt werden (Der Array wird einfach spaltenweise "heruntergenudelt"). - Bei vollem Screen wird der Text in zwei Schüben auf den Drucker kopiert (Memo: LPRINT funktioniert nicht, da der Drucker den geänderten Zeichensatz nicht hat!).

```

10 REM Direktausdruck der Ver
ikalcharacters
12
14 REM memo 1. CLEAR 64599 2. UC
HARS-CODES laden 3. Textstri
ng AS laden Variable CHAR
S auf Addr 64500 Umlagen
16
18 LET s=31
19 LET z=0
20 FOR n=1 TO LEN s
  PRINT AT z,s; s(n)
  LET z=(z+1) AND (z<21)
  LET s=s-(z=0)
  IF s=0 THEN INPUT "NRUTER!
etwas evtl. in
if s=31 THEN CLS
GO TO 32
30 IF s<>" THEN LET n=LEN s
32 NEXT n
34 REM Tabellenloesung
  (Vorteil:
  Moegl., den Text zu "splitte
n" in mehreren Schüben a
uszudrucken)
36 LET dim=44
40 DIM t$(dim,32)
42 LET n=1
44 FOR s=32 TO 1 STEP -1
  FOR z=1 TO dim
    LET t$(z,s)=s(n)
  IF n>LEN s THEN LET z=di
  LET s=s-1
52 NEXT z
54 NEXT s
56 INPUT "PRINT ? =>RETURN":is
IF is<>" THEN GO TO 62
57 LET s=22
58 FOR n=n TO s
  PRINT t$(n)
  NEXT n
  IF dim>n THEN COPY
  GO TO 56
62 COPY
110 POKE 23606,0
    POKE 23607,60

```

Dieser Test zeigt, wie Vertikalbuchstaben eingesetzt werden. Man dreht also den Bildschirm und genießt ein völlig neues Screen-Gefühl. Die Einsatzmöglichkeiten sind aber noch vielseitiger: beispielsweise ist es nun möglich Druckausgaben unbegrenzter Breite zu realisieren. Man muss nur den Text in mehreren Schüben ausdrucken. Aber auch Grafiken mit 256 pixels können vertikal in beliebiger Länge erstellt und beschriftet werden. Und Viel mehr....

Der Vorteil der Tabellenlösung liegt in der Druckeroption. Es ist so möglich, auf einen Schlag 64 Zeichenzeilen zu Papier zu bringen. Was auch für SINCLAIR-Drucker (Alphacom 32/TIMEX, Seikosha 50 S etc.) gilt..

Wenn das Verfahren der Zeichengenerierung in Basic zu umständlich

ist, für den haben wir noch eine spezielle Maschinencode-Routine parat. Da läuft das Ganze mit nur 28 DATA-Codes ab. Das einfache MC-Programm schaltet blitzschnell auf Fettdruck um. Dazu wird der Spectrum-Zeichensatz kopiert und (Assembler SLR) bitweise "geshifft". Ein neues Pixel umzeichnet zum vorhandenen die Buchstabenkontur und erbringt so die proper Schrift.

```

990 CLEAR 64568
991 FOR f=65338 TO 65365: READ
a: POKE f,a: NEXT f
992 RANDOMIZE USR 65338
993 DATA 33,6,61,17,37,252,1,0,
3,126,293,47,168,18,19,35,11,12,
176,32,244,33,57,251,34,64,92,2
61

```

Zum Schluß noch einen Hinweis: Wer wieder seinen normalen Zeichensatz haben will - auch in Fall der mutwilligen Schriftverfälschung zwecks Listschutz! -, erreicht dies mit

POKE 23606,0: POKE 23607,60

Soviel zu Zeichen und Zeichensätzen. Wenden wir uns nun der Bildschirmdarstellung von Texten zu. Auch da gibt es viel Interessantes - bis hin zur einfachen Textverarbeitung.

#### Texte auf dem Bildschirm

##### Inputs

Im folgenden werden wir uns behutsam dem Feld der Textverarbeitung nähern. Schritt für Schritt gehen wir von der Input-Anzeige bis hin zu weiterer Textbehandlung vor; im zweiten Teil des Kapitels wird dann die Entwicklung eines (bescheidenen) Textprogramms geschildert.

Einfachstes Texthandling erfolgt zunächst per INPUT. Hinter dem Befehl verbirgt sich mehr als man gemeinhin ehnt - ein kompletter Zeileneditor, der von System zur Verfügung gestellt wird. Arbeitet man mit Eingabemasken, kann man nur neidisch auf die guten Insert und Delete-Funktionen blicken. Was liegt also näher, als direkt den oberen Screen-Teil mit Input zu beschreiben? Dazu sei als Stichwort INPUT AT (oberer Bildschirm) genannt; wie ist sowas zu realisieren?

1. Überlegung - Man wechselt die "streams", um 'Unten' die Eingabe und 'Oben' die Anzeige zu bekommen.

INPUT #2;"BITTE EINGABE ";#1;N;#2;(N)

Damit hat man zwar clever die Screen-Bereiche getauscht, eine direkte Anzeige der Inputzeile ist 'Oben' aber nicht erreicht. Zudem geht es etwas weniger elegant auch so:

INPUT "BITTE EINGABEZAHL ";N: PRINT AT 2,0;N

2. Überlegung - INPUT kann wie PRINT mit AT positioniert werden, da liegt ein anderer Bedenke nahe. Zwar ist der obere Bildschirm nicht direkt angesteuerbar, aber der Eingabebereich läßt sich soweit hochschieben, daß die Eingabe an jede gewünschte Stelle zu liegen kommt.

INPUT AT 21,0;AT 0,0;"ZAHL 7";N

Nachteil - eine aufgebaute Maske wird zerstört, da alles auf dem Screen weggeschoben wird.

3. Überlegung - Wir nehmen die Inputfunktion im oberen Bereich nach. Der Weg läuft über INKEYS und beinhaltet nach Erzeugen einer Maske folgende Schritte:

- > INKEYS-Abfrage
- > INPUT-Simulation (Cursor,Editing)
- > Ergebnisübernahme nach ENTER (als Quittierung)

```

10 REM INPUT AT 21,0;AT 0,0;"ZAHL 7";N
16 LET Z=5
LET S=14
LET len=0
DIM i$(len)
17 PAUSE 0
20 PRINT AT 2,3;"*****"
*****;AT 3,3;"*;*;*;*;*;*;*;*;*;*";AT 4,3;"*;*;*;*;*";AT 5,3;"*;*;*;*";AT 6,3;"*;*;*";AT 7,3;"*;*";AT 8,3;"*";AT 9,3;" ";AT 10,3;" ";AT 11,3;" ";AT 12,3;" ";AT 13,3;" ";AT 14,3;" ";AT 15,3;" ";AT 16,3;" ";AT 17,3;" ";AT 18,3;" ";AT 19,3;" ";AT 20,3;" ";AT 21,3;" ";AT 22,3;" ";AT 23,3;" ";AT 24,3;" ";AT 25,3;" ";AT 26,3;" ";AT 27,3;" ";AT 28,3;" ";AT 29,3;" ";AT 30,3;" ";AT 31,3;" ";AT 32,3;" ";AT 33,3;" ";AT 34,3;" ";AT 35,3;" ";AT 36,3;" ";AT 37,3;" ";AT 38,3;" ";AT 39,3;" ";AT 40,3;" ";AT 41,3;" ";AT 42,3;" ";AT 43,3;" ";AT 44,3;" ";AT 45,3;" ";AT 46,3;" ";AT 47,3;" ";AT 48,3;" ";AT 49,3;" ";AT 50,3;" ";AT 51,3;" ";AT 52,3;" ";AT 53,3;" ";AT 54,3;" ";AT 55,3;" ";AT 56,3;" ";AT 57,3;" ";AT 58,3;" ";AT 59,3;" ";AT 60,3;" ";AT 61,3;" ";AT 62,3;" ";AT 63,3;" ";AT 64,3;" ";AT 65,3;" ";AT 66,3;" ";AT 67,3;" ";AT 68,3;" ";AT 69,3;" ";AT 70,3;" ";AT 71,3;" ";AT 72,3;" ";AT 73,3;" ";AT 74,3;" ";AT 75,3;" ";AT 76,3;" ";AT 77,3;" ";AT 78,3;" ";AT 79,3;" ";AT 80,3;" ";AT 81,3;" ";AT 82,3;" ";AT 83,3;" ";AT 84,3;" ";AT 85,3;" ";AT 86,3;" ";AT 87,3;" ";AT 88,3;" ";AT 89,3;" ";AT 90,3;" ";AT 91,3;" ";AT 92,3;" ";AT 93,3;" ";AT 94,3;" ";AT 95,3;" ";AT 96,3;" ";AT 97,3;" ";AT 98,3;" ";AT 99,3;" ";AT 100,3;" ";AT 101,3;" ";AT 102,3;" ";AT 103,3;" ";AT 104,3;" ";AT 105,3;" ";AT 106,3;" ";AT 107,3;" ";AT 108,3;" ";AT 109,3;" ";AT 110,3;" ";AT 111,3;" ";AT 112,3;" ";AT 113,3;" ";AT 114,3;" ";AT 115,3;" ";AT 116,3;" ";AT 117,3;" ";AT 118,3;" ";AT 119,3;" ";AT 120,3;" ";AT 121,3;" ";AT 122,3;" ";AT 123,3;" ";AT 124,3;" ";AT 125,3;" ";AT 126,3;" ";AT 127,3;" ";AT 128,3;" ";AT 129,3;" ";AT 130,3;" ";AT 131,3;" ";AT 132,3;" ";AT 133,3;" ";AT 134,3;" ";AT 135,3;" ";AT 136,3;" ";AT 137,3;" ";AT 138,3;" ";AT 139,3;" ";AT 140,3;" ";AT 141,3;" ";AT 142,3;" ";AT 143,3;" ";AT 144,3;" ";AT 145,3;" ";AT 146,3;" ";AT 147,3;" ";AT 148,3;" ";AT 149,3;" ";AT 150,3;" ";AT 151,3;" ";AT 152,3;" ";AT 153,3;" ";AT 154,3;" ";AT 155,3;" ";AT 156,3;" ";AT 157,3;" ";AT 158,3;" ";AT 159,3;" ";AT 160,3;" ";AT 161,3;" ";AT 162,3;" ";AT 163,3;" ";AT 164,3;" ";AT 165,3;" ";AT 166,3;" ";AT 167,3;" ";AT 168,3;" ";AT 169,3;" ";AT 170,3;" ";AT 171,3;" ";AT 172,3;" ";AT 173,3;" ";AT 174,3;" ";AT 175,3;" ";AT 176,3;" ";AT 177,3;" ";AT 178,3;" ";AT 179,3;" ";AT 180,3;" ";AT 181,3;" ";AT 182,3;" ";AT 183,3;" ";AT 184,3;" ";AT 185,3;" ";AT 186,3;" ";AT 187,3;" ";AT 188,3;" ";AT 189,3;" ";AT 190,3;" ";AT 191,3;" ";AT 192,3;" ";AT 193,3;" ";AT 194,3;" ";AT 195,3;" ";AT 196,3;" ";AT 197,3;" ";AT 198,3;" ";AT 199,3;" ";AT 200,3;" ";AT 201,3;" ";AT 202,3;" ";AT 203,3;" ";AT 204,3;" ";AT 205,3;" ";AT 206,3;" ";AT 207,3;" ";AT 208,3;" ";AT 209,3;" ";AT 210,3;" ";AT 211,3;" ";AT 212,3;" ";AT 213,3;" ";AT 214,3;" ";AT 215,3;" ";AT 216,3;" ";AT 217,3;" ";AT 218,3;" ";AT 219,3;" ";AT 220,3;" ";AT 221,3;" ";AT 222,3;" ";AT 223,3;" ";AT 224,3;" ";AT 225,3;" ";AT 226,3;" ";AT 227,3;" ";AT 228,3;" ";AT 229,3;" ";AT 230,3;" ";AT 231,3;" ";AT 232,3;" ";AT 233,3;" ";AT 234,3;" ";AT 235,3;" ";AT 236,3;" ";AT 237,3;" ";AT 238,3;" ";AT 239,3;" ";AT 240,3;" ";AT 241,3;" ";AT 242,3;" ";AT 243,3;" ";AT 244,3;" ";AT 245,3;" ";AT 246,3;" ";AT 247,3;" ";AT 248,3;" ";AT 249,3;" ";AT 250,3;" ";AT 251,3;" ";AT 252,3;" ";AT 253,3;" ";AT 254,3;" ";AT 255,3;" ";AT 256,3;" ";AT 257,3;" ";AT 258,3;" ";AT 259,3;" ";AT 260,3;" ";AT 261,3;" ";AT 262,3;" ";AT 263,3;" ";AT 264,3;" ";AT 265,3;" ";AT 266,3;" ";AT 267,3;" ";AT 268,3;" ";AT 269,3;" ";AT 270,3;" ";AT 271,3;" ";AT 272,3;" ";AT 273,3;" ";AT 274,3;" ";AT 275,3;" ";AT 276,3;" ";AT 277,3;" ";AT 278,3;" ";AT 279,3;" ";AT 280,3;" ";AT 281,3;" ";AT 282,3;" ";AT 283,3;" ";AT 284,3;" ";AT 285,3;" ";AT 286,3;" ";AT 287,3;" ";AT 288,3;" ";AT 289,3;" ";AT 290,3;" ";AT 291,3;" ";AT 292,3;" ";AT 293,3;" ";AT 294,3;" ";AT 295,3;" ";AT 296,3;" ";AT 297,3;" ";AT 298,3;" ";AT 299,3;" ";AT 300,3;" ";AT 301,3;" ";AT 302,3;" ";AT 303,3;" ";AT 304,3;" ";AT 305,3;" ";AT 306,3;" ";AT 307,3;" ";AT 308,3;" ";AT 309,3;" ";AT 310,3;" ";AT 311,3;" ";AT 312,3;" ";AT 313,3;" ";AT 314,3;" ";AT 315,3;" ";AT 316,3;" ";AT 317,3;" ";AT 318,3;" ";AT 319,3;" ";AT 320,3;" ";AT 321,3;" ";AT 322,3;" ";AT 323,3;" ";AT 324,3;" ";AT 325,3;" ";AT 326,3;" ";AT 327,3;" ";AT 328,3;" ";AT 329,3;" ";AT 330,3;" ";AT 331,3;" ";AT 332,3;" ";AT 333,3;" ";AT 334,3;" ";AT 335,3;" ";AT 336,3;" ";AT 337,3;" ";AT 338,3;" ";AT 339,3;" ";AT 340,3;" ";AT 341,3;" ";AT 342,3;" ";AT 343,3;" ";AT 344,3;" ";AT 345,3;" ";AT 346,3;" ";AT 347,3;" ";AT 348,3;" ";AT 349,3;" ";AT 350,3;" ";AT 351,3;" ";AT 352,3;" ";AT 353,3;" ";AT 354,3;" ";AT 355,3;" ";AT 356,3;" ";AT 357,3;" ";AT 358,3;" ";AT 359,3;" ";AT 360,3;" ";AT 361,3;" ";AT 362,3;" ";AT 363,3;" ";AT 364,3;" ";AT 365,3;" ";AT 366,3;" ";AT 367,3;" ";AT 368,3;" ";AT 369,3;" ";AT 370,3;" ";AT 371,3;" ";AT 372,3;" ";AT 373,3;" ";AT 374,3;" ";AT 375,3;" ";AT 376,3;" ";AT 377,3;" ";AT 378,3;" ";AT 379,3;" ";AT 380,3;" ";AT 381,3;" ";AT 382,3;" ";AT 383,3;" ";AT 384,3;" ";AT 385,3;" ";AT 386,3;" ";AT 387,3;" ";AT 388,3;" ";AT 389,3;" ";AT 390,3;" ";AT 391,3;" ";AT 392,3;" ";AT 393,3;" ";AT 394,3;" ";AT 395,3;" ";AT 396,3;" ";AT 397,3;" ";AT 398,3;" ";AT 399,3;" ";AT 400,3;" ";AT 401,3;" ";AT 402,3;" ";AT 403,3;" ";AT 404,3;" ";AT 405,3;" ";AT 406,3;" ";AT 407,3;" ";AT 408,3;" ";AT 409,3;" ";AT 410,3;" ";AT 411,3;" ";AT 412,3;" ";AT 413,3;" ";AT 414,3;" ";AT 415,3;" ";AT 416,3;" ";AT 417,3;" ";AT 418,3;" ";AT 419,3;" ";AT 420,3;" ";AT 421,3;" ";AT 422,3;" ";AT 423,3;" ";AT 424,3;" ";AT 425,3;" ";AT 426,3;" ";AT 427,3;" ";AT 428,3;" ";AT 429,3;" ";AT 430,3;" ";AT 431,3;" ";AT 432,3;" ";AT 433,3;" ";AT 434,3;" ";AT 435,3;" ";AT 436,3;" ";AT 437,3;" ";AT 438,3;" ";AT 439,3;" ";AT 440,3;" ";AT 441,3;" ";AT 442,3;" ";AT 443,3;" ";AT 444,3;" ";AT 445,3;" ";AT 446,3;" ";AT 447,3;" ";AT 448,3;" ";AT 449,3;" ";AT 450,3;" ";AT 451,3;" ";AT 452,3;" ";AT 453,3;" ";AT 454,3;" ";AT 455,3;" ";AT 456,3;" ";AT 457,3;" ";AT 458,3;" ";AT 459,3;" ";AT 460,3;" ";AT 461,3;" ";AT 462,3;" ";AT 463,3;" ";AT 464,3;" ";AT 465,3;" ";AT 466,3;" ";AT 467,3;" ";AT 468,3;" ";AT 469,3;" ";AT 470,3;" ";AT 471,3;" ";AT 472,3;" ";AT 473,3;" ";AT 474,3;" ";AT 475,3;" ";AT 476,3;" ";AT 477,3;" ";AT 478,3;" ";AT 479,3;" ";AT 480,3;" ";AT 481,3;" ";AT 482,3;" ";AT 483,3;" ";AT 484,3;" ";AT 485,3;" ";AT 486,3;" ";AT 487,3;" ";AT 488,3;" ";AT 489,3;" ";AT 490,3;" ";AT 491,3;" ";AT 492,3;" ";AT 493,3;" ";AT 494,3;" ";AT 495,3;" ";AT 496,3;" ";AT 497,3;" ";AT 498,3;" ";AT 499,3;" ";AT 500,3;" ";AT 501,3;" ";AT 502,3;" ";AT 503,3;" ";AT 504,3;" ";AT 505,3;" ";AT 506,3;" ";AT 507,3;" ";AT 508,3;" ";AT 509,3;" ";AT 510,3;" ";AT 511,3;" ";AT 512,3;" ";AT 513,3;" ";AT 514,3;" ";AT 515,3;" ";AT 516,3;" ";AT 517,3;" ";AT 518,3;" ";AT 519,3;" ";AT 520,3;" ";AT 521,3;" ";AT 522,3;" ";AT 523,3;" ";AT 524,3;" ";AT 525,3;" ";AT 526,3;" ";AT 527,3;" ";AT 528,3;" ";AT 529,3;" ";AT 530,3;" ";AT 531,3;" ";AT 532,3;" ";AT 533,3;" ";AT 534,3;" ";AT 535,3;" ";AT 536,3;" ";AT 537,3;" ";AT 538,3;" ";AT 539,3;" ";AT 540,3;" ";AT 541,3;" ";AT 542,3;" ";AT 543,3;" ";AT 544,3;" ";AT 545,3;" ";AT 546,3;" ";AT 547,3;" ";AT 548,3;" ";AT 549,3;" ";AT 550,3;" ";AT 551,3;" ";AT 552,3;" ";AT 553,3;" ";AT 554,3;" ";AT 555,3;" ";AT 556,3;" ";AT 557,3;" ";AT 558,3;" ";AT 559,3;" ";AT 560,3;" ";AT 561,3;" ";AT 562,3;" ";AT 563,3;" ";AT 564,3;" ";AT 565,3;" ";AT 566,3;" ";AT 567,3;" ";AT 568,3;" ";AT 569,3;" ";AT 570,3;" ";AT 571,3;" ";AT 572,3;" ";AT 573,3;" ";AT 574,3;" ";AT 575,3;" ";AT 576,3;" ";AT 577,3;" ";AT 578,3;" ";AT 579,3;" ";AT 580,3;" ";AT 581,3;" ";AT 582,3;" ";AT 583,3;" ";AT 584,3;" ";AT 585,3;" ";AT 586,3;" ";AT 587,3;" ";AT 588,3;" ";AT 589,3;" ";AT 590,3;" ";AT 591,3;" ";AT 592,3;" ";AT 593,3;" ";AT 594,3;" ";AT 595,3;" ";AT 596,3;" ";AT 597,3;" ";AT 598,3;" ";AT 599,3;" ";AT 600,3;" ";AT 601,3;" ";AT 602,3;" ";AT 603,3;" ";AT 604,3;" ";AT 605,3;" ";AT 606,3;" ";AT 607,3;" ";AT 608,3;" ";AT 609,3;" ";AT 610,3;" ";AT 611,3;" ";AT 612,3;" ";AT 613,3;" ";AT 614,3;" ";AT 615,3;" ";AT 616,3;" ";AT 617,3;" ";AT 618,3;" ";AT 619,3;" ";AT 620,3;" ";AT 621,3;" ";AT 622,3;" ";AT 623,3;" ";AT 624,3;" ";AT 625,3;" ";AT 626,3;" ";AT 627,3;" ";AT 628,3;" ";AT 629,3;" ";AT 630,3;" ";AT 631,3;" ";AT 632,3;" ";AT 633,3;" ";AT 634,3;" ";AT 635,3;" ";AT 636,3;" ";AT 637,3;" ";AT 638,3;" ";AT 639,3;" ";AT 640,3;" ";AT 641,3;" ";AT 642,3;" ";AT 643,3;" ";AT 644,3;" ";AT 645,3;" ";AT 646,3;" ";AT 647,3;" ";AT 648,3;" ";AT 649,3;" ";AT 650,3;" ";AT 651,3;" ";AT 652,3;" ";AT 653,3;" ";AT 654,3;" ";AT 655,3;" ";AT 656,3;" ";AT 657,3;" ";AT 658,3;" ";AT 659,3;" ";AT 660,3;" ";AT 661,3;" ";AT 662,3;" ";AT 663,3;" ";AT 664,3;" ";AT 665,3;" ";AT 666,3;" ";AT 667,3;" ";AT 668,3;" ";AT 669,3;" ";AT 670,3;" ";AT 671,3;" ";AT 672,3;" ";AT 673,3;" ";AT 674,3;" ";AT 675,3;" ";AT 676,3;" ";AT 677,3;" ";AT 678,3;" ";AT 679,3;" ";AT 680,3;" ";AT 681,3;" ";AT 682,3;" ";AT 683,3;" ";AT 684,3;" ";AT 685,3;" ";AT 686,3;" ";AT 687,3;" ";AT 688,3;" ";AT 689,3;" ";AT 690,3;" ";AT 691,3;" ";AT 692,3;" ";AT 693,3;" ";AT 694,3;" ";AT 695,3;" ";AT 696,3;" ";AT 697,3;" ";AT 698,3;" ";AT 699,3;" ";AT 700,3;" ";AT 701,3;" ";AT 702,3;" ";AT 703,3;" ";AT 704,3;" ";AT 705,3;" ";AT 706,3;" ";AT 707,3;" ";AT 708,3;" ";AT 709,3;" ";AT 710,3;" ";AT 711,3;" ";AT 712,3;" ";AT 713,3;" ";AT 714,3;" ";AT 715,3;" ";AT 716,3;" ";AT 717,3;" ";AT 718,3;" ";AT 719,3;" ";AT 720,3;" ";AT 721,3;" ";AT 722,3;" ";AT 723,3;" ";AT 724,3;" ";AT 725,3;" ";AT 726,3;" ";AT 727,3;" ";AT 728,3;" ";AT 729,3;" ";AT 730,3;" ";AT 731,3;" ";AT 732,3;" ";AT 733,3;" ";AT 734,3;" ";AT 735,3;" ";AT 736,3;" ";AT 737,3;" ";AT 738,3;" ";AT 739,3;" ";AT 740,3;" ";AT 741,3;" ";AT 742,3;" ";AT 743,3;" ";AT 744,3;" ";AT 745,3;" ";AT 746,3;" ";AT 747,3;" ";AT 748,3;" ";AT 749,3;" ";AT 750,3;" ";AT 751,3;" ";AT 752,3;" ";AT 753,3;" ";AT 754,3;" ";AT 755,3;" ";AT 756,3;" ";AT 757,3;" ";AT 758,3;" ";AT 759,3;" ";AT 760,3;" ";AT 761,3;" ";AT 762,3;" ";AT 763,3;" ";AT 764,3;" ";AT 765,3;" ";AT 766,3;" ";AT 767,3;" ";AT 768,3;" ";AT 769,3;" ";AT 770,3;" ";AT 771,3;" ";AT 772,3;" ";AT 773,3;" ";AT 774,3;" ";AT 775,3;" ";AT 776,3;" ";AT 777,3;" ";AT 778,3;" ";AT 779,3;" ";AT 780,3;" ";AT 781,3;" ";AT 782,3;" ";AT 783,3;" ";AT 784,3;" ";AT 785,3;" ";AT 786,3;" ";AT 787,3;" ";AT 788,3;" ";AT 789,3;" ";AT 790,3;" ";AT 791,3;" ";AT 792,3;" ";AT 793,3;" ";AT 794,3;" ";AT 795,3;" ";AT 796,3;" ";AT 797,3;" ";AT 798,3;" ";AT 799,3;" ";AT 800,3;" ";AT 801,3;" ";AT 802,3;" ";AT 803,3;" ";AT 804,3;" ";AT 805,3;" ";AT 806,3;" ";AT 807,3;" ";AT 808,3;" ";AT 809,3;" ";AT 810,3;" ";AT 811,3;" ";AT 812,3;" ";AT 813,3;" ";AT 814,3;" ";AT 815,3;" ";AT 816,3;" ";AT 817,3;" ";AT 818,3;" ";AT 819,3;" ";AT 820,3;" ";AT 821,3;" ";AT 822,3;" ";AT 823,3;" ";AT 824,3;" ";AT 825,3;" ";AT 826,3;" ";AT 827,3;" ";AT 828,3;" ";AT 829,3;" ";AT 830,3;" ";AT 831,3;" ";AT 832,3;" ";AT 833,3;" ";AT 834,3;" ";AT 835,3;" ";AT 836,3;" ";AT 837,3;" ";AT 838,3;" ";AT 839,3;" ";AT 840,3;" ";AT 841,3;" ";AT 842,3;" ";AT 843,3;" ";AT 844,3;" ";AT 845,3;" ";AT 846,3;" ";AT 847,3;" ";AT 848,3;" ";AT 849,3;" ";AT 850,3;" ";AT 851,3;" ";AT 852,3;" ";AT 853,3;" ";AT 854,3;" ";AT 855,3;" ";AT 856,3;" ";AT 857,3;" ";AT 858,3;" ";AT 859,3;" ";AT 860,3;" ";AT 861,3;" ";AT 862,3;" ";AT 863,3;" ";AT 864,3;" ";AT 865,3;" ";AT 866,3;" ";AT 867,3;" ";AT 868,3;" ";AT 869,3;" ";AT 870,3;" ";AT 871,3;" ";AT 872,3;" ";AT 873,3;" ";AT 874,3;" ";AT 875,3;" ";AT 876,3;" ";AT 877,3;" ";AT 878,3;" ";AT 879,3;" ";AT 880,3;" ";AT 881,3;" ";AT 882,3;" ";AT 883,3;" ";AT 884,3;" ";AT 885,3;" ";AT 886,3;" ";AT 887,3;" ";AT 888,3;" ";AT 889,3;" ";AT 890,3;" ";AT 891,3;" ";AT 892,3;" ";AT 893,3;" ";AT 894,3;" ";AT 895,3;" ";AT 896,3;" ";AT 897,3;" ";AT 898,3;" ";AT 899,3;" ";AT 900,3;" ";AT 901,3;" ";AT 902,3;" ";AT 903,3;" ";AT 904,3;" ";AT 905,3;" ";AT 906,3;" ";AT 907,3;" ";AT 908,3;" ";AT 909,3;" ";AT 910,3;" ";AT 911,3;" ";AT 912,3;" ";AT 913,3;" ";AT 914,3;" ";AT 915,3;" ";AT 916,3;" ";AT 917,3;" ";AT 918,3;" ";AT 919,3;" ";AT 920,3;" ";AT 921,3;" ";AT 922,3;" ";AT 923,3;" ";AT 924,3;" ";AT 925,3;" ";AT 926,3;" ";AT 927,3;" ";AT 928,3;" ";AT 929,3;" ";AT 930,3;" ";AT 931,3;" ";AT 932,3;" ";AT 933,3;" ";AT 934,3;" ";AT 935,3;" ";AT 936,3;" ";AT 937,3;" ";AT 938,3;" ";AT 939,3;" ";AT 940,3;" ";AT 941,3;" ";AT 942,3;" ";AT 943,3;" ";AT 944,3;" ";AT 945,3;" ";AT 946,3;" ";AT 947,3;" ";AT 948,3;" ";AT 949,3;" ";AT 950,3;" ";AT 951,3;" ";AT 952,3;" ";AT 953,3;" ";AT 954,3;" ";AT 955,3;" ";AT 956,3;" ";AT 957,3;" ";AT 958,3;" ";AT 959,3;" ";AT 960,3;" ";AT 961,3;" ";AT 962,3;" ";AT 963,3;" ";AT 964,3;" ";AT 965,3;" ";AT 966,3;" ";AT 967,3;" ";AT 968,3;" ";AT 969,3;" ";AT 970,3;" ";AT 971,3;" ";AT 972,3;" ";AT 973,3;" ";AT 974,3;" ";AT 975,3;" ";AT 976,3;" ";AT 977,3;" ";AT 978,3;" ";AT 979,3;" ";AT 980,3;" ";AT 981,3;" ";AT 982,3;" ";AT 983,3;" ";AT 984,3;" ";AT 985,3;" ";AT 986,3;" ";AT 987,3;" ";AT 988,3;" ";AT 989,3;" ";AT 990,3;" ";AT 991,3;" ";AT 992,3;" ";AT 993,3;" ";AT 994,3;" ";AT 995,3;" ";AT 996,3;" ";AT 997,3;" ";AT 998,3;" ";AT 999,3;" ";AT 1000,3;" ";AT 1001,3;" ";AT 1002,3;" ";AT 1003,3;" ";AT 1004,3;" ";AT 1005,3;" ";AT 1006,3;" ";AT 1007,3;" ";AT 1008,3;" ";AT 1009,3;" ";AT 1010,3;" ";AT 1011,3;" ";AT 1012,3;" ";AT 1013,3;" ";AT 1014,3;" ";AT 1015,3;" ";AT 1016,3;" ";AT 1017,3;" ";AT 1018,3;" ";AT 1019,3;" ";AT 1020,3;" ";AT 1021,3;" ";AT 1022,3;" ";AT 1023,3;" ";AT 1024,3;" ";AT 1025,3;" ";AT 1026,3;" ";AT 1027,3;" ";AT 1028,3;" ";AT 1029,3;" ";AT 1030,3;" ";AT 1031,3;" ";AT 1032,3;" ";AT 1033,3;" ";AT 1034,3;" ";AT 1035,3;" ";AT 1036,3;" ";AT 1037,3;" ";AT 1038,3;" ";AT 1039,3;" ";AT 1040,3;" ";AT 1041,3;" ";AT 1042,3;" ";AT 1043,3;" ";AT 1044,3;" ";AT 1045,3;" ";AT 1046,3;" ";AT 1047,3;" ";AT 1048,3;" ";AT 1049,3;" ";AT 1050,3;" ";AT 1051,3;" ";AT 1052,3;" ";AT 1053,3;" ";AT 1054,3;" ";AT 1055,3;" ";AT 1056,3;" ";AT 1057,3;" ";AT 1058,3;" ";AT 1059,3;" ";AT 1060,3;" ";AT 1061,3;" ";AT 1062,3;" ";AT 1063,3;" ";AT 1064,3;" ";AT 1065,3;" ";AT 1066,3;" ";AT 1067,3;" ";AT 1068,3;" ";AT 1069,3;" ";AT 1070,3;" ";AT 1071,3;" ";AT 1072,3;" ";AT 1073,3;" ";AT 1074,3;" ";AT 1075,3;" ";AT 1076,3;" ";AT 1077,3;" ";AT 1078,3;" ";AT 1079,3;" ";AT 1080,3;" ";AT 1081,3;" ";AT 1082,3;" ";AT 1083,3;" ";AT 1084,3;" ";AT 1085,3;" ";AT 1086,3;" ";AT 1087,3;" ";AT 1088,3;" ";AT 1089,3;" ";AT 1090,3;" ";AT 1091,3;" ";AT 1092,3;" ";AT 1093,3;" ";AT 1094,3;" ";AT 1095,3;" ";AT 1096,3;" ";AT 1097,3;" ";AT 1098,3;" ";AT 1099,3;" ";AT 1100,3;" ";AT 1101,3;" ";AT 1102,3;" ";AT 1103,3;" ";AT 1104,3;" ";AT 1105,3;" ";AT 1106,3;" ";AT 1107,3;" ";AT 1108,3;" ";AT 1109,3;" ";AT 1110,3;" ";AT 1111,3;" ";AT 1112,3;" ";AT 1113,3;" ";AT 1114,3;" ";AT 1115,3;" ";AT 1116,3;" ";AT 1117,3;" ";AT 1118,3;" ";AT 1119,3;" ";AT 1120,3;" ";AT 1121,3;" ";AT 1122,3;" ";AT 1123,3;" ";AT 1124,3;" ";AT 1125,3;" ";AT 1126,3;" ";AT 1127,3;" ";AT 1128,3;" ";AT 1129,3;" ";AT 1130,3;" ";AT 1131,3;" ";AT 1132,3;" ";AT 1133,3;" ";AT 1134,3;" ";AT 1135,3;" ";AT 1136,3;" ";AT 1137,3;" ";AT 1138,3;" ";AT 1139,3;" ";AT 1140,3;" ";AT 1141,3;" ";AT 1142,3;" ";AT 1143,3;" ";AT 1144,3;" ";AT 1145,3;" ";AT 1146,3;" ";AT 1147,3;" ";AT 1148,3;" ";AT 1149,3;" ";AT 1150,3;" ";AT 1151,3;" ";AT 1152,3;" ";AT 1153,3;" ";AT 1154,3;" ";AT 1155,3;" ";AT 1156,3;" ";AT 1157,3;" ";AT 1158,3;" ";AT 1159,3;" ";AT 1160,3;" ";AT 1161,3;" ";AT 1162,3;" ";AT 1163,3;" ";AT 1164,3;" ";AT 1165,3;" ";AT 1166,3;" ";AT 1167,3;" ";AT 1168,3;" ";AT 1169,3;" ";AT 1170,3;" ";AT 1171,3;" ";AT 1172,3;" ";AT 1173,3;" ";AT 1174,3;" ";AT 1175,3;" ";AT 1176,3;" ";AT 1177,3;" ";AT 1178,3;" ";AT 1179,3;" ";AT 1180,3;" ";AT 1181,3;" ";AT 1182,3;" ";AT 1183,3;" ";AT 1184,3;" ";AT
```



Das Programm hat als Vorteil - echter INPUT AT in einer Maske und einfache Cursorbewegungen im Feld vorgegebener Länge (LEN = 9); Nachteil - kein volles Editing, INSERT + DELETE fehlen noch. Zur Erläuterung kurz die Variablen der Routine:

Z - Startzeile ) für LEN - Länge der Eingabe  
S - Startspalte ) Werteanzeige IS - Inputstring  
P - Zeichenzähler X - Inputzeichencode

Nun aber zur vollständigen INPUT-Simulation mit allen Editing-Funktionen; wir nennen die Routine zur Unterscheidung "inscreen-INPUT".

inscreen-INPUT

Was muß die Lösung bringen? Welche Funktionen sind zu liefern? Fassen wir sie kurz zusammen, ehe wir das Programm erläutern.

- > Anzeige der Eingabestelle mittels eine Punktmarkierung (Länge des erwarteten Inputs)
- > Flashing Cursor unter Kontrolle des Benutzers (leicht zu realisieren mit der SCREENS-Funktion)
- > volles insert und delete, solange die Eingabe nicht abgeschlossen ist

```

10 REM "inscreen-INPUT"
12 LET X=16
13 LET Y=11
14 LET LA=6
15 LET XN=X+LA-1
16 PRINT AT Y,X;"....." ( T
17 LET FLASH=0
18 LET INSERT=40
19 LET DELETE=50
20 GOSUB FLASH
21 PAUSE 0
22 IF IS=INKEY$
23 IF CODE IS=13 THEN STOP
24 IF IS=" " THEN GO SUB INSE
25 LET X=X+(X<XN AND CODE IS=9)
26 LET X=X-(X>X0 AND CODE IS=8)
27 IF CODE IS=12 THEN GO SUB D
28 GO TO 20
29
30 IF X>X0 THEN PRINT AT Y,X-1
31 SCREENS (Y,X-1)
32 PRINT AT Y,X; FLASH 1; SCREE
33 N$ (Y,X); FLASH 0; SCREENS (
34 Y,X+1)
35 RETURN
36 FOR I=XN-1 TO X STEP -1
37 PRINT AT Y,I+1; SCREENS (Y,
38 I)
39 NEXT I
40 PRINT AT Y,X; IS
41 LET X=X+(X<XN)
42 RETURN
43 FOR I=X TO XN-1
44 PRINT AT Y,I; SCREENS (Y,I+
45 1)
46 NEXT I
47 PRINT AT Y,XN;"."
48 LET IS=CHR$ 0
49 RETURN

```

Nun zur Lösung. Gehen wir kurz auf ein paar Besonderheiten ein, geben wir stichwortartig Hilfestellung zum Lesen und für eigene INPUT-Arbeit.

- Die Längenauswahl (s. Zeile 16) erfolgt anhand eines vorgegebenen Strings, der >= der vorgegebenen LA sein muß.
- Die SCREENS-Funktion (Zeile 30) printet das Zeichen so, wie es vorliegt, vor die aktuelle Stelle; dann wird das aktuelle Zeichen angedruckt, Flash an- und ausgeschaltet und das Folgezeichen geprintet. Sinn: Synchronisation der Positionierung und damit richtige Anzeige nach Delete/Insert.
- Bei Insert (Zeile 40) druckt die Schleife rückwärts - bis zur aktuellen Stelle; dann kommt das neue Zeichen, und der Spaltenzähler wird erhöht. Da ein Zeichen hinzukommt, startet der Ausdruck vor der letzten bisherigen Datenstelle (XN - 1); gezeigt wird diese jedoch an "alter" Position (I + 1). Damit ist Platz für das nächste neue Zeichen geschaffen.
- Delete läuft ähnlich. Da ein Zeichen wegfällt, zählt die Schleife (Zeile 50) bis zur vorletzten Datenstelle (XN - 1). Jedes Folgezeichen wird von rechts her vorpositioniert (I + 1). IS wird vorgesetzt; der Punkt am Ende signalisiert, daß ein Zeichen frei geworden ist. (Merke: Bei Code=12 enthält IS tatsächlich dieses Zeichen, die Spalte muß folglich um 1 reduziert werden, damit die Spaltenanzeige wieder stimmt)
- Eine Überlaufschutzung gewährleistet, daß linker und rechter "Rand" nicht überschritten werden (X<XN bzw. X>X0).

Nachzutragen sind noch die verwendeten Variablen und ihre Bedeutung:

LA = Länge Input X0 = Start Anzeigebereich (Spalte)  
Y = Druckspalte ) aktuell XN = Ende Anzeigebereich (Spalte)  
X = Druckzeile ) I = Zählvariable

Warum so viel Aufwand, wenn es auch schnell und kurz geht - so mag man fragen? Lösungen in BASIC sind oft nicht einfach; besser geht es in Maschinencode. SINCLAIR-Fans, die nur schnelle Hilfe haben wollen, sollen abschließend auch bedient werden. Für sie haben wir ein kurzes MC-Programm parat (Dank an "Computer Kontakt" 8/85). Dazu ein kurzes Wort.

Zur Veranschaulichung wird zuerst der Screen mit "Balken" voll gemalt. Man kann damit besser sehen, wie MC optisch besser als das INPUT AT arbeitet. Ersteres zeigt mit einem Stern, wo der Input hinkommt. Bemerkenswert ist, daß der MC direkt im Input-Statement aufgerufen wird. Die Vorpositionierung wird man mit AT oder TAB vornehmen.

```

10 REM "ERSATZLOESUNG" (STAT
11 MC)
12 FOR N=0 TO 31
13 PRINT AT N,M;" ";
14 NEXT N
15 INPUT AT 22,0;AT 0,0;"?";X$
16 IF X$<>" " THEN GO TO 14

```

Genug der einfachen Input's, gehen wir zu Texten mit  
ausgeprägter Formatierung über.

## Text + Formatierung

Die wichtigste Frage bei Texten (weitere Überlegungen dazu im Rahmen von Programm TEXTUER) ist

wie ist Text zu organisieren und zu speichern und  
wie ist er in eine formatierte Ausgabeform zu bringen ?

Kleinere Texte wird man in einem String behandeln, wemngleich eine Riesenzeichenkette auch Grenzen hat (beim 48 K Spectrum ca. 14 K). Die kann beliebig verkürzt, verlängert und umgruppiert werden; allerdings sinkt mit zunehmender Länge die Verarbeitungsgeschwindigkeit sichtbar.

Interessant bei Wahl der "single string"-Form ist die Handhabung von Wörtern. Texte zu formatieren, heißt sich mit dem Problem "Spacing" auseinander zu setzen. Was ist daran zu verstehen? Wer formatiert, muß Wortzwischenräume berücksichtigen. Jede druckgerechte Form eines Textes muß zumindest sicherzustellen, daß es nicht an Rand zu Wörtern kommt (engl. word wrapping). Die Spaces zwischen Worten müssen sich vermeiden - und damit die Programmierung vereinfachen -, wenn man die Wörtern von vornherein hätte. Erster Gedanke fürs Texthandling - statt Spaces mitzuschleifen, werden an deren Stelle Wörtern (als Codes) im Text gespeichert. Das kostet kein Byte zusätzlich und erleichtert Ausgabeoperationen erheblich, von Dingen wie Wortsursor u.ä. ganz zu schweigen. Allerdings hat die Idee einen Nachteil - Längen- oder Codierung kostet Zeit...

Die folgende Routine TEXTILEN ist nicht ohne Pfiff. Sie unterscheidet sich vom sonst Gebotenen und soll zu weiterer

"Textarbeit" anregen. - Was geschieht im Programm ?

Zuerst wird die Eingabe IS in Zwischenstring AS gebracht; Entscheidungskriterium ist das englische Pfundzeichen. Kommt ein Leerzeichen (= Wortzwischenraum), tritt an seine Stelle die Wortlänge (Variable N) als Code. Danach wandert das Wort in Textdateistring IS, an den es "angeklebt" wird. Zur erneuten WortZählung ist AS zu leeren und N auf Null zu setzen. So entsteht in IS abgelegt die Folge

Länge W o r t Länge W o r t ...

Der Nachteil, daß TS nicht mehr direkt gedruckt werden kann ist aufzuwiegen durch die leichtere Formatierung (2. Programmteil ab Zeile 46). Bei vorgegebener Breite (Variable BR) ist die Bildschirmformat 32 Zeichen/Zeile) wird einfach von Längenfeld zu Längenfeld gesprungen. Liegt die Breite im Rahmen von BR, werden alle Zeichen bis zur "Lücke" gedruckt und ein Space (Zeichen 32) ist jedoch ein Space-Übergang erreicht, erfolgt Zellenvorschub um die Länge von N. Dann wird weitergezogen, erfolgt abgekehrt wird. Am Ende ist TS leer. Möchte man ihn erhalten, ist vorab zu sichern (z.B. mit LET AS = TS).

```

9 REM 15-11-1967
10 REM -----
11 gen INPUT Text mit Wortlaen
12
13
14 PRINT " 10.1.1968 (Ende
15 mit 2)
16 LET t$=""
17 LET a$=""
18 LET n=1
19 PAUSE 0.5
20 LET i$=INKEY
21 IF i$="0" OR CODE i$=12
22 THEN GO TO 30
23 IF i$="2" THEN GO SUB 40
24 IF i$="4" THEN GO SUB 40
25 PRINT i$ " THEN LET a$=a$+i$
26
27 LET n=n+1
28 GO TO 30
29
30 LET a$=CHR$ (n-1)+a$
31 LET a$=a$+a$
32 LET n=0
33 RETURN
34 PRINT " 10.1.1968"
35
36 REM BREITE frei waelhbar
37 LET br=32
38
39 LET n=1+n+CODE t$(1)
40 IF n<br THEN PRINT t$(2 TO 1+CODE t$(1)); "
41 GO TO 34
42
43 PRINT
44 GO TO 48
45
46 IF t$="(2+CODE t$(1) TO )
47 THEN PRINT
48
49
50

```

Wer den Vorteil des direkten Stringdrucks (von IS) behalten möchte, wird die folgende Routine für 'Flattersatz' zu schätzen wissen. Sie arbeitet ohne "Längen" und gestattet den direkten Ausdruck ohne eigene Formatierung. String IS ist nämlich bereits vorformatiert. Wie geht das?

Der ganze Trick besteht darin, den Text durchzuzählen und nach dem letzten Wort innerhalb der Zeilenbreite das Leerzeichen durch CR = Zeilenvorschub (Code 13) zu ersetzen. Im Einzelnen:

Die FOR-NEXT-Schleife geht IS durch, bis Breite BR erreicht wird. Ist das letzte Zeichen kein Space, wird solange zurückgezählt, bis ein "wordspace" gefunden ist. Der wird mit CR (Carriage Return) ersetzt. Dann wird von vorn "weitergerudert". Anschließend kann der Text mit PRINT formatiert ausgegeben werden. Einzige Voraussetzung für Anwendung der "CR-FORMATIERUNG":

BR = Druckbreite ) ge-  
IS = Textstring ) setzt  
N = Zählvariable frei

```

DIE EINFACHSTE FORMATIERUNG
DRUCKT DEN TEXT LINKSBUENDIG
UND BEHÄLT DIE WORTLÄNGEN
WIRD DAFÜR GESORGT, DASS KEINE
WORTSPREIZUNGEN (ENGL. WORD
WRAPPING) VORKOMMEN. DIE ROUTINE
PRÜFT DAS AB UND FÜGT AN
PASSENDER STELLE EINEN
ZEILENVORSCHUB (CHR$(13))
VOR JEDEM WORT. SO KANN BELIEBIG
(UND SCHNELL!) OFT GEPRINTET
WERDEN...

```

```

1000 REM "FLATTERSATZ"
1002 INPUT "DRUCKBREITE"; BR
1010 LET BR=BR+1
FOR N=BR TO LEN IS STEP BR
1020 GO TO 1020+(IS(N)=" ")
1021 NEXT N
PRINT IS

```

Die Vorteile dieser wohl kürzesten "Flattersatz"-Routine sind beträchtlich - Datenstring IS bleibt erhalten, formatierte Ausgabe jederzeit und blitzschnell. Normalpapierdrucker kann direkt (mit LPRINT) angesteuert werden, wenig Programmaufwand.

#### Blocksatz

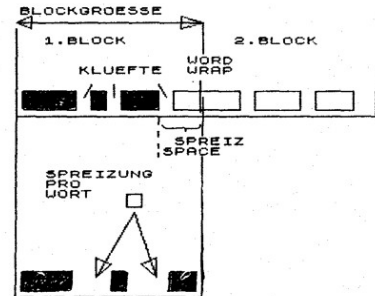
Das sauberste Druckbild liefert der Blocksatz; darunter ist die sowohl links- wie rechtsbündige Textwiedergabe zu verstehen. Blocksatz krönt gewissermaßen die Druckausgabe für den Hausgebrauch. Wer schnellere und noch bessere Formatierung haben will, wird sich in der Regel an kommerzielle Produkte (in Maschinencode) wenden. Was ist von einer BASIC-Routine BLOCKSATZ zu verlangen? Kurzum ein bündiger Textdruck in gegebener Breite und ab vorgewählter Spalte ohne Wortbrüche. - Die Programmanforderungen lassen sich in drei Punkten zusammenfassen:

1. eine Word Wrap-Lösung (wie oben)
2. zusätzliche Spreizung des Zeilentextes
3. einfache Benutzung

Das Ergebnis - Programm BLOCKSATZ - hat es denn auch in sich; schildern wir in der gebotenen Kürze, wie es im Wesentlichen funktioniert.

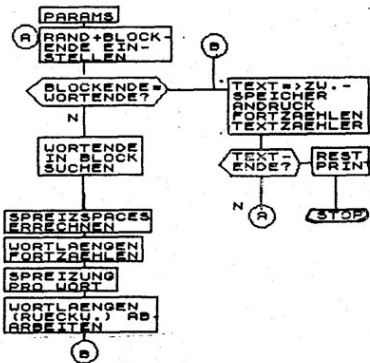
Gegeben ist zunächst ein Textstring IS; Druckbreite BR und Druckbeginn (= 1. Spalte) DB werden vom User gesetzt. Zuerst wird Zwischenstring AS mit der echten Breite dimensioniert; AS nimmt in Verlauf der ersten FOR-NEXT-Schleife die Textteilstücke auf. Da die Logik ziemlich verzwickelt ist, sei erst die Mechanik der "Spreizung" dargestellt. Dazu statt vieler Worte das folgende Schema.

#### WORTSPREIZUNG (SCHEMATISCH)



Ergibt die Prüfung auf "Trennlinie", daß ein "sauberer" Block vorliegt, wird AS gefüllt (inc. führender Spaces für leer-Spalten vorn) und gedruckt. Andernfalls erhält IS die Wortlängen (als Codes) zur späteren Spreizung. Da die Restzeichen, die zum vollen Block fehlen, ebenfalls ermittelt werden, kann die Spacezahl pro Wort (zwecks Einfügen) errechnet werden. Erst nach alledem wird IS mit seinen Längencodes rückwärts abgearbeitet und vor jedes Wort die Pro-Wort-Spaces eingefügt. - Klingt sehr kompliziert? Ist es auch... Um Hilfestellung zu geben, die Groblogik für diejenigen, die sich weiter ins Programm vertiefen wollen.

# PROLOGIK "BLOCKSATZ"



Nun aber genug des grausamen Spiels. Bringen wir abschließend die Variablen von BLOCKSATZ, das Programm selber und einige Screen-Copies, die zeigen, wie alles in Praxis funktioniert.

Variablen: TS = Text I = Start d. Daten in AS  
 BR = Druckbreite ('Block') E = Ende o. Textblocks  
 DB = Spalte Druckbeginn  
 DIM AS(BR-1+DB) = dimens.  
 Zwischenstring echter beginne  
 Breite N = Zahlvariable  
 IS = Userinput + Wortzähler

```

10 REM BLOCKSATZ
11
12
13
14 REM TEXTSTRING VORH. MIT TS;
15 WORTLAENGEN WERDEN MIT IS A
16 USGERICHTET
17
18 INPUT "DRUCKBREITE? (RETURN=
19 IF IS<>"" THEN LET BR=VAL I
20
21 INPUT "LINKER RAND (DRUCKBEG
22 INN: RETURN=1.ST.)"; IS
23 LET DB=1
24 IF IS<>"" THEN LET DB=VAL I
25
26 DIM AS(BR-1+DB)
27 REM AS T A R T *****
  
```

```

28 FOR N=1 TO LEN TS-BR
29 LET T=DB
30 LET EN+BR-1
31 IF TS(E+1)=? THEN LET AS
32 (T TO )=TS(N TO E)
33 PRINT AS
34 LET N=N+1
35 GO TO 28
36 IF TS(E)<>"" THEN LET E=E
37
38 GO TO 28
39 LET N=N+BR-E
40 LET N=CHR$(N)
41 FOR E=E TO 0 STEP -1
42 IF TS(E)<>"" THEN LET I=
43 I+CHR$(N)
44 GO TO 34
45 LET I=(LEN IS)=CHR$(CODE
46 IS(LEN IS)+1)
47 NEXT I
48 LET S=S/(LEN IS)-(LEN IS)+1
49
50 FOR E=LEN IS TO 1 STEP -1
51 LET ST=CODE IS(E)+1
52 LET AS(T TO )=TS(N TO N+ST
53 LET T=T+ST+5
54 LET N=N+ST
55 NEXT N
56 PRINT AS
57 LET N=N-1
58 NEXT N
59 LET T=DB
60 LET T TO )=TS(N TO )
61 PRINT AS
62 REM *****
63 ***** E N D E
  
```

## TEST FÜR FORMATTIERUNG

Dies ist ein Testtext. Er soll zeigen, wie die Blocksatz-Routine arbeitet. Zur Trennung dient lediglich das Leerzeichen, man kann weitere Separatoren (Komma, Punkt etc.) einführen. Da die Laengen jeweils in einem Hilfsstring festgehalten werden, ist das Ganze recht langsam. In BASIC kann man eben bei so komplizierten Sachen keine Wunder erwarten. So schneller wird das Ganze, wenn die Wortlaengen extra gespeichert sind, oder im Text selber bereits stehen.

## TEST FÜR BLOCKSATZ (2. Version)

Dies ist ein Testtext. Er soll zeigen, wie die Blocksatz-Routine arbeitet. Zur Trennung dient lediglich das Leerzeichen, man kann weitere Separatoren (Komma, Punkt etc.) einführen. Da die Laengen jeweils in einem Hilfsstring festgehalten werden, ist das Ganze recht langsam. In BASIC kann man eben bei so komplizierten Sachen keine Wunder erwarten. So schneller wird das Ganze, wenn die Wortlaengen extra gespeichert sind, oder im Text selber bereits stehen.

## Von der Zielsetzung zum Programm

## Vorüberlegungen

Der folgende Abschnitt verfolgt zwei Absichten

1. soll gezeigt werden, wie man konkret ein Programm entwickelt
2. soll die Verarbeitung von Texten programmtechnisch veranschaulicht werden.

Da bekanntlich nichts so aufschlussreich ist wie das praktische Beispiel, wollen wir ausgehend von der Idee den Weg zur BASIC-Lösung verfolgen. Betrachtungsgegenstand sind Textdaten. Verwillen wir zunächst bei letzterem, ehe wir ein paar Regeln der Programmentwicklung vorführen.

Texte stellen nicht-numerischen Daten dar, welche sowohl im Ganzen wie auch in Teilen (= Worte) variabel sind. Es handelt sich also um Strings, d.h. reine Zeichenketten. Zwei Punkte tauchen in dem Zusammenhang auf

#### Datenorganisation und Codespeicherung

Was konkret heißt: a) Wie sollen Texte abgelegt werden - als einfacher String oder als Stringarray? b) Welche Codes kommen vor und wie kann man den Speicherplatz optimieren? Gehen wir zuerst auf letzteres ein. - Bei Text sind in der Regel Zeichen des ASCII-Codes angesprochen. Besonderheiten wie deutsche Sonderzeichen werden meist auf vorhandene druckbare Zeichen umgelegt, je nachdem, welche Codes der Drucker verlangt. Ansonsten sind Codes bis 127 angesprochen, also die 7-Bit-Kombinationen, welchen Druckzeichen zugeordnet sind. Das Alphabethat hat nur 26 Zeichen; rechnet man die Kleinbuchstaben hinzu, ergibt sich die doppelte Menge. Dazu kommen noch ein paar notwendige Sonderzeichen. Worauf läuft das hinaus?

Normalerweise wird man pro Zeichen ein Byte benutzen, d.h. eine Kombination von 8 Bits. Benötigt wird aber nur maximal die Hälfte der Kombinationen, nämlich weniger als 127 Zeichen. Realisiert man die Großbuchstaben per Umschalter (= ein ansonsten unbenutztes Sonderzeichen), kommt man gar mit einem Viertel der möglichen Zeichen aus - ca. 30. Man könnte folglich geneigt sein, die nächste 2er Potenz anzusteuern. Dann blieben eine Reihe Bits frei, sodaß man u.U. zwei Buchstaben in einem Byte unterbringen könnte. Die nächste Bitkombination, welche das gesamte Alphabet abdeckt, ist  $32 = 2 \text{ hoch } 5$ , also 5 Bits von 8 verfügbaren eines Bytes. Leider lassen sich so noch keine zwei Buchstaben "verpacken", wohl jedoch, wenn man auf  $2 \text{ mal } 16 (= 4 \text{ Bits})$  ginge. Dazu müßte man ein Steuerbyte vor nachfolgende 8 Alphazeichen legen. Es würde angehen, ob jeweils die erste oder die zweite Hälfte der 32 Möglichkeiten der Bitkombination besetzt

Dies ist ein Testtext. Er soll zeigen, wie die Blocksatz-Routine arbeitet. Zur Trennung dient lediglich das Leerzeichen, man kann weitere Separatoren (Komma, Punkt etc.) einführen. Da die Laengen jeweils in einem Hilfsstring festgehalten werden, ist das Ganze recht langsam. In BASIC kann man eben bei so komplizierten Sachen keine Wunder erwarten. Schneller wird das Ganze, wenn die Wortlaengen extra gespeichert sind, oder im Text selber bereits stehen.

Dies ist ein Testtext. Er soll zeigen, wie die Blocksatz-Routine arbeitet. Zur Trennung dient lediglich das Leerzeichen, man kann weitere Separatoren (Komma, Punkt etc.) einführen. Da die Laengen jeweils in einem Hilfsstring festgehalten werden, ist das Ganze recht langsam. In BASIC kann man eben bei so komplizierten Sachen keine Wunder erwarten. Schneller wird das Ganze, wenn die Wortlaengen extra gespeichert sind, oder im Text selber bereits stehen.

ist... Soviel nur als Anregung zur Datenkompression für MC-Tüftler; in BASIC ist Derartiges ein Unding, weshalb die Idee hier nicht weiterverfolgt sei.

Nun zur Datenorganisation, ehe wir die Programmentwicklung von TEXTIVER schildern. - Verfolgt man die gütige Maxime - ein Zeichen - ein Byte - weiter, stellt sich die Frage: Wie soll Text verarbeitet werden, als ein String oder als Stringarray? Der SINCLAIR läßt im Gegensatz zu anderen Heimcomputern die Anlage unbegrenzter Strings zu. Man kann sich die Arbeit einfach machen, indem bei Text lediglich ein Riesenstring verwaltet wird; was im SINCLAIR-BASIC recht simpel durch "Slicing" zu bewerkstelligen ist. Aber Vorsicht!

Wer es mal probiert hat, Strings "beliebiger" Länge anzulegen - am anschaulichsten mittels einer einfachen Schleife, in der ein String immer um ein Stück verlängert wird - wird zu seinem Erstaunen feststellen, daß es da eine Höchstgrenze gibt. Sie liegt bei rund 14.000 Bytes, dann geht nichts mehr. Und noch schlimmer; praktiziert man eifriges Umbesetzen ("Slicing"), ist weit darunter Schluss. Wie kommt das? Auch das "Betriebssystem" im ROM kann nicht zaubern: Intern werden auf Maschinenebene "verschobene" Stücke zwischengespeichert. Die Ausführungskontrolle fordert zusätzlichen Speicherplatz an, es kommt zum "out of memory" ... Was nahelegt, mit der Array-Lösung zu arbeiten. Die hat noch zwei weitere Vorteile: 1. ist ein Stringlicing genau so gut möglich (wie TEXTIVER zeigt), 2. bekommt man keine Schwierigkeiten beim Abspeichern (Das MicroDrive fordert strikte Dimensionierung, der TonbandSAVE ist da nicht so penibel).

Also flugs einen Textarray mit sagen wir 50 Bildschirmseiten dimensioniert und weiter geht es mit der Verarbeitung; 35.000 Zeichen lassen sich schön für Texte ausnutzen, richtig? Weit gefehlt. Das zuvor Gesagte gilt eingeschränkt auch für Stringarrays, vor allem, wenn man INTEXT-Operationen vollführen will. Man bleibe also tunlichst in eingeschränktem Rahmen von beispielsweise 15 Textseiten. Damit läßt man dem Rechner noch genug Freiraum, alle erforderlichen Manipulationen zu vollziehen.

#### TOP DOWN Vorgehensweise

Nach derlei Vorüberlegungen wenden wir uns wieder unserem Ziel einer Textverarbeitung zu. Ohne uns zusehr in Methodik zu verlieren, sei die Vorgehensweise kurz wiedergegeben. Eine moderne Art und Weise der Programmentwicklung ist unter dem Schlagwort TOP to DOWN bekannt. Was nichts anderes heißt als - man hänge sich von Allgemeinen zum Speziellen immer weiter an die realisierbare Lösung heran. Zuerst kommen ein paar Grundüberlegungen, dann Zielvorgaben, dann die programmtechnischen "Headlines" bis schließlich am Ende ablauffähige Module stehen. Dabei ist, was das Programm anbelangt, die Aufspaltung in Steuerung und Ausführung und die

Realisierung in abgegrenzten, überschaubaren "Paketen" gütige Praxis. Das soll an Prinzipien vorerst reichen..

Was muß ein Textverarbeitungsprogramm alles bringen? Versuchen wir die wichtigsten (bescheidenen) Features zu formulieren.. Zuerst die schnelle Aufnahme, Änderung und Handhabung des eingegebenen Textes; der Text muß sogleich auf dem Bildschirm erscheinen, die aktuelle Textstelle sei durch Cursor sichtbar. Alle Bewegungen sind unter Cursorcontrol vorzunehmen, sodaß der Benutzer immer sieht, was geschieht. Weiter ist unentbehrlich das Einfügen von Zeichen an beliebiger Stelle und in beliebiger Länge; durch zeichenweises "Delete" ergänzt. Eine Copyfunktion zum nachträglichen Redigieren möge hinzu treten, ebenso zeilenweises Löschen und "Platzschaffen" (= Einfügen von Leerzeilen). Was fehlt noch? Natürlich Dinge wie SAVE und LOAD, möglichst auch von Textteilen (z.B. den Text ab Cursorposition). Damit die Textarbeit auch sichtbaren Ausdruck findet, tritt ein formatierter Druckerprint hinzu. Weitere "Extras" machen ein Textverarbeitungsprogramm noch "rund": Blättern, d.h. schnelles seitenweises Sichten des Gesamttextes; ansprechendes Menü mit Aufzeigen aller Funktionen; allgemein - ein einfaches Handling ohne viel Umschalterei und schlecht merkbare Kürzel..

Nachdem wir die Erfordernisse (Fachterminus "requirements") und damit das Ziel formuliert haben, ist der "technische" Teil zu überlegen. Wie hat die Lösung im Prinzip auszusehen?

Nachdem wir die Frage der Datenorganisation bereits abgekehrt haben, sei ein Hauptproblem bei Text und Datenhandling angesprochen:

#### die Synchronisation von Screen und Textarray

Würde man das "File" in 704-Zeichen-Blöcke aufteilen, wäre eine Bildschirmseitenanzeige einfach; nicht jedoch das Update. Da taucht nämlich regelmäßig die Frage des "Grenzübergangs" auf. Alle nachfolgenden Blöcke müßten bei Delete oder Insert eines einzigen Zeichens verschoben und neu ausgerichtet werden. Also ist ein Array ohne weitere Unterteilung angebracht.

Es bleibt noch genug Fieselei für die Synchronisation von Screen und Datenarray übrig:

- + SINCLAIR läßt die Spalten von 0 bis 31 (für 32 Zeichen/Zeile) laufen, die Subskripte jedoch kennen diesen Überlauf nicht;
- + Gleiches gilt für die 22 Zeilen pro Screen; die Dimensionierung schaut mit 704 Zeichen pro Seite anders aus
- + Schließlich sind Anfang und Ende beim Printen anders als im File zu behandeln: Dort hat das 1. Zeichen als Subskript 1, die Druckstelle arbeitet mit 0,0 (= Spalte 0, Zeile 0); letztes Zeichen des Files ist durch die Dimensionierung (festgehalten in Variable G wie Größe beispielsweise) vorgegeben; der Printüberlauf hingegen kennt nur die Festwerte 31 (Spalten) bzw. 21 (Zeilen)

Gerade beim Overflow-Check des Cursors wird uns die angesprochene differierende Zählweise noch zu schaffen machen.

Nun aber zum Programm und seiner Entstehung. Zeigen wir, wie alle aufgeführten Forderungen realisiert werden; zeigen wir vor allem, wie in Überlegten Schritten das "Werk" aufgebaut wurde..

Von der STEUERUNG zum Modul

Beim Programmieren von TEXTUER wurde auf Nachvollziehbarkeit großer Wert gelegt. Damit der SINCLAIR-Fan das Ergebnis leichter nachvollziehen kann, werden Unterprogramme ("Module") per indirekter Adressierung angesprungen. Statt

GOSUB 25

heißt es (LET CURSOR = 25) GOSUB CURSOR.

Das verlängert zwar die Ausführzeit, erleichtert jedoch den Überblick. Wenn es bei schnellem Tippen zum "Zeichenschluckauf" kommt, mag man die indirekten durch direkte Adressen ersetzen.

Fange wir mit dem Allgemeinen an, der Programmsteuerung. In der Regel ist sie identisch mit dem Hauptmenu. Es zeigt an, was der Benutzer alles wählen kann, ist also gleichzeitig Wunschliste fürs Programm.

Das Hauptmenu drückt alle Optionen auf den Screen und zwar ohne viele Print-Statements. Es wird noch ein hübscher Rahmen gezeichnet, um das Ganze optisch abzurunden. Anschließend wartet ein PAUSE-Stop auf Userreaktion, auf daß das "Bild" dann verschwindet.

```

99 REM
100 CLS
PRINT TAB 6;"[CURSOR]";
TAB 1;"1" TAB 2;"2" TAB 3;"3" TAB 4;"4" TAB 5;"5" TAB 6;"6" TAB 7;"7" TAB 8;"8" TAB 9;"9" TAB 10;"0" TAB 11;"1" TAB 12;"2" TAB 13;"3" TAB 14;"4" TAB 15;"5" TAB 16;"6" TAB 17;"7" TAB 18;"8" TAB 19;"9" TAB 20;"0" TAB 21;"1" TAB 22;"2" TAB 23;"3" TAB 24;"4" TAB 25;"5" TAB 26;"6" TAB 27;"7" TAB 28;"8" TAB 29;"9" TAB 30;"0" TAB 31;"1" TAB 32;"2" TAB 33;"3" TAB 34;"4" TAB 35;"5" TAB 36;"6" TAB 37;"7" TAB 38;"8" TAB 39;"9" TAB 40;"0" TAB 41;"1" TAB 42;"2" TAB 43;"3" TAB 44;"4" TAB 45;"5" TAB 46;"6" TAB 47;"7" TAB 48;"8" TAB 49;"9" TAB 50;"0" TAB 51;"1" TAB 52;"2" TAB 53;"3" TAB 54;"4" TAB 55;"5" TAB 56;"6" TAB 57;"7" TAB 58;"8" TAB 59;"9" TAB 60;"0" TAB 61;"1" TAB 62;"2" TAB 63;"3" TAB 64;"4" TAB 65;"5" TAB 66;"6" TAB 67;"7" TAB 68;"8" TAB 69;"9" TAB 70;"0" TAB 71;"1" TAB 72;"2" TAB 73;"3" TAB 74;"4" TAB 75;"5" TAB 76;"6" TAB 77;"7" TAB 78;"8" TAB 79;"9" TAB 80;"0" TAB 81;"1" TAB 82;"2" TAB 83;"3" TAB 84;"4" TAB 85;"5" TAB 86;"6" TAB 87;"7" TAB 88;"8" TAB 89;"9" TAB 90;"0" TAB 91;"1" TAB 92;"2" TAB 93;"3" TAB 94;"4" TAB 95;"5" TAB 96;"6" TAB 97;"7" TAB 98;"8" TAB 99;"9" TAB 100;"0" TAB 101;"1" TAB 102;"2" TAB 103;"3" TAB 104;"4" TAB 105;"5" TAB 106;"6" TAB 107;"7" TAB 108;"8" TAB 109;"9" TAB 110;"0" TAB 111;"1" TAB 112;"2" TAB 113;"3" TAB 114;"4" TAB 115;"5" TAB 116;"6" TAB 117;"7" TAB 118;"8" TAB 119;"9" TAB 120;"0" TAB 121;"1" TAB 122;"2" TAB 123;"3" TAB 124;"4" TAB 125;"5" TAB 126;"6" TAB 127;"7" TAB 128;"8" TAB 129;"9" TAB 130;"0" TAB 131;"1" TAB 132;"2" TAB 133;"3" TAB 134;"4" TAB 135;"5" TAB 136;"6" TAB 137;"7" TAB 138;"8" TAB 139;"9" TAB 140;"0" TAB 141;"1" TAB 142;"2" TAB 143;"3" TAB 144;"4" TAB 145;"5" TAB 146;"6" TAB 147;"7" TAB 148;"8" TAB 149;"9" TAB 150;"0" TAB 151;"1" TAB 152;"2" TAB 153;"3" TAB 154;"4" TAB 155;"5" TAB 156;"6" TAB 157;"7" TAB 158;"8" TAB 159;"9" TAB 160;"0" TAB 161;"1" TAB 162;"2" TAB 163;"3" TAB 164;"4" TAB 165;"5" TAB 166;"6" TAB 167;"7" TAB 168;"8" TAB 169;"9" TAB 170;"0" TAB 171;"1" TAB 172;"2" TAB 173;"3" TAB 174;"4" TAB 175;"5" TAB 176;"6" TAB 177;"7" TAB 178;"8" TAB 179;"9" TAB 180;"0" TAB 181;"1" TAB 182;"2" TAB 183;"3" TAB 184;"4" TAB 185;"5" TAB 186;"6" TAB 187;"7" TAB 188;"8" TAB 189;"9" TAB 190;"0" TAB 191;"1" TAB 192;"2" TAB 193;"3" TAB 194;"4" TAB 195;"5" TAB 196;"6" TAB 197;"7" TAB 198;"8" TAB 199;"9" TAB 200;"0" TAB 201;"1" TAB 202;"2" TAB 203;"3" TAB 204;"4" TAB 205;"5" TAB 206;"6" TAB 207;"7" TAB 208;"8" TAB 209;"9" TAB 210;"0" TAB 211;"1" TAB 212;"2" TAB 213;"3" TAB 214;"4" TAB 215;"5" TAB 216;"6" TAB 217;"7" TAB 218;"8" TAB 219;"9" TAB 220;"0" TAB 221;"1" TAB 222;"2" TAB 223;"3" TAB 224;"4" TAB 225;"5" TAB 226;"6" TAB 227;"7" TAB 228;"8" TAB 229;"9" TAB 230;"0" TAB 231;"1" TAB 232;"2" TAB 233;"3" TAB 234;"4" TAB 235;"5" TAB 236;"6" TAB 237;"7" TAB 238;"8" TAB 239;"9" TAB 240;"0" TAB 241;"1" TAB 242;"2" TAB 243;"3" TAB 244;"4" TAB 245;"5" TAB 246;"6" TAB 247;"7" TAB 248;"8" TAB 249;"9" TAB 250;"0" TAB 251;"1" TAB 252;"2" TAB 253;"3" TAB 254;"4" TAB 255;"5" TAB 256;"6" TAB 257;"7" TAB 258;"8" TAB 259;"9" TAB 260;"0" TAB 261;"1" TAB 262;"2" TAB 263;"3" TAB 264;"4" TAB 265;"5" TAB 266;"6" TAB 267;"7" TAB 268;"8" TAB 269;"9" TAB 270;"0" TAB 271;"1" TAB 272;"2" TAB 273;"3" TAB 274;"4" TAB 275;"5" TAB 276;"6" TAB 277;"7" TAB 278;"8" TAB 279;"9" TAB 280;"0" TAB 281;"1" TAB 282;"2" TAB 283;"3" TAB 284;"4" TAB 285;"5" TAB 286;"6" TAB 287;"7" TAB 288;"8" TAB 289;"9" TAB 290;"0" TAB 291;"1" TAB 292;"2" TAB 293;"3" TAB 294;"4" TAB 295;"5" TAB 296;"6" TAB 297;"7" TAB 298;"8" TAB 299;"9" TAB 300;"0" TAB 301;"1" TAB 302;"2" TAB 303;"3" TAB 304;"4" TAB 305;"5" TAB 306;"6" TAB 307;"7" TAB 308;"8" TAB 309;"9" TAB 310;"0" TAB 311;"1" TAB 312;"2" TAB 313;"3" TAB 314;"4" TAB 315;"5" TAB 316;"6" TAB 317;"7" TAB 318;"8" TAB 319;"9" TAB 320;"0" TAB 321;"1" TAB 322;"2" TAB 323;"3" TAB 324;"4" TAB 325;"5" TAB 326;"6" TAB 327;"7" TAB 328;"8" TAB 329;"9" TAB 330;"0" TAB 331;"1" TAB 332;"2" TAB 333;"3" TAB 334;"4" TAB 335;"5" TAB 336;"6" TAB 337;"7" TAB 338;"8" TAB 339;"9" TAB 340;"0" TAB 341;"1" TAB 342;"2" TAB 343;"3" TAB 344;"4" TAB 345;"5" TAB 346;"6" TAB 347;"7" TAB 348;"8" TAB 349;"9" TAB 350;"0" TAB 351;"1" TAB 352;"2" TAB 353;"3" TAB 354;"4" TAB 355;"5" TAB 356;"6" TAB 357;"7" TAB 358;"8" TAB 359;"9" TAB 360;"0" TAB 361;"1" TAB 362;"2" TAB 363;"3" TAB 364;"4" TAB 365;"5" TAB 366;"6" TAB 367;"7" TAB 368;"8" TAB 369;"9" TAB 370;"0" TAB 371;"1" TAB 372;"2" TAB 373;"3" TAB 374;"4" TAB 375;"5" TAB 376;"6" TAB 377;"7" TAB 378;"8" TAB 379;"9" TAB 380;"0" TAB 381;"1" TAB 382;"2" TAB 383;"3" TAB 384;"4" TAB 385;"5" TAB 386;"6" TAB 387;"7" TAB 388;"8" TAB 389;"9" TAB 390;"0" TAB 391;"1" TAB 392;"2" TAB 393;"3" TAB 394;"4" TAB 395;"5" TAB 396;"6" TAB 397;"7" TAB 398;"8" TAB 399;"9" TAB 400;"0" TAB 401;"1" TAB 402;"2" TAB 403;"3" TAB 404;"4" TAB 405;"5" TAB 406;"6" TAB 407;"7" TAB 408;"8" TAB 409;"9" TAB 410;"0" TAB 411;"1" TAB 412;"2" TAB 413;"3" TAB 414;"4" TAB 415;"5" TAB 416;"6" TAB 417;"7" TAB 418;"8" TAB 419;"9" TAB 420;"0" TAB 421;"1" TAB 422;"2" TAB 423;"3" TAB 424;"4" TAB 425;"5" TAB 426;"6" TAB 427;"7" TAB 428;"8" TAB 429;"9" TAB 430;"0" TAB 431;"1" TAB 432;"2" TAB 433;"3" TAB 434;"4" TAB 435;"5" TAB 436;"6" TAB 437;"7" TAB 438;"8" TAB 439;"9" TAB 440;"0" TAB 441;"1" TAB 442;"2" TAB 443;"3" TAB 444;"4" TAB 445;"5" TAB 446;"6" TAB 447;"7" TAB 448;"8" TAB 449;"9" TAB 450;"0" TAB 451;"1" TAB 452;"2" TAB 453;"3" TAB 454;"4" TAB 455;"5" TAB 456;"6" TAB 457;"7" TAB 458;"8" TAB 459;"9" TAB 460;"0" TAB 461;"1" TAB 462;"2" TAB 463;"3" TAB 464;"4" TAB 465;"5" TAB 466;"6" TAB 467;"7" TAB 468;"8" TAB 469;"9" TAB 470;"0" TAB 471;"1" TAB 472;"2" TAB 473;"3" TAB 474;"4" TAB 475;"5" TAB 476;"6" TAB 477;"7" TAB 478;"8" TAB 479;"9" TAB 480;"0" TAB 481;"1" TAB 482;"2" TAB 483;"3" TAB 484;"4" TAB 485;"5" TAB 486;"6" TAB 487;"7" TAB 488;"8" TAB 489;"9" TAB 490;"0" TAB 491;"1" TAB 492;"2" TAB 493;"3" TAB 494;"4" TAB 495;"5" TAB 496;"6" TAB 497;"7" TAB 498;"8" TAB 499;"9" TAB 500;"0" TAB 501;"1" TAB 502;"2" TAB 503;"3" TAB 504;"4" TAB 505;"5" TAB 506;"6" TAB 507;"7" TAB 508;"8" TAB 509;"9" TAB 510;"0" TAB 511;"1" TAB 512;"2" TAB 513;"3" TAB 514;"4" TAB 515;"5" TAB 516;"6" TAB 517;"7" TAB 518;"8" TAB 519;"9" TAB 520;"0" TAB 521;"1" TAB 522;"2" TAB 523;"3" TAB 524;"4" TAB 525;"5" TAB 526;"6" TAB 527;"7" TAB 528;"8" TAB 529;"9" TAB 530;"0" TAB 531;"1" TAB 532;"2" TAB 533;"3" TAB 534;"4" TAB 535;"5" TAB 536;"6" TAB 537;"7" TAB 538;"8" TAB 539;"9" TAB 540;"0" TAB 541;"1" TAB 542;"2" TAB 543;"3" TAB 544;"4" TAB 545;"5" TAB 546;"6" TAB 547;"7" TAB 548;"8" TAB 549;"9" TAB 550;"0" TAB 551;"1" TAB 552;"2" TAB 553;"3" TAB 554;"4" TAB 555;"5" TAB 556;"6" TAB 557;"7" TAB 558;"8" TAB 559;"9" TAB 560;"0" TAB 561;"1" TAB 562;"2" TAB 563;"3" TAB 564;"4" TAB 565;"5" TAB 566;"6" TAB 567;"7" TAB 568;"8" TAB 569;"9" TAB 570;"0" TAB 571;"1" TAB 572;"2" TAB 573;"3" TAB 574;"4" TAB 575;"5" TAB 576;"6" TAB 577;"7" TAB 578;"8" TAB 579;"9" TAB 580;"0" TAB 581;"1" TAB 582;"2" TAB 583;"3" TAB 584;"4" TAB 585;"5" TAB 586;"6" TAB 587;"7" TAB 588;"8" TAB 589;"9" TAB 590;"0" TAB 591;"1" TAB 592;"2" TAB 593;"3" TAB 594;"4" TAB 595;"5" TAB 596;"6" TAB 597;"7" TAB 598;"8" TAB 599;"9" TAB 600;"0" TAB 601;"1" TAB 602;"2" TAB 603;"3" TAB 604;"4" TAB 605;"5" TAB 606;"6" TAB 607;"7" TAB 608;"8" TAB 609;"9" TAB 610;"0" TAB 611;"1" TAB 612;"2" TAB 613;"3" TAB 614;"4" TAB 615;"5" TAB 616;"6" TAB 617;"7" TAB 618;"8" TAB 619;"9" TAB 620;"0" TAB 621;"1" TAB 622;"2" TAB 623;"3" TAB 624;"4" TAB 625;"5" TAB 626;"6" TAB 627;"7" TAB 628;"8" TAB 629;"9" TAB 630;"0" TAB 631;"1" TAB 632;"2" TAB 633;"3" TAB 634;"4" TAB 635;"5" TAB 636;"6" TAB 637;"7" TAB 638;"8" TAB 639;"9" TAB 640;"0" TAB 641;"1" TAB 642;"2" TAB 643;"3" TAB 644;"4" TAB 645;"5" TAB 646;"6" TAB 647;"7" TAB 648;"8" TAB 649;"9" TAB 650;"0" TAB 651;"1" TAB 652;"2" TAB 653;"3" TAB 654;"4" TAB 655;"5" TAB 656;"6" TAB 657;"7" TAB 658;"8" TAB 659;"9" TAB 660;"0" TAB 661;"1" TAB 662;"2" TAB 663;"3" TAB 664;"4" TAB 665;"5" TAB 666;"6" TAB 667;"7" TAB 668;"8" TAB 669;"9" TAB 670;"0" TAB 671;"1" TAB 672;"2" TAB 673;"3" TAB 674;"4" TAB 675;"5" TAB 676;"6" TAB 677;"7" TAB 678;"8" TAB 679;"9" TAB 680;"0" TAB 681;"1" TAB 682;"2" TAB 683;"3" TAB 684;"4" TAB 685;"5" TAB 686;"6" TAB 687;"7" TAB 688;"8" TAB 689;"9" TAB 690;"0" TAB 691;"1" TAB 692;"2" TAB 693;"3" TAB 694;"4" TAB 695;"5" TAB 696;"6" TAB 697;"7" TAB 698;"8" TAB 699;"9" TAB 700;"0" TAB 701;"1" TAB 702;"2" TAB 703;"3" TAB 704;"4" TAB 705;"5" TAB 706;"6" TAB 707;"7" TAB 708;"8" TAB 709;"9" TAB 710;"0" TAB 711;"1" TAB 712;"2" TAB 713;"3" TAB 714;"4" TAB 715;"5" TAB 716;"6" TAB 717;"7" TAB 718;"8" TAB 719;"9" TAB 720;"0" TAB 721;"1" TAB 722;"2" TAB 723;"3" TAB 724;"4" TAB 725;"5" TAB 726;"6" TAB 727;"7" TAB 728;"8" TAB 729;"9" TAB 730;"0" TAB 731;"1" TAB 732;"2" TAB 733;"3" TAB 734;"4" TAB 735;"5" TAB 736;"6" TAB 737;"7" TAB 738;"8" TAB 739;"9" TAB 740;"0" TAB 741;"1" TAB 742;"2" TAB 743;"3" TAB 744;"4" TAB 745;"5" TAB 746;"6" TAB 747;"7" TAB 748;"8" TAB 749;"9" TAB 750;"0" TAB 751;"1" TAB 752;"2" TAB 753;"3" TAB 754;"4" TAB 755;"5" TAB 756;"6" TAB 757;"7" TAB 758;"8" TAB 759;"9" TAB 760;"0" TAB 761;"1" TAB 762;"2" TAB 763;"3" TAB 764;"4" TAB 765;"5" TAB 766;"6" TAB 767;"7" TAB 768;"8" TAB 769;"9" TAB 770;"0" TAB 771;"1" TAB 772;"2" TAB 773;"3" TAB 774;"4" TAB 775;"5" TAB 776;"6" TAB 777;"7" TAB 778;"8" TAB 779;"9" TAB 780;"0" TAB 781;"1" TAB 782;"2" TAB 783;"3" TAB 784;"4" TAB 785;"5" TAB 786;"6" TAB 787;"7" TAB 788;"8" TAB 789;"9" TAB 790;"0" TAB 791;"1" TAB 792;"2" TAB 793;"3" TAB 794;"4" TAB 795;"5" TAB 796;"6" TAB 797;"7" TAB 798;"8" TAB 799;"9" TAB 800;"0" TAB 801;"1" TAB 802;"2" TAB 803;"3" TAB 804;"4" TAB 805;"5" TAB 806;"6" TAB 807;"7" TAB 808;"8" TAB 809;"9" TAB 810;"0" TAB 811;"1" TAB 812;"2" TAB 813;"3" TAB 814;"4" TAB 815;"5" TAB 816;"6" TAB 817;"7" TAB 818;"8" TAB 819;"9" TAB 820;"0" TAB 821;"1" TAB 822;"2" TAB 823;"3" TAB 824;"4" TAB 825;"5" TAB 826;"6" TAB 827;"7" TAB 828;"8" TAB 829;"9" TAB 830;"0" TAB 831;"1" TAB 832;"2" TAB 833;"3" TAB 834;"4" TAB 835;"5" TAB 836;"6" TAB 837;"7" TAB 838;"8" TAB 839;"9" TAB 840;"0" TAB 841;"1" TAB 842;"2" TAB 843;"3" TAB 844;"4" TAB 845;"5" TAB 846;"6" TAB 847;"7" TAB 848;"8" TAB 849;"9" TAB 850;"0" TAB 851;"1" TAB 852;"2" TAB 853;"3" TAB 854;"4" TAB 855;"5" TAB 856;"6" TAB 857;"7" TAB 858;"8" TAB 859;"9" TAB 860;"0" TAB 861;"1" TAB 862;"2" TAB 863;"3" TAB 864;"4" TAB 865;"5" TAB 866;"6" TAB 867;"7" TAB 868;"8" TAB 869;"9" TAB 870;"0" TAB 871;"1" TAB 872;"2" TAB 873;"3" TAB 874;"4" TAB 875;"5" TAB 876;"6" TAB 877;"7" TAB 878;"8" TAB 879;"9" TAB 880;"0" TAB 881;"1" TAB 882;"2" TAB 883;"3" TAB 884;"4" TAB 885;"5" TAB 886;"6" TAB 887;"7" TAB 888;"8" TAB 889;"9" TAB 890;"0" TAB 891;"1" TAB 892;"2" TAB 893;"3" TAB 894;"4" TAB 895;"5" TAB 896;"6" TAB 897;"7" TAB 898;"8" TAB 899;"9" TAB 900;"0" TAB 901;"1" TAB 902;"2" TAB 903;"3" TAB 904;"4" TAB 905;"5" TAB 906;"6" TAB 907;"7" TAB 908;"8" TAB 909;"9" TAB 910;"0" TAB 911;"1" TAB 912;"2" TAB 913;"3" TAB 914;"4" TAB 915;"5" TAB 916;"6" TAB 917;"7" TAB 918;"8" TAB 919;"9" TAB 920;"0" TAB 921;"1" TAB 922;"2" TAB 923;"3" TAB 924;"4" TAB 925;"5" TAB 926;"6" TAB 927;"7" TAB 928;"8" TAB 929;"9" TAB 930;"0" TAB 931;"1" TAB 932;"2" TAB 933;"3" TAB 934;"4" TAB 935;"5" TAB 936;"6" TAB 937;"7" TAB 938;"8" TAB 939;"9" TAB 940;"0" TAB 941;"1" TAB 942;"2" TAB 943;"3" TAB 944;"4" TAB 945;"5" TAB 946;"6" TAB 947;"7" TAB 948;"8" TAB 949;"9" TAB 950;"0" TAB 951;"1" TAB 952;"2" TAB 953;"3" TAB 954;"4" TAB 955;"5" TAB 956;"6" TAB 957;"7" TAB 958;"8" TAB 959;"9" TAB 960;"0" TAB 961;"1" TAB 962;"2" TAB 963;"3" TAB 964;"4" TAB 965;"5" TAB 966;"6" TAB 967;"7" TAB 968;"8" TAB 969;"9" TAB 970;"0" TAB 971;"1" TAB 972;"2" TAB 973;"3" TAB 974;"4" TAB 975;"5" TAB 976;"6" TAB 977;"7" TAB 978;"8" TAB 979;"9" TAB 980;"0" TAB 981;"1" TAB 982;"2" TAB 983;"3" TAB 984;"4" TAB 985;"5" TAB 986;"6" TAB 987;"7" TAB 988;"8" TAB 989;"9" TAB 990;"0" TAB 991;"1" TAB 992;"2" TAB 993;"3" TAB 994;"4" TAB 995;"5" TAB 996;"6" TAB 997;"7" TAB 998;"8" TAB 999;"9" TAB 1000;"0"

```

98

Menue wie auch die folgende Steuerung legen wir in die zukünftige Programm-Mitte, um davor Platz für schnellere Unterroutinen zu haben; hernach kommen langsamere Module, die seltener aufgerufen werden.

Bei Betrachtungen der Modulangebote fällt auf, daß mit zwei Ebenen gearbeitet wird. Der EDITING-Status ist zur Zeicheneingabe und für rasche Aktionen da. Alles kann durch bestehende Tastenbelegung erledigt werden - ohne "Umschalten". Anders beim HANDLING; darunter fallen Eingaben, welche seltener und vor allem gravierender sind. Mit dem "Umschalter" OR (Symbol Shift U) gelangt man etwa ins "Blättern"; Umschalten ist nötig, weil die gleichen Cursortasten benutzt werden. Auch "Kopieren", SAVE und LOAD etc. sind nur per HANDLING-Status erreichbar.

Wie sieht die eigentliche Steuerung aus? Was soll sie erledigen? Machen wir uns an eine kurze Beschreibung, Genaues ist in einem eigenen Schaubild aufgezeigt.

```

100 GO SUB SEITENANDRUCK
101 GO SUB ZEICHEN
102 IF Z=197 THEN GO SUB HANDLI
NG
GO TO 100
103 IF Z=35 THEN GO SUB INSERT
GO TO 100
104 IF Z=195 THEN GO SUB ZEILEN
LOESCH
GO TO 100
105 IF Z=31 AND Z<125 THEN GO S
UB ZEICHENVER
GO TO 100
106 GO SUB CURSOR
GO TO 100

```

Nach umfangreichen Änderungen sieht der Bildschirm anders als vorher aus. Eine Seite neu zu formatieren erledigt SEITENANDRUCK als erstes Untermodul. (Einzelbeschreibung der Module siehe Punkt Programmbeschreibung)

Normalerweise wird man aber nach Rücksprung beim Zeichenlesen landen, im Programm ZEICHENVER heißen. Nach jedem Tastendruck muß notgedrungen die Abfrage folgen, ob Text- oder Steuerzeichen eingegeben wurden. Ist letzteres der Fall, wird entweder auf die HANDLING-Ebene verzweigt oder aber INSERT bzw. ZEILENLOESCH angesteuert. (ZeichenDELETE und LeerzeilenINSERT erfolgen "unmittelbar" bei der Cursorbearbeitung)

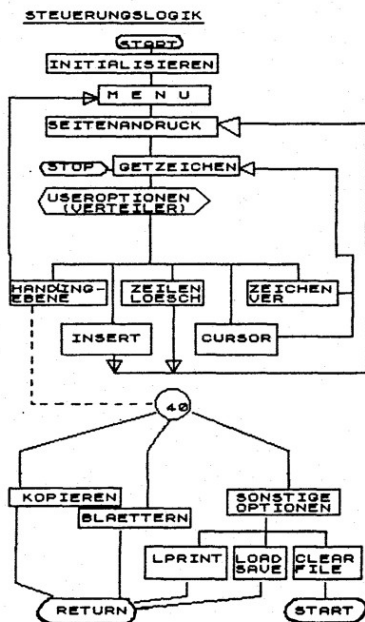
Liegen verarbeitbare (Text-)Zeichen vor, landet man im Modul mit dem treffenden Namen ZEICHENVER. Hier wird Text eingespeichert, angezeigt und Seiten-/Zeilenüberlauf gecheckt. Liegen keine Textzeichen mehr an bzw. wurden sie verarbeitet, sind Cursorangaben abzufragen. In Modul CURSOR wird auch der genannte Overflow-Check angesprungen, da ein Seitenwechsel durch Cursorbewegung verursacht sein kann. Nach alledem kehrt die Steuerung wieder an den Anfang zurück - spricht entweder zu

Menueanzeige  
Seitenandruck  
oder ZEICHEN

99



Wie man sehen kann, ist bereits eine Menge Programmlogik festgelegt, ohne daß ein einziger Modul-Befehl geschrieben wurde. Auch das ist ein Prinzip des TOP DOWN-Ansatzes, nämlich: Module von den Erfordernissen her zu definieren, die sie von der Steuerung erhalten. Hinzu tritt die sign. Schnittstellendefinition, d.h. es sind notwendige Ein- und Ausgaben festzuklopfen (Beachte dazu die genaue Modulbeschreibung, bei welcher in einer Extraspalte alle verarbeiteten Variablen bzw. weitere aufgerufene Module verzeichnet sind). - Schauen wir uns die STEUERUNGSLOGIK im Diagramm genauer an.



Was entnehmen wir daraus ?

Neben einer Reihe Details - z.B. Zeilenlösch wird durch NOT (= Code 195), INSERT mit dem Hesh-Zeichen (Code = 35) angestoßen - erkennen wir die notwendigen Module von TEXTVER als "black boxes". Darunter versteht man eine Funktionseinheit, deren Aufgabe, nicht jedoch konkrete Realisierung man weiß. Ehe wir in letztere einsteigen, sei der Initialisierungsteil gleich mit erledigt. Alle Sprungkonstanten müssen schließlich gesetzt sein, bevor die leeren "Schachteln" gefüllt wird.

```

110 REM INITIALISIEREN
115
120
121 INPUT "WIEVIEL SEITEN ? (MAX
122   20) : MAX
123   0 TO 121
124 LET G=MAX*704
125 LET MAX=MAX-1
126 LET X=0
127 LET Y=0
128 LET SEITE=10
129 LET ZEILEN=10
130 LET SPALTEN=10
131 LET CURSOR=25
132 LET HANDLING=36
133 LET INSERT=35
134 LET MENUE=15
135 GO TO 100
  
```

Wichtigste Aufgabe von INITIALISIEREN ist Dimensionierung des Datenarrays IS. Je nach Userinput wird aus der Seitenanforderung die Größe G errechnet. MAX gibt die maximale Screenzahl ab und ist die höchste fortgezählte Seite. SEITE und Zeilen/Spaltenzähler Y,X werden entsprechend auf Anfang gesetzt. Nun zu den einzelnen Modulen, die sich auch untereinander aufrufen können. Man braucht die konkrete Ausgestaltung zunächst nicht zu kennen; nur was sie leisten sollen und die übergebenen Variablen müssen bekannt sein. So lassen sich auch leicht "unbesetzte" Programmteile mit testen, sofern die "Versorgung" von Testgrößen sichergestellt ist (Was etwa mittels einer Reihe von LET-Anweisungen erfolgen kann). Man lasse sich nicht irritieren, wenn nicht alles sofort erläutert wird. Wichtig ist, was geschieht und nicht sogleich das was. Zudem sich oft für eine Funktion verschiedene Lösungen finden lassen, wie noch anhand eines Beispiels gezeigt werden wird.

Die Programmierung der Module

Fangen wir mit dem Einfachsten an - mit GETZEICHEN. Das Modul soll ein Zeichen von der Tastatur lesen, den Code auf STOP abprüfen, - sonst nichts.

```

0005 REM GETZEICHEN
0010 GO TO 100
0015
0020 REM GETZEICHEN
0025 INPUT "ZEICHEN : "
0030 LET Z=CODE INKEY$
0035 IF Z<>226 THEN RETURN
0040 STOP
  
```

In der Regel wird das Zeichen auf dem Screen dargestellt werden, sofern es druckbar ist, also als Textdatum in IS eingeht. Die mit dem Cursor mitlaufende ("synchronisierte") Position ist jeweils in Variable POS festgehalten. Das Modul "Drucke Zeichen" sieht ebenfalls recht einfach aus:

```
12 REM Drucke Zeichen
13 OVER 0
14 PRINT AT Y,X;T$(POS);
15 RETURN
```

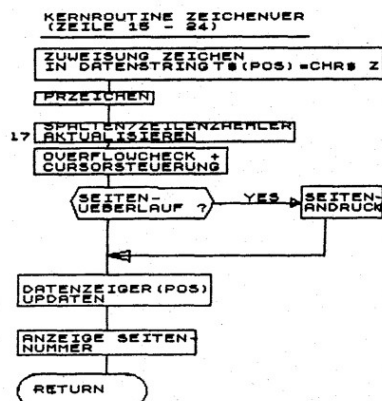
Ebenso simpel ist der Ausdruck einer Seite aus dem Textstring IS heraus. Das Teilstück wird einfach aus der laufenden SEITE ermittelt; als Hilfsgröße dient Variable S.

```
49 REM Drucke Seite
50 CLS
51 LET S=SEITE*704
52 PRINT T$(S+1 TO S+704);
53 RETURN
```

Solchermaßen vorbereitet können wir uns das eigentliche Kernmodul ZEICHENVER vornehmen. Es hat drei Hauptaufgaben:

1. Overflowcheck aller Cursor-, Zeilen- und Seitenüberläufe.
2. Einspeichern von Datenelementen in den Datenarray IS
3. Fortführen der aktuellen Positionierung mit eventuellem Seitenandruck bei "Überschreiten" der untersten Zeile

Schauen wir uns zuerst die reine Logik von ZEICHENVER an, bevor die nicht so einfachen Listingzeilen kommen.



Die Kernroutine verarbeitet zunächst das letzte eingegebene Textzeichen. Nach diesem "Kopfteil" kommt die wichtige Zeilen / Spaltenasynchronisation; hier werden die Cursor-Steuersymbole verarbeitet, sodaß der Teil Anspornpunkt für eine Reihe anderer Module darstellt.

Es schließt sich der Überlaufcheck an, d.h. wenn Spalte 33 erreicht ist, wird auf neue Zeile positioniert und Spalte auf 0 gesetzt. Und was ist zu tun, wenn man zur letzten Zeile einer Seite kommt? Dann wird Zeile / Spalte auf 0,0 gesetzt und die folgende Seite angedruckt. Hilfsvariable S in Zusammenhang mit der trickreichen SINCLAIR-Logik macht es möglich.

Abschließend muß noch der Datenzeiger POS, welcher den Bezug zum Datenarray IS fürs Drucken sicherstellt, fortgeführt werden. Die aktuelle Seitennummer erscheint zur Orientierung am Fuß im Input-Bereich. Wir bedienen uns dabei eines Microdrives-Statements, das auch ohne ein solches Gerät funktioniert. Mit PRINT # 1; AT ... kann der untere Screen-Bereich angesprochen werden, es lassen sich sogar bestimmte Zeilen/Spalten ansprechen. Allerdings achte man darauf, möglichst nur die erste Zeile zu benutzen, weil ansonsten der übrige Bildschirm hochgeschoben wird. Und noch was ist wichtig: Da ein INPUT-Befehl das "Geprintete" überschreibt, muß mit PAUSE 0 und INKEY\$ gearbeitet werden. Aber schauen wir uns das Listing von ZEICHENVER doch gleich an.

```
14 REM Drucke Zeichen
15 LET S=(POS)/CHR$(Z)
16 GO TO SUB PRUEFE ZEICHEN
17 GO TO 19
18 LET X=X+(Z=9)-(Z=0)
19 LET Y=Y+(Z=10)-(Z=11)
20 LET X=X+(X>31)-(X<0)
21 IF X<0 THEN LET X=31
22 IF X>31 THEN LET X=0
23 LET S=(Y/21)-(Y<0)
24 LET Y=Y+(S=0)+21*(S<0)
25 LET SEITE=0 THEN LET SEITE=0
26 IF SEITE>MAX THEN LET SEITE=MAX
27 IF S THEN GO SUB SEITENANDR
28 UCK
29 LET POS=SEITE*704+X+1+Y*32
30 PRINT #1; INVERSE 1; AT 0,11
31 SEITE
32 RETURN
```

Mit der eleganten Verwendung boolescher Operatoren gelingt das Kunststück, ohne viele IF's die gesamte Überlaufsteuerung für alle Bildschirmbewegungen zu verkraften. Zur Erinnerung sei auf die vortrefflichen Möglichkeiten SINCLAIR-spezifischer Vergleichsoperationen zurückgekommen; die Größen X und Y können zudem auch "expression" (zusammengesetzte Ausdrücke sein):

X AND Y ergibt X, wenn Y "true" (bestätigt) X

ergibt 0, wenn Y "false"  
 X OR Y ergibt 1, wenn Y "true"  
 ergibt X, wenn Y "false" (bestätigt X)  
 Vergleiche (z.B. X=Y) ergeben 1, wenn "true"  
 ergeben 0, wenn "false"

Boolesche Vergleiche lassen sich - wie gezeigt - sinnvoll zum Initialisieren von Null / Eins und zum "An- und Abschalten" von Werten einsetzen.

Mit ZEICHENVER die Cursorcontrol im Griff fehlt zur Bildschirmdarstellung eigentlich noch ein Cursorzeichen. Es muß vorhandenen Text "übergehen" und Stellen gut sichtbar markieren. Wir haben uns für ein schwarzes Viereck entschlossen, das mit OVER 1 über Text wandert. Außerdem ist in Modul CURSOR gleich DELETE einzelner Zeichen (Code 12) und Einfügen platzschaffender Leerzeilen mit eingebaut worden.

```
25 REM ZEICHENVER
26 GO SUB PRZEICHEN
IF Z=12 THEN LET T=T*( TO POS-1)+T*(POS+1 TO G)
LET Z=0
27 IF Z=13 THEN LET X=32
GO TO 19
28 OVER 1
PRINT AT Y,X;"■";
RETURN
```

Bei einer weiteren Editing-Funktion, dem INSERT, müssen wir uns etwas einfallen lassen. Wie soll der einzufügende Text am besten in IS "implantiert" werden? Und wie kann man den User optisch darauf aufmerksam machen, daß er sich in "Einfüge-Modus" befindet?

Würde man Einfügungen Zeichen für Zeichen - wie bei DELETE - verarbeiten, käme das Ganze recht langsam. Also legen wir einen Zeichenstring an, wozu die ursprüngliche Position des Files (POS) "gemerkt" wird. Leider ist INPUT nicht anwendbar, wie bereits erwähnt. Also bleibt zum sichtbaren In-Screen-Andruck der INVERSE-Print. Ende der Eingabe bei ENTER; wenn nichts eingegeben wurde, soll eine Leerzeile "insertet" werden. Ansonsten können wir für Fragen des Überlaufs Modul ZEICHENVER anspringen.

```
29 REM INSERT
30 LET A=POS
LET X=""
OVER 0
31 GO SUB 16
GO SUB GETZEICHEN
IF Z<13 THEN LET X=X+(CH
AND Z>31)
PRINT INVERSE 1;AT Y,X;X
GO TO 31
32 IF X="" THEN LET X=""
33
34 LET T=T*( TO A)+X+T*(A+1
TO G)
INVERSE 0
RETURN
```

INSERT besitzt eine eigene Eingabesteuerung, nämlich solange, wie Zeichen eingegeben werden. Danach wird der String X in File eingefügt, INVERSE wieder zurück gesetzt und abschließend ein Seitenendruck gebracht (s. Hauptsteuerung).

Den Zeileninsert steht zur Seite die Zeilenlöschung. Exakt ab laufender Position sollen die folgenden 32 Zeichen aus dem Textfile IS herausgenommen werden. Es empfiehlt sich daher, an den Zeilenanfang zu positionieren, falls man genau eine Screen-Zeile weg haben will.

```
35 REM ZEILENLOESCHUNG
36 LET POS=POS-(X+1)
LET T=T*( TO POS)+T*(POS+3
2 TO G)
37 LET POS=POS OR POS<=0
38
```

Damit ist alles, was mit unmittelbarer Eingabe und EDITING zu tun hat, erledigt. Bleibt das HANDLING vorhandenen Textes; darunter fallen dann Dinge wie Blättern, Textkopieren usw.

#### Text-HANDLING

Auf die HANDLING-Ebene kommt man mit 'Symbol Shift U' (= "OR"). Es soll dem Benutzer deutlich gemacht werden, daß er keinen Text mehr eingibt, sondern diesen manipuliert. Daher die Trennung vom reinen EDITING. Optisch hervorgehoben erscheinen auf der Seiten-Zeile die gewählten Optionen sowie weitere Hinweise. Schon damit ist erkennbar, daß man sich auf einer anderen Ebene befindet.

Das folgende Modul stellt so was wie einen "Kopf" dar. Wird etwas anderes als "Kopieren" gewählt, wird sogleich in andere Untermodule gesprungen. Sehen wir uns den ersten HANDLING-Teil einmal an.

```
39 REM HANDLING
40 PRINT AT 0,0; BRIGHT 1;"
HANDLING";
PAUSE 0
IF A$="INKEY$
THEN GO SUB 130
RETURN
41 LET A=POS
LET X=""
PRINT X;"
"KOPIE" ENDE
42 REM ZEILENLOESCHUNG
GO SUB GETZEICHEN
IF Z<13 THEN GO SUB CURSOR
GO TO 42
43 LET X=T*(A TO POS)
PRINT #1; BRIGHT 1;AT 0,19;
"ZIEL"
44 GO SUB GETZEICHEN
IF Z<13 THEN GO SUB CURSOR
GO TO 44
45 LET T=T*( TO POS)+X+T*(PO
S+1 TO G)
GO SUB SEITENANDRUCK
PAUSE 0
RETURN
```

Der Start ist nicht weiter von Belang, interessant ist die Copy-Funktion. Darunter ist keine ScreenCopy zu verstehen, sondern die Anlage eines duplizierten, markierten Textstückes ab einer gewählten Zielposition. "KOPIE" arbeitet ähnlich wie die INSERT-Routine, hat also auch eine eigene Steuerung. Zunächst wird von laufender Position ausgehend das Ende des zu kopierenden Textes angesteuert; dann ist die Zielposition anzukommen und gleichermaßen mit ENTER zu quittieren. Die Prozedur endet mit Anlage der Textkopie in IS (aus Zwischenstring XS) und Seitenendruck. - Man hätte das Ganze auch über eine INTEXT-Routine abwickeln können, die von INSERT und KOPIE angesteuert wird. Aber das wäre unübersichtlicher.

Durch die bisherigen Vorbereitungen ist Modul BLAETTERN nun einfach. Es braucht lediglich auf "left" (Code 9) und "right" (Code 8) abgefragt werden, um einen Seitenswechsel zu provozieren. Mittels vertrauter Cursorsteuern kann beliebig in File geblättert werden. Alle anderen Tasten bewirken RETURN und damit - allgemein bei HANDLING - die Menüanzeige.

```

130 REM *****
132 IF AS=CHR$ 8 OR AS=CHR$ 9 THEN
  LET Y=Y+22-123 AND (AS=CHR$ 9)
  GO SUB 20
  PRINT "BRIGHT 1: AT 0,19";
  "BLAETTERN/\\\";
  PAUSE 0
  LET AS=INKEY$
  GO TO 132

```

#### Die Formatierung von Texten

##### Zwei Lösungen zur Auswahl

Formatierung heißt Textausgeben in eine lesergerechte Form zu bringen. Meist ist darin die Übergabe von Steuercodes für Normalspieldrucker eingeschlossen. Im engeren Sinn kann man unter Formatierung die Aufbereitung mit vorgegebener Zeilenbreite und ohne Wortbrüche verstehen, als Minimalforderung sozusagen...

Bei der Realisierung in TEXVER gab es zwei Lösungswege zum sgn. Flattersatz. Unter letzteren ist zu verstehen die linksbündige Ausrichtung des Zeilenanfangs, wobei der rechte "Rand" ein ausgefranztes Aussehen erhält. Es wird jeweils bei Zeilenende geprüft, ob ein Wort noch (in die vorgegebene Breite) paßt. Ist das nicht der Fall, bleibt der Rest der Zeile frei, das Wort wird in die neue Zeile "gezogen".

Nun führen bekanntlich viele Wege nach Rom. Dementsprechend gibt es mehrere Möglichkeiten der Formatierung in BASIC. Schildern wir auch hier den Werdegang beider Module, die alternativ sind. Idee ist jeweils eine andere: bei "FLATTERSATZ", der zuerst entstand, läuft die Formatierung über eine Zwischenstring XS;

Modul "LPRINT", die zweite elegantere Lösung, geht den Textarray IS direkt durch und besorgt danach den Zeilenvorschub. Beide Module lassen sich für die formatierte Ausgabe beliebiger Druckbreiten - also auch Non-2X-Geräte - einsetzen. Man muß lediglich die Variablenverwendung anpassen:

```

FLATTERSATZ Textstring IS
             Druckbreite Z
             Zwischenstring XS
             Zählvariablen A, E, N

```

```

LPRINT : Textstring IS ( dessen Länge in G)
         Druckbreite Z
         bei Andruck ab einer bestimmten Position -> POS
         Zählvariablen A, E, N

```

##### Die Lösung mit FLATTERSATZ

Bei diesem Weg der Formatierung geht es, das Problem "word wrapping" (= Worttrennung) mittels eines Zwischenstrings in den Griff zu bekommen. Schwierigkeiten gibt es bei erstem Füllen von XS und beim Ausgeben des Textrestes am Ende. "Vor-" und "Nachlauf" sind extra zu programmieren, um diese Besonderheiten abzudecken.

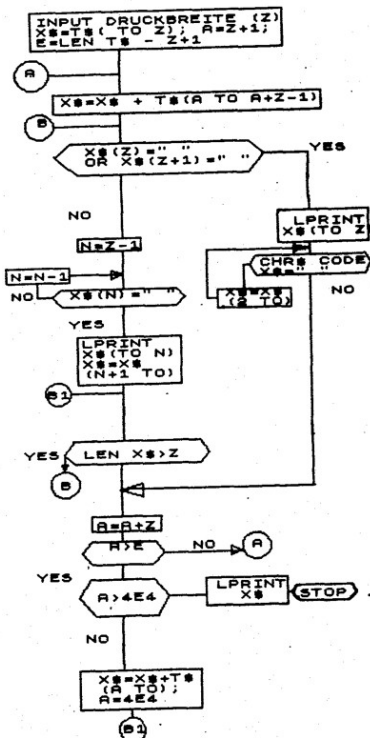
Im Hauptteil des Moduls läuft die Worttrennung am Zeilenende nach folgendem Muster ab:

Vorschub auf neue Zeile erfolgt, wenn entweder letztes Zeilenzeichen Space oder letztes Zeilenzeichen Buchstabe und erstes Zeichen einer neuen Zeile Space ist. Dann liegt ein genau mit der Zeilentrennung synchron laufender Wortbruch vor. - Das aber ist die Ausnahme; was geschieht, falls ein Wort über den "Zeilenrand" hinaus reicht?

In dem Fall muß das zwischengespeicherte Textstück solange "von hinten" durchsucht werden, bis die erste Wortlücke auftaucht. Bis dahin r wird gedruckt, dann folgt Zeilenvorschub und Fortsetzung mit dem folgenden Wort. Führende Spaces sind dabei zu eliminieren, um ein linksbündiges Bild zu gewährleisten.

Die daraus zu entwickelnde Programmierlogik bedient sich des Strings XS, der den laufenden Text zunächst in d o p p e l t e r Breite aufnimmt. Warum das? Nur wenn die Zeilentrennlinie innerhalb des Zwischenstrings liegt, läßt sich der "Trenncheck" unkompliziert erledigen. - Aber schauen wir uns dazu die "Logik" an.

LOGIK FLATTERSATZ  
(ZEILE 160-179)



Auf zwei Besonderheiten sei hingewiesen. Schließlich gilt es, nicht bloß "kopierfertige" Lösungen zu bringen, sondern eigenes Tun zu fördern..

1. Wenn Zwischenzähler A > E (wie Ende) ist, bekommt A den Wert 4E4 (= 40.000); danach erfolgt nochmal ein "kleiner" Durchlauf - Grund: Um den letzten Textrest auszudrucken, wird A quasi als Endeschalter benutzt und mit Endekriterium 40.000 besetzt.

2. Nach "erfolgreichem" Ausdruck (d.h. genau am Zeilende war eine "Kluft") wird XS in der Form CHR\$ CODE XS abgefragt; warum nicht so: IF XS(1) = " " THEN..? Grund: Bei letzterer Form würde, wenn XS genau Z + 1 lang und letzte Stelle = Space, u.U. ein falsches Subskript entstehen. XS wäre ein Leerstring geworden und würde mit 1 qualifiziert - Folge: Abbruch mit "subscript wrong"... Mit CHR\$ CODE XS wird genau das Gleiche erreicht, allerdings auch, wenn XS ein Leerstring ist. Es kommt zu keinem Fehlerstop!

Das Ganze sieht programmiert so aus; weshalb der anderen Lösung (LPRINT) der Vorzug gegeben wurde, werden wir als nächstes zu diskutieren haben.

```

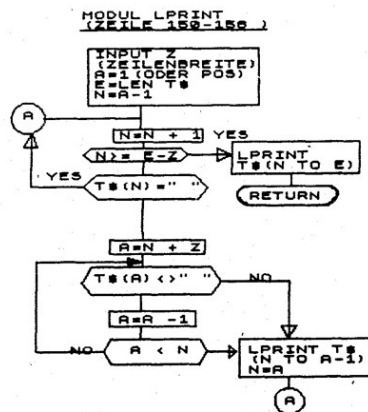
1500 REM *****
1501 INPUT "DRUCKBREITE ?":Z
1502 IF NOT Z THEN GO TO 162
1503 LET XS=TS( TO Z)
1504 LET B=LEN TS - Z+1
1505 LET XB=XS+TS(A TO A+Z-1)
1506 IF XS(Z)="" OR XS(Z+1)="" THEN
1507   LPRINT XB(TO Z)
1508   GO TO 167
1509 LET N=N-1
1510 IF XS(N)="" THEN LET N=N-1
1511 GO TO 169
1512 LPRINT XS(TO N)
1513 XS=XS(N+1 TO )
1514 LET LEN XS > Z THEN GO TO 167
1515 LET A=A+Z
1516 IF A > E THEN GO TO 165
1517 LPRINT XS
1518 LET B=B+Z
1519 IF A > 4E4 THEN GO TO 175
1520 XS=XS+TS(A TO )
1521 A=4E4
1522 GO TO 162
1523 LPRINT XS
1524 RETURN
1525 LPRINT XS( TO Z)
1526 LET XS=XS(Z+1 TO )
1527 IF CHR$ CODE XS="" THEN LET
1528   GO TO 175
1529 GO TO 172

```

## Die Lösung mit LPRINT

Modul LPRINT als zweiter Ansatz ist nicht nur kürzer, sondern auch eleganter. Einmal wird kein Zwischenstring benötigt, zum zweiten beschleunigt sich der Ablauf mittels einer FOR-NEXT-Schleife (wie ist in der folgenden Logik der Übersichtlichkeit halber aufgelöst worden). Wenn gleich jedes Zeichen von IS auf Space durchgeprüft wird, der Rechner also mehr "Arbeit" hat, ist LPRINT nicht langsamer. Auch der umständlich "Vor-" und "Nachlauf" fällt weg, was der Programmlänge gut bekommt. Wie geht das Modul vor?

Ab gegebenem Anfang (1. Zeichen oder ab POSITION) wird IS durchgezählt und auf Space gecheckt. Solange das letzte Teilstück nicht erreicht ist, wird auf Wortlücken abgestellt. In der kleinen (FOR-NEXT-)Schleife geht es ab Wortanfang weiter, bis eine "Kluft" - Rückwärts gezählt - erreicht ist. Dann wird der bisherige Textrest ausgedruckt und N auf den aktuellen, neuen Wortanfang gesetzt. Anschließend erhöht die "große" Schleife Zähler N, bis wieder ein Wortteil (d.h. keine Spaces mehr) "getroffen" ist. Die dauernde Initialisierung von FOR-NEXT bedeutet für den SINCLAIR zwar eine Menge Schufferlei, geht aber recht schnell. - Bei Textende, wenn noch ein Rest (Länge E - Z) abzuarbeiten ist, wird er direkt ausgeprintet und der Weg ist frei zum RETURN..



110

Die programmtechnische Lösung ist einfach "herunterprogrammiert". Um das Ganze optimal ins übrige Programm einzubetten, wurde noch ein wenig "Optik" miteinbezogen. Ferner ist die Option "Drucken ab laufender Position" durch "Logikvergleich" realisiert (s. Zeile 154). Man kann auch die "große" Schleife per FOR-NEXT erledigen, was so aussehen dürfte:

Zeile 155 : FOR N = N TO E - Z - 1

Ende Zeile 158 (statt GOTO 155):

NEXT N : LPRINT T\$(N TO E) : RETURN

E wie Ende ist auf G (= Gesamtlänge von IS) gesetzt, kann jedoch auch eine zweite Position im Text selber erhalten; das war der Grund für die Zwischenspeicherung. Dem SINCLAIR-Programmierer stehen hier weitere Möglichkeiten der Gestaltung offen.. Doch nun das Listing selber.

```

150 REM FORMATIERTER MODUL
151 PRINT #1; BRIGHT 1; AT 0,19;
    "LPRINT"
152 INPUT "DRUCKBREITE ?": Z
153 INPUT "1=ab POSITION, 2=GESAMTE
    LÄNGE": A
    IF A < 1 OR A > 2 THEN GO TO 15
154 LET A=POS OR A=2
    LET E=G
    LET N=A-1
155 FOR N=N TO E-Z-1
    LPRINT T$(N TO E)
    NEXT N
156 RETURN
157 FOR A=N+Z TO N STEP -1
    IF T$(A) <> " " THEN GO TO 155
158 LPRINT T$(N TO A-1)
    LET N=A
    GO TO 155
  
```

Damit sind wir am Ende der Programmentwicklung. Ein paar Screencopies sollen noch die "Optik" ein wenig veranschaulichen. Nachzutragen bleibt die formale Programmbeschreibung; in ihr ist das Bisherige stichwortartige wieder zu finden. Auch für den Fall, daß man selber am Programm etwas tun möchte, ist die Dokumentation ganz nützlich.

```

133 REM MODUL
134 IF A$="P" THEN GO TO 150
135 IF A$="C" THEN
    PRINT #1; BRIGHT 1; AT 0,19;
    "CLEAR FILE";
    PAUSE 0
136 IF A$="L" AND A$ <> "S" THEN
    RETURN
137 PRINT #1; BRIGHT 1; AT 0,19;
    "LADEN" AND A$="S"; ("SPEI
    CHERN" AND A$="S");
140 INPUT "FILENAME ?": IS
    IF IS="" THEN GO TO 140
143 IF A$="L" THEN LOAD IS DATA
    T$
    LET G=LEN T$
    LET MAX=INT (G/704) - 1
    LET X=0
    LET SEITE=X
    LET POS=1
144 IF A$="S" THEN SAVE IS DATA
    T$
145 RETURN
  
```

111

MEINE DAMEN UND HERREN,  
ES IST MIR EINE BESONDERE EHRE, DIE HONORATIIONEN DER STADT  
HIER BEGRUESSEN ZU KOENNEN. MINDESTENS 10 MILLIONEN UMSATZ  
SEH ICH VERSAMMELT, VOM STEUERAUFKOMMEN GANZ ZU SCHWEIGEN.  
SIE ALLE WOLLEN DOCH SICHER, DASS ETWAS FUR JUGND UND  
GESUNDHEIT GETAN WIRD. WIE SIE ALLE WISSEN, IST EIN NEUES  
SCHWIMMBAD IN UNSERER GEMEINDE SEIT LANGEM GEPLANT.  
WESHALB NICHTS GETAN WURDE, LAG AN LAECHERLICHEN 2  
MILLIONEN. WIE WAER ES MIT EINER KRAEFTIGEN SPENDE ? ALSO,  
GEBEN SIE IHREM HERZEN UND GELDBEUTEL EINEN STOESS UND  
ENTRICHTEN EINEN OBULUS.  
WER NOCH ZOEGERT, SEI VERSICHERT, ER KOMMT NICHT EHER HIER  
WEG, BIS DAS GELD BEISAMMEN IST.  
SAALDIENER, DIE TUEREN VERRAMMELT UND KEINEN RAUS LASSEN.  
ICH VERABSCHIEDE MICH ERST MAL FUER DREI STUNDEN. ICH  
KOMM DANN ZURUECK, UM ZU SEHEN; OB DAS GELD IM BEUTEL  
REICHT.  
GERD PFIFFIG

(GEMEINDEDIENER)

Programmdokumentation

Modulbeschreibung

Zeilen-Nr.	Titel u. Kurzbeschreibung	Verwendete Variablen(Module)
10 - 11	GETZEICHEN INKEYS-Zeichenzuweisung (Z) und Stop-Abfrage	Z
12 - 13	PRZEICHEN Andruck des Datenstrings (IS) mit laufender Position auf dem Screen	IS(POS) Y,X(-Zeile/Sp.)
14 - 24	ZEICHENVER Zuweisung Zeichen in Datenstring + Andruck Aktualisieren Zeilen/Spaltenwert gem. Cursor; Überlaufcheck Fortschreiben Seiten-Nr. u. Datenpointer (POS) Andruck Seite bzw. Seiten-Nr.	(PRZEICHEN) IS(POS) Y,X SEITE,POS (SEITENANDRUCK)
25 - 28	CURSOR	

	Andruck akt. Zeichen Zeichendelete u. Zeilenvorschub OVER-Print d. Cursorzeichens	(PRZEICHEN) (SEITENANDRUCK) (ZEICHENVER) Z,X,Y IS(POS)
29 - 34	INSERT Zwischenspeichern akt. Posit.(in A) Endeabfrage, sonst Zeichenspeichern in String XS Andruck (invers) d. eingegeb. Textes ggf. Leerzeilen-INSERT Einfügen Hilfsstring in Textdaten	(GETZEICHEN) A,POS Z, XS Y,X IS
35 - 38	ZEILENLOESCH Textverkürzung ab akt. Position Positionsscheck	IS,POS,X
39 - 45	HANDLING Anzeige und Abfrage User-Option "KOPIE"-Routine: - Marker Textende (Anfang-POS) setzen - Zwischenspeichern Text - Zielpos. setzen - Text kopieren + Seitenandruck	(GETZEICHEN + CURSOR) AS,XS A,POS IS (SEITENANDRUCK)
49 - 50	SEITENANDRUCK Löschen Screen, Errechnen Start/Ende mittels d. akt. SEITE Andruck Daten auf ScreenSeite	S,SEITE
100 - 118	HAUPT-MENU <s. eigene Logik> Anzeige d. Optionen Andruck akt. Seite Einlesen Zeichen + Steuern Verarbeitung Cursor-Abfrage	Z (GETZEICHEN, SEITENANDRUCK, INSERT, ZEILEN- LOESCH, ZEICHEN- VER, CURSOR)
119 - 126	INITIALISIEREN Abfrage Seiten + Errechnen Dimensionierung Textarray (IS) Erstbesetzung Print- + Posit.variablen Aufbau des Sprungzieltabelaus	MAX,G Y,X,SEITE,POS
129 - 132	BLAETTERN Cursorcheck + Seite auf Überlauf setzen Ansteuern Seitenandruck(in ZEICHENVER)	(ZEICHENVER) AS,Y



133 - 146	INKEYS-Abfrage f. Weiterblättern	
	SONSTIGE OPTIONEN	AS, IS (wie b. INITIA- LISIEREN)
150 - 158	Sprung z. formatiertem LPRINT LOAD / SAVE Clear File	
	LPRINT (s. Einzellogik)	A, Z E, N, G IS, POS
	Bestimmen Start u. Druckbreite WORDWRAP-Check Andruck Textzeile	

#### Variablenliste

Strings IS Textdatenarray  
AS Handling-Optionen u. Abfrage b. BLÄTTERN  
XS Zwischenstring f. KOPIE u.  
weitere Intext-Funktionen  
IS Ladenname b. LOAD/SAVE

#### Konstanten

GETZEICHEN=10  
PRZEICHEN=13  
SEITENANDRUCK=50  
CURSOR=25  
ZEILENLOESCH=36  
HANDLING=40  
INSERT=30  
ZEICHENWER=15

#### Num. Variablen

A Startmarkierung f. Intext-Operationen  
E Endemarkierung b. format. Textausgabe  
Y Zeile  
X Spalte  
Z Code d. eingelesenen Zeichens  
S Hilfsvariable f. Seitenandruck  
SEITE akt. Bildschirmseite  
POS Datenzeiger akt. Textstelle (in IS)  
MAX max. reservierte Seitenzahl  
S max. reservierter Datenbereich  
N Zählvariable

#### Benutzungshinweise

- > Theoretisch könnte man den RAM bis auf etwa 35 K für Text benutzen. 50 Bildschirmseiten sind jedoch wegen des "Slicings" entschieden zuviel; man begnüge sich beim Initialisieren mit höchstens 20 Seiten.
- > Zwei Funktionen sind nicht im Menu angezeigt - Zeilenlöschen (mit Symb.Shift "NOT") und Leerzeilen-Insert (nach INSERT Eingabe = ENTER). Es werden exakt 32 Zeichen - gleich eine Bildschirmzeile - ab laufender Position benutzt.
- > Zeichenweises DELETE läuft wie beim Programmediting, nämlich mit Caps Shift "0" (Delete-Taste). Da der gesamte Textarray um ein Zeichen verschoben wird, ist die Bewegung ein wenig schleppend. Bei größeren "Löscharbeiten" empfiehlt sich zeilenweises Vorgehen.
- > Im HANDLING-Modus (mit "OR" erreicht) erfolgt das Kopieren ab laufender Position; man stelle den Anfang daher vor Kommandoaufruf ein. Ende ist die mit ENTER quittierte Textstelle inklusive. Kopiert wird hinter die markierte Zielposition.
- > Bei der formatierten Druckausgabe wähle man die Zeilenbreite nicht zu klein, sonst kann es zu Verklemmungen kommen. Das breiteste Wort ist Minimalgröße. Auch besichte man, daß Leerzeilen bei LPRINT nicht gedruckt werden.
- > Blättern wird initialisiert mit < > Cursor, danach erst geht das eigentliche Paging gemäß Pfeilrichtung vorstatten.
- > Da mit INSERT und DELETE das Textfile neu ausgerichtet wird, ist es zweckmäßig, die optische Feinarbeit (Abwätze, Einrückungen etc.) erst nach vollständig redigierter Eingabe vorzunehmen

Ansonsten ist TEXVER frei für Verbesserungen aller Art (z.B. Blockoperationen, Find & Replace, Drucksteuerzeichen etc.). Wir wünschen dem durch dieses Buch geschulten Hobbyprogrammierer dazu viel Erfolg.

```

RECHNER-TEXT
E D I T I N G
CURSOR / \ / < >
CAPS SHIFT 0 = DELETE
(ZEILENLOESCHEN="NOT")
# = INSERT
( " " = LEERZEILE )
STOP=PROGR.ENDE
H A N D L I N G
(UMSCHALTER="OR")
CURSOR / \ / BLÄTTERN
SK-OPIEREN
SS-AUE/L-ORD
P-PRINT DRUCKER
C-CLEAR FILE

```

## Kapitel IV: Verarbeitung von Daten

### DATEN IN LISTEN

#### Arbeiten mit Records + Reports

##### Datenorganisation

In diesem Kapitel werden wir uns mit dem eigentlichen Datenhandling beschäftigen. Wir haben die Rubriken "Text" und "Zahlen" behandelt, nun geht es um "Daten". Darunter ist die Kombination beider als verfügbare Elemente programmtechnischer Aktivitäten zu verstehen. Die wichtigsten Fragen ranken sich - wie bereits in Kapitel III angesprochen - um den Komplex Datenorganisation und Speicherung. Wie ordnet man Daten, um Informationen zu speichern, wie manipuliert man sie zum Zweck der Präsentation?

Zur Speicherung werden wir uns noch äußern, im Mittelpunkt steht zunächst die Datenorganisation. Kernpunkt ist, wie man logisch zusammengehörende Informationen in eine Form bringt, die mit dem SINCLAIR handhabbar ist. Damit verbunden stellt sich das Problem, Daten so aufzubereiten, daß sie "ein gutes Bild" für den Benutzer abgeben. So heißt denn auch der folgende Abschnitt Records und Reports. Bevor wir auf die Verknüpfung von Daten in Listen eingehen - Dinge wie Suchen und Sortieren etc. kommen in der zweiten "Abteilung" - , seien ein paar kurze Begriffserklärungen gemacht:

Reports sind "Listen", Masken, Layouts für die stellengerechte Platzierung von Daten auf dem Bildschirm. Records sind logische Einheiten ("Sätze") aus einem geordneten Datenbestand ("File"), die dort mehrfach abgelegt sind.

Daten können in verschiedenen Typen auftreten. In BASIC gibt es davon nur zwei - Floating-Point Zahlen und Zeichenketten (strings). Leider kennt das SINCLAIR-Basic keinen Zahlentyp INTEGER (Ganzzahl in zwei Bytes im Bereich von positiv 0 - 65536), wenngleich über die CODE-Klausel Ersatz möglich ist; auch der PASCAL-Typ Boolean ist nur indirekt handhabbar (u.z. als logisches Vergleichsergebnis). Andere Datentypen, jenseits von BASIC in Hochsprachen bekannt, sind nicht möglich. Was uns besonders interessiert, ist der Verbundtyp 'Record'. Er stellt die Verbindung der beiden BASIC-Typen "Zahl" und "Zeichen" dar, was anderes ist im Beginners all purpose symbolic instruction code (B-A-S-I-C) sowieso nicht machbar. Wie schafft man es, einen Verbund von Zahlen und Zeichen herzustellen, der als gleiche Folge ("Satz") immer wieder vorkommt und ein File bildet?

Zeichen und Zahlen können jeweils einzeln als Feld mit eigenem Namen isoliert sein oder als Tabelle (engl. Array). Die Zeichenkette (als single string) hat dabei noch den Vorteil, längenmäßig beliebig manipulierbar zu sein. Erster Gedanke zum künstlichen Verbundtyp wäre - man transformiert Zahlen in die Stringform (was mit STR\$ ja geht). Dann sind Zahlen und Texte gemischt, und man braucht sich nur noch über die Frage - Einzelstring oder Stringarray - zu unterhalten. Das hat einen gravierenden Nachteil: Will man die vollen 8 Stellen (plus evtl. Dezimalpunkt) einer Floatingpointzahl nutzen, wird die Stringzahl zu speicheraufwendig, von anderen Nachteilen einmal abgesehen. Die normale Zahl kostet bekanntlich nur 5 Bytes ohne Bezeichner. Was also ist zu tun, um einen Satz wie folgt zu verwalten?

```
Beispiel: NAME (Text)
          ANSCHRIFT (Text)
          ALTER (Zahl)
          GEXALT (Zahl)
          QUALIFIKATION (Zahlencode 0-9)
```

Zunächst machen wir aus Vereinfachungsgründen eine Einschränkung - die Texte sollen jeweils eine Maximallänge von z.B. 20 Stellen haben. Das bedeutet, wir arbeiten bei Text nicht mit "Stellenanzeigern" (Fachterminus POINTER), sondern speichern die Information NAME und ANSCHRIFT in Stringarray-Form. Bei Zahlen liegt die Tabellenform sowieso nahe.

Bleiben wir vorerst beim Datenteil und seiner Organisation. Da bietet sich zur Unterscheidung die Dimensionierung von zwei Arrays an (z.B. mit DIM D (RECORDS,3) und DIM DS (RECORDS,3,20)). Nun taucht ein Problem auf: Die Verbindung von Satzteil (fortlaufend) zu Datenarray (nicht fortlaufend) ist schwierig herzustellen. Bei einem angenommenen aktuellen Record (z.B. R=7) läuft ein Ausdruck wie folgt:

lfd. Nr.	Item(-Satz- element)	Datenelement mit gültigem Subskript
(1)	PRINT "NAME ";	DS (R,1)
(2)	PRINT "ANSCHRIFT ";	DS (R,2)
(3)	PRINT "ALTER ";	D (R,1)
(4)	PRINT "GEXALT ";	D (R,2)
(5)	PRINT "QUALIFIKATION ";	D (R,3)

Einfacher und flexibler ist es, die laufende Item-Nummer dazu zu benutzen, um mittels eines Unterscheidungsmerkmals in einem Zahlenarray das betreffende Datenelement direkt abzurufen. Für Zahlen geeignet wäre etwa die Charakteristik "Negativ-" oder "Positivwert". Gesetzt den Fall, P(1..7) enthält die Zuordnung zu den Datarrays, würde P(4) = 1 bedeuten: Item Nr. 4 ist in Datenelement 1 von D(...) zu finden; wo "Anschrift" (=Item Nr. 2) steht, zeigt der Zugriff auf P(2). P(2) hätte den Wert -2, was bedeutet: Item Nr. 2 steht im Textarray (Negativzahl) an Stelle 2 = DS(R,2). So auch realisiert in REPDATA.. Störend ist und bleibt beim so gewonnenen "Verbundtyp" noch die

starre Vorgabe der Bezeichner. Dem kann man leicht abhelfen. Die Bezeichner werden nicht vorgegeben, sondern von Benutzer nach seinen Wünschen bestimmt und in einem eigenen Bezeichnerstring gespeichert. Das hat noch einen weiteren Vorzug: Die Reports (quasi als Berichtslisten) können sie gleichfalls verwenden und folglich flexibel gestaltet werden. Wie geht man diesen Darstellungsteil nun an?

Eine Liste (report) wird geschaffen, indem Items über ihre Bezeichner in ein von Benutzer gestaltetes 'Layout' gebracht werden. Letzteres beinhaltet nicht nur Reportnummer und Titel, sondern zusätzliche Informationen. Weiter gefragt - wie wird die Liste gestaltet?

Zuerst muß man wissen, wie die Verbindung Feldbezeichner und Bildschirm ("Formular") zustande kommt. Die Zuordnung kann - neben dem Titel und sonstigen Rahmenwerk - über eine 'Liste' erreicht werden. Ein Zahlenarray, der Bezeichner-, Zeilen- und Spaltennummern enthält, ist naheliegend. Es gibt aber einen ökonomischeren Weg - über einen String (RS wie Reportstring). Da Positionen nur die Werte 0 - 22 (Zeile) bzw. 0 - 31 (Spalte) annehmen, empfiehlt sich die Verwendung von Zeichencodes. Damit ist bekanntlich der Ganzzahlbereich von 0 - 255 benutzbar. Am besten speichert man alle zum Report notwendigen Informationen gleich mit ab, als da sind TITEL, BEZEICHNER, STELLE (Zeile, Spalte pro Feld).

Bei mehreren Reports taucht das Problem des Zugriffs auf diese Reportinformationen auf. Wie ist das am besten zu organisieren? Da die zur Erzeugung einer Darstellung notwendigen Angaben (in RS) hintereinander liegen, wäre es zu umständlich, bei Generierung den String voll "durchzumudeln". Die Angaben sind variabel, da ein Report einen lang ein anderer einen kurzen Titel hat, der eine Report mit 3 Bezeichnern arbeitet, während der andere z.B. Informationen von 7 Bezeichnern braucht usw. - Die beste (nicht unbedingt übersichtlichste) Lösung wird einen Datenzeiger bemühen. Er zeigt bei vorgegebener Reportnummer immer auf den Anfang der Informationen (in RS). Die lassen sich dann schnell, d.h. im direkten Zugriff, abfragen und verarbeiten.

Nachdem aufgezeigt wurde, wie Daten und Bezeichner organisiert werden, wie bei Reports über Feldbezeichner die Verbindung zu Daten hergestellt wird, kommen wir zur eigentlichen Synthese von Daten und Listen.

#### Die Synthese von Daten und Listen

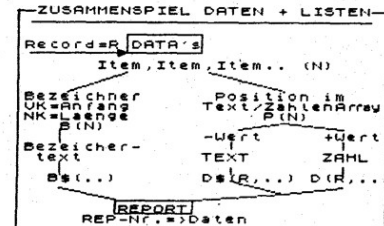
Ausgangsfrage ist - wie verknüpft man aktuelle Satzfelder mit der Platzierung auf einer Bildschirm-Liste?

Hauptproblem ist die Transformation der Datenanordnung in die Listform. Da ein Satz Zeichen und Zahlen (in Arrays) umfaßt, ist

die Verbindung zum definierten Report herzustellen. Er sieht die Platzierung von "NAME, ANSCHRIFT, GEGENSTAND" u.U. in ganz anderer Folge vor. kann. - Die Lösung erfolgt über die Bezeichner in Verbindung mit schon angesprochenen Datenzeigern. Der Bezeichner besitzt zunächst folgende Bindegliedfunktion

- beim Update weiß der Benutzer, was an Daten einzugeben bzw. zu ändern ist
- der gewählte Report "kennt" die Bezeichner und greift über Zeiger auf die Daten-Items zu, um sie auf dem Schirm zu platzieren

Der erste Schritt zur Synthese beginnt also bei den Bezeichnern, die zweckmäßigerweise als String ausgelagert werden. Den Zugang zu den Texten bildet eine Liste B(...), welche die Angabe "wo" im Bezeichnerstring (BS) und "wie lang" enthält. In REPDATA ist das platzsparend durch eine Zahl (I) repräsentiert, nämlich als Vor- und Nachkommawert. Im zweiten Schritt wird die Verbindung Bezeichner zu realen Datenitem programmiert. Weiß man den Bezeichner, will man auch wissen, welches Item gemeint ist. Dazu dient ein Zahlenarray P(...), der - wie bereits erwähnt - sagt, welche laufende (Bezeichner-)Nummer auf welches Datenelement in den beiden Arrays TEXT und ZAHL zeigt. Nun benötigt man nur noch die Angabe, aus welchen Bezeichner-Nummern sich ein Report zusammensetzt. Die enthält der Report-Info-String RS und sein "Verwalter", der Array R(Reportnummer). Letzterer zeigt je Report auf den Anfang der Reportangaben in RS. Der Zusammenhang wird noch genauer programmtechnisch dargestellt werden; zunächst das prinzipielle Zusammenspiel "Daten" von und "Listen" im Schaubild.



Die Zuordnung von Daten zu Reports (Programm REPDATA)

#### Aufgabenstellung

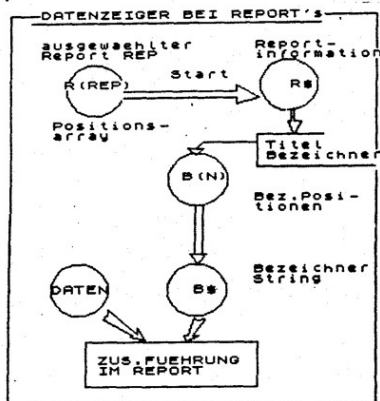
Ziel der vorgestellten Programmlösung ist es, eine (einfache)

Datenverwaltung mit vielfältiger Datendarstellung in Reports zu verbinden, realisieren. Den simplen Grundfunktionen im Datenbereich (sbn. DATA DIVISION) steht die für den User bequeme Handhabung selbstdefinierter Listen gegenüber. Hauptaufgabe war ein hohes Maß an Bedienerfreundlichkeit. Sie konkretisiert sich in folgenden Forderungen:

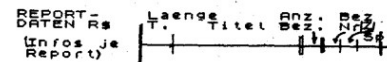
- freie Dimensionierung der Daten und Mischung von Zahlen- und Textfeldern (als "Record")
- unbeschränkte Vorgabe von Bezeichnern durch den Benutzer
- Entwurf beliebig vieler, verschiedener Reports
- freie Platzierung der Felder und Gestaltung der Reports
- Aufruf unterschiedlicher Reports beim Setzlesen incl. Reportwechsel innerhalb eines positionierten Satzes
- weitere Unterstützungsfunktionen (wie z.B. "Blättern")

#### Programmablauf

Realisiert wird die Zielsetzung in zwei "Abteilungen" - der DATA und der REPORT DIVISION. Beide erfüllen unterschiedliche Funktionen: die Datenabteilung managt die Sätze (Update, Löschen, Neuaufnahme), die Reportabteilung bringt die Daten in die gewählte Listform. Wie die DATA DIVISION Bezeichner verarbeitet und auf Satzitems zugreift wurde bereits gezeigt. Wie geht es weiter bei der Daten-Darstellung? Auch hier arbeitet REPDATA mit Zeigern als "Verbindungsglieder". Der gewählte Report REP greift auf die Reportinformationen zu, welche in RS liegen. Danach holt er sich seine Bezeichner, generiert das Layout und platziert die Daten auf den Screen. Der konkrete Zusammenhang wird durch das folgende Schaubild deutlicher.

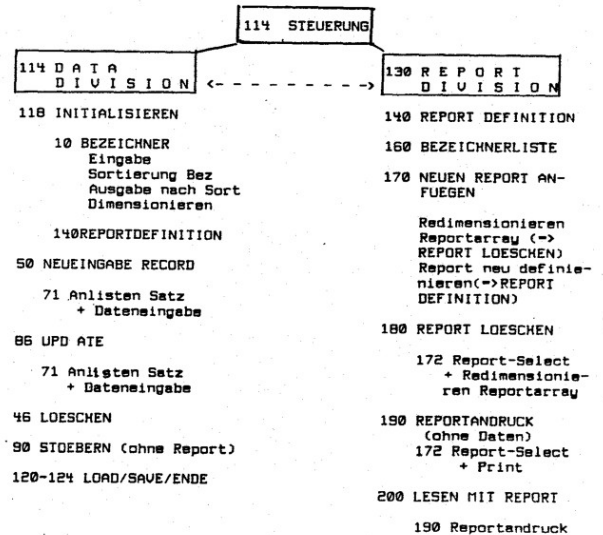


Interessant ist, welche Informationen genau zur Erzeugung eines Reports aus RS kommen. Dazu die Aufschlüsselung der Eintragungen im Reportstring, wobei Zahlenangaben dort natürlich als Codes stehen.



Es würde zu weit führen, die Bestandteile des Programms REPDATA bis in alle Einzelheiten zu erläutern. Wichtig fürs Verständnis ist jedoch der grobe funktionale Zusammenhang. Die Titel der Module besagen bereits, was in ihnen im wesentlichen geschieht (Mehr findet sich in der folgenden Programm-Dokumentation).

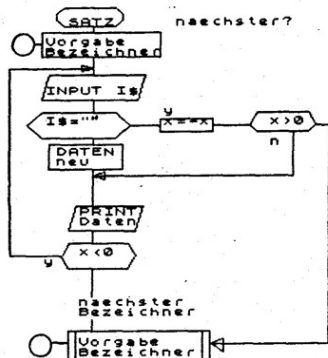
#### Funktionalzusammenhang Module



+ Auffüllen  
mit Daten

Auf einen Leckerbissen möchten wir noch eingehen - die Satzverarbeitung (Die Logik ist im extra dargestellt). Das Modul kann auch in eigenen Programmen eingesetzt werden; es ermöglicht nicht bloß die Eingabe von Daten, wobei Feldbezeichner vorgegeben werden. Vielmehr kann mit Korrekturen beim INPUT solange "gekreist" werden, bis die Eingabe ok ist. Erst mit der Leertaste quittiert der User sie endgültig. Gibt er zunächst nichts ein, erscheint ein ggf. vorhandener Feldinhalt. Er kann dann solange überschrieben werden, bis der Benutzer zufrieden ist und mit RETURN das Feld verläßt..

#### LOGIK SATZVERARBEITUNG (Zeile 70-85)



Auf weitere 'Logik' sei verzichtet. Ehe das Programm selber gebracht wird, sei sein "Output" anhand einiger Screencopies gezeigt. REPDATA präsentiert sich dem Benutzer wie folgt (s. Hardcopy-Folge):

Menu DATA DIVISION (gleichzeitig Hauptsteuerung).  
Menu REPORT DIVISION mit Rückprungoption  
Festlegen der Bezeichner (anfangs) + Liste  
Update mit Input-Korrektur (Bezeichner invers)  
Stöbern im File unter Cursor-Control ohne Report  
..oder mittels definierter Reports

```

>>> HAUPTMENU <<<
DATA DIVISION
1-INITIALISIEREN
2-NEUEINGABE
3-UPDATE
4-LOESCHEN
5-STOEERN
6-ABRUF
7-ENDE
8->REPORT DIVISION
  
```

```

>>> MENU <<<
REPORT DIVISION
1-REP.DEFINITION
2-BEZEICHNERLISTE
3-NEUEN REP.
4-REP. LOESCHEN
5-REP. ANDRUCK
6-LESEN MIT REP.
7->DATA DIVISION
  
```

```

LISTE DER BEZEICHNER
NR T E X T(==TEXTITEM)
1 *ANSCHRIFT
2 *NAME
3 ALTER
4 GEGALT
5 QUALIFIKATION
  
```

```

----- S A T Z 2 -----
ANSCHRIFT 14405 ANGELROGGE
NAME SCHULZE, GERO
ALTER 38
GEGALT 38945
QUALIFIKATION 3
  
```

```

A T Z 1
00000 KOELN 90
00000 IER, HANS
00000 9927
A T Z 2
14405 ANGELROGGE,
38 SCHULZE, GERO
38945
A T Z 3
14405 TELGDE
38 BAUMANN FRANZ
38945
00000 6709
  
```

4	PERSONAL DATEN
*NAMEJANSENSEN HANS	
*ANSCHRIFT5508 HEIMBACHES	
QUALIFIKATIONS	GEHALT39999
ALTER37	

4	GEHALTSLISTE
*NAMEJANSENSEN HANS	
GEHALT39999	

# Programmdokumentation

## Programmaufbau

Module (Zeilen)	Inhalt mit Unter/Aufrufmodulen	verwandte Variablen
<DATA DIVISION>		
INITIALISIEREN (10-44,140-153)a)INITIALISIEREN DATEN		
	BEZEICHNER festlegen (10-24) -Festlegen Anzahl DATA-Items -Eingabe Bezeichnungen -Aufbau Bezeichnerstring (BS) und Positionsarray (B(...))	IS BS P(BEND) B(BEND) ZAHL,TEXT X,Y,N
	SORTIERUNG Bezeichnerliste (28-34) Austauschsort unter Verwendung der def. Funktionen FN B(X),FN G (X) Memo:keine Änderung der Bezeichner (BS) - nur der Positionierungen in B(N)	BEND M,N BS, B(...) X0,X
	AUSGABE der Bezeichner nach Sort(36-39) -Fortzahlen Text- u. Zahlen-Items und Ablage im Zugriffsarray P(...) -Ausgabe Liste -Copy-Abfrage	BS,BEND N TEXT,ZAHL P(...)
	b)DIMENSIONIERUNG der Daten (42-44) -Eingabe Anzahl Records -Reservieren Text- u. Zahlenarrays	D(REC, ZAHL) DS(REC, TEXT,20)
LOESCHEN Record (46-48)	-Wahl d. Rec.Nr	R,N

124

	-Löschen Zahl/Textfelder	D(R,N) DS(R,N) ZAHL,TEXT
NEUEINGABE RECORD (50-68)	-Aufsuchen nächster leerer Satz -ANLISTEN SATZ und Dateneingabe	N,Y,R,REC ZAHL,TEXT DS(...,D(...))
ANLISTEN SATZ (71-84)	(ohne Report) Satzverarbeitung für Lesen,Update + Neueingabe -Vorgabe Bezeichner -Andruck vorh. Daten mit Typenselektion -fakultative Dateneingabe	BS,IS DS(...) D(...) P(N),X,N
UPDATE (85-88)	-Recordanwahl -Ändern der Daten im Untermodul ANLISTEN SATZ	IS R,REC
BROWSE (STOEßERN ohne Report) (90-100)	-Festlegen Einsprungstelle im File -Check auf "leer" (kein Andruck) -Datenprint gem Item-Liste P(...) -Cursorsteuerung fürs Blättern	IS R,REC ZAHL,TEXT N,BEND P(N) D(...),DS(...)

## STEUERUNG Hauptmenu <DATA DIVISION>

(110-129)	-Display Menu -Abfrage Useroption und Unterprogramm- aufrufe -Direktbearbeitung von LOAD/SAVE/Sprung in die REPORT DIVISION	IS
<REPORT DIVISION> MENUE (130-139)		
	-Display Report-Menu -Abfrage Useroption + Unterprogramm- aufrufe -ggf. Sprung zur DATA DIVISION	IS
REPORTDEFINITION (140-153)		
	-Dimensionieren Positionsarray R(...) d.h. Anzahl Reports gemäß User -Definieren Report mit Titel,Bezeich- nern -Platzieren Bezeichner auf dem 'Layout' unter Cursorsteuerung -Sichern Reportinfos in RS incl. Posi- tionierungen (Zeile,Spalte) -Setzen Zeiger in R(...) auf Reportan- fang (RS)	REND R(REND) N,M B(...) Y0,Y1 BS X,RS,ZS

125

BEZEICHNER (160-164) aktuelle Liste für Reportdefinition  
-Andruck  
-Copy-Anfrage

NEUEN REPORT ANFÜGEN (170-171)  
-Redimensionieren Reportzeiger-Array R()  
-Unteraufruf v. REPORT.LOESCHEN, um Reportarray R() neu zu setzen  
-Reportstring RS mit neuem Report füllen + R-Array aktualisieren (Unteraufruf REPORTDEFINITION)

REPORT-SELECT (173-177) Auswahl eines Reports auf der Input-  
leiste (Titelanzeige mit Cursorcontrol)

REPORT LOESCHEN (180-183)  
-Unteraufruf von REPORT-SELECT  
-Auswahl der betreffenden Reportinfos (in RS)  
-Stringverkürzung (von RS) und Redimensionieren Positionsarray R(N)

REPORTANDRUCK (190-199)  
-Unteraufruf REPORT-SELECT  
-Auslesen Item-Infos (aus RS)  
-Generieren Report als Leerdruk bzw. bei vorhandenen Daten Selektion von Zahlen- oder Textdaten

REPORTLESEN (analog zu SIOBERN in der 'DATA DIVISION') (202-212)  
-Einsprungsstelle im File festlegen  
-Unteraufruf REPORT-SELECT und REPORTANDRUCK (mit Daten)  
-Cursorsteuerung für satzweises Lesen bzw. Auswahl eines anderen Reports (Useroption 'R')

#### Variablenliste

1. Datenarrays  
DS(REC,N) TEXT-Daten (Satz,Item)  
D(REC,N) ZAHLEN-Daten (Satz,Item)

2. REPORT-Daten  
BS Bezeichnertexte  
RS Reportinformationen (Titel, Anzahl Bezeichner, BezeichnerNr, Zeilen/Spaltenposition)

3. Datenzeiger  
P(BEND) Verknüpfung BezeichnerNr zum Subscript  
Datenarray (Negativwert=Textarray, Posi-

tivwert=Zahlenarray)  
B(BEND) Position der Bezeichner in BS (Vorkomma-  
wert=Stelle, Nachkommawert=Länge)  
R(REND) Start der Reportinfos in RS je Report

4. Begrenzer  
BEND Anzahl Bezeichner  
ZAHL Anzahl Zahlenitems (pro Record)  
TEXT Anzahl Textitems (pro Record)  
REND definierte Reportzahl  
REC Maximalzahl Sätze (reservierter Datenbereich)

5. aktuelle Daten  
IS Userinput  
R bearbeiteter Record  
REP gewählter Report

6. sonstige Variablen  
ZS Löschesymbol bei REPORTDEFINITION  
X0,X1 Zwischengespeicherte Positionierungen  
Y0,Y1 Laufvariablen  
N,M Zahl- und Abfragevariablen  
X,Y

7. Funktionen  
FN B(X) ..liefert den Vorkommawert im Positionierungsarray 'Bezeichner'(B(...))  
=> Start Bezeichner in BS  
FN G(X) ..liefert den Nachkommawert von B(...)  
plus Bezeichnerlänge  
=> Ende Bezeichner in BS

#### Benutzungshinweise für REPDATA

- + Bei Ja/Nein-Abfragen des Programms muß - wie angezeigt - mit Y bzw. N geantwortet werden (sonst Systemstop !)
- + Textbezeichner signalisieren ihre Funktion mit "": alle Textitems sind standardmäßig mit 20 Stellen Maximum angelegt (wer mehr will, kann Zeile 44 ändern bzw. eine Inputanfrage einbauen)
- + für die Reportdefinition müssen Bezeichner mit Nr. angegeben werden; sie werden in sortierter Folge beim Initialisieren oder später auf Wunsch ausgegeben (Kopie empfehlenswert). Wenn die alphabetische Folge stört (z.B. beim Update), kann sie durch Bezeichnerinput mit führendem Space beeinflussen. Dann erscheint fortan " -" (Useroption vor "Titel" etc.)
- + Da die Routine "Anlisten Satz" bei Neueingabe, Update und Nur-Lesen (ohne Ändern) benutzt wird, ist folgendes Prozedere zu erwarten: bei Ersteingabe - wenn sofort Dateninput, keine Korrekturmöglichkeit  
- letztere mit Trick: Leerinput, darauf "Leerranzige"; danach Korrekturen machbar  
bei Update - vorhandene Daten können mit Leer-

input sichtbar gemacht werden; da-  
nach kann überschrieben werden  
oder man übergeht das Item  
- nach Korrektur von Daten ist es  
möglich, beliebig lange zu "krei-  
sen", bis die endgültige Eingabe  
steht

+ Das STOEßERN im File mit oder ohne Report erfolgt durch Cur-  
sorbewegung (Pfeiltasten ohne Shift); es gibt für den User 2  
Arten der CursorControl - satzweises Blättern (Record vor -  
"B", zurück - "S") und Wahl des Reports mit den Up- und Down-  
Tasten ("7" bzw. "8"). Um beim Lesen den Report zu wechseln,  
wird - wie angezeigt - Option "R" gewählt, worauf man anhand  
der Titelanzeige die Reports durchgeht. Quittiert wird im  
Übrigen mit "Space"...

```

0400 REM REP-TEST
0401 TO 100
0402 IF FN B(N) = INT B(X)
0403 THEN FN B(N) = 100*(B(X) - FN B(X))
0404 REM REP-TEST
0405 REM BEZEICHNER
0406 REM BEZEICHNER DATAITEMS=BE
0407 REM BEZEICHNER: BEND
0408 REM BEZEICHNER: BEND
0409 REM BEZEICHNER: BEND
0410 REM BEZEICHNER: BEND
0411 REM BEZEICHNER: BEND
0412 REM BEZEICHNER: BEND
0413 REM BEZEICHNER: BEND
0414 REM BEZEICHNER: BEND
0415 REM BEZEICHNER: BEND
0416 REM BEZEICHNER: BEND
0417 REM BEZEICHNER: BEND
0418 REM BEZEICHNER: BEND
0419 REM BEZEICHNER: BEND
0420 REM BEZEICHNER: BEND
0421 REM BEZEICHNER: BEND
0422 REM BEZEICHNER: BEND
0423 REM BEZEICHNER: BEND
0424 REM BEZEICHNER: BEND
0425 REM BEZEICHNER: BEND
0426 REM BEZEICHNER: BEND
0427 REM BEZEICHNER: BEND
0428 REM BEZEICHNER: BEND
0429 REM BEZEICHNER: BEND
0430 REM BEZEICHNER: BEND
0431 REM BEZEICHNER: BEND
0432 REM BEZEICHNER: BEND
0433 REM BEZEICHNER: BEND
0434 REM BEZEICHNER: BEND
0435 REM BEZEICHNER: BEND
0436 REM BEZEICHNER: BEND
0437 REM BEZEICHNER: BEND
0438 REM BEZEICHNER: BEND
0439 REM BEZEICHNER: BEND
0440 REM BEZEICHNER: BEND
0441 REM BEZEICHNER: BEND
0442 REM BEZEICHNER: BEND
0443 REM BEZEICHNER: BEND
0444 REM BEZEICHNER: BEND
0445 REM BEZEICHNER: BEND
0446 REM BEZEICHNER: BEND
0447 REM BEZEICHNER: BEND
0448 REM BEZEICHNER: BEND
0449 REM BEZEICHNER: BEND
0450 REM BEZEICHNER: BEND
0451 REM BEZEICHNER: BEND
0452 REM BEZEICHNER: BEND
0453 REM BEZEICHNER: BEND
0454 REM BEZEICHNER: BEND
0455 REM BEZEICHNER: BEND
0456 REM BEZEICHNER: BEND
0457 REM BEZEICHNER: BEND
0458 REM BEZEICHNER: BEND
0459 REM BEZEICHNER: BEND
0460 REM BEZEICHNER: BEND
0461 REM BEZEICHNER: BEND
0462 REM BEZEICHNER: BEND
0463 REM BEZEICHNER: BEND
0464 REM BEZEICHNER: BEND
0465 REM BEZEICHNER: BEND
0466 REM BEZEICHNER: BEND
0467 REM BEZEICHNER: BEND
0468 REM BEZEICHNER: BEND
0469 REM BEZEICHNER: BEND
0470 REM BEZEICHNER: BEND
0471 REM BEZEICHNER: BEND
0472 REM BEZEICHNER: BEND
0473 REM BEZEICHNER: BEND
0474 REM BEZEICHNER: BEND
0475 REM BEZEICHNER: BEND
0476 REM BEZEICHNER: BEND
0477 REM BEZEICHNER: BEND
0478 REM BEZEICHNER: BEND
0479 REM BEZEICHNER: BEND
0480 REM BEZEICHNER: BEND
0481 REM BEZEICHNER: BEND
0482 REM BEZEICHNER: BEND
0483 REM BEZEICHNER: BEND
0484 REM BEZEICHNER: BEND
0485 REM BEZEICHNER: BEND
0486 REM BEZEICHNER: BEND
0487 REM BEZEICHNER: BEND
0488 REM BEZEICHNER: BEND
0489 REM BEZEICHNER: BEND
0490 REM BEZEICHNER: BEND
0491 REM BEZEICHNER: BEND
0492 REM BEZEICHNER: BEND
0493 REM BEZEICHNER: BEND
0494 REM BEZEICHNER: BEND
0495 REM BEZEICHNER: BEND
0496 REM BEZEICHNER: BEND
0497 REM BEZEICHNER: BEND
0498 REM BEZEICHNER: BEND
0499 REM BEZEICHNER: BEND
0500 REM BEZEICHNER: BEND

```

```

34 REM AUSGABE SORT .BEZEICHNE
35 REM SORTIERTE BEZEICHNER
36 REM SORTIERTE BEZEICHNER
37 REM SORTIERTE BEZEICHNER
38 REM SORTIERTE BEZEICHNER
39 REM SORTIERTE BEZEICHNER
40 REM SORTIERTE BEZEICHNER
41 REM SORTIERTE BEZEICHNER
42 REM SORTIERTE BEZEICHNER
43 REM SORTIERTE BEZEICHNER
44 REM SORTIERTE BEZEICHNER
45 REM SORTIERTE BEZEICHNER
46 REM SORTIERTE BEZEICHNER
47 REM SORTIERTE BEZEICHNER
48 REM SORTIERTE BEZEICHNER
49 REM SORTIERTE BEZEICHNER
50 REM SORTIERTE BEZEICHNER
51 REM SORTIERTE BEZEICHNER
52 REM SORTIERTE BEZEICHNER
53 REM SORTIERTE BEZEICHNER
54 REM SORTIERTE BEZEICHNER
55 REM SORTIERTE BEZEICHNER
56 REM SORTIERTE BEZEICHNER
57 REM SORTIERTE BEZEICHNER
58 REM SORTIERTE BEZEICHNER
59 REM SORTIERTE BEZEICHNER
60 REM SORTIERTE BEZEICHNER
61 REM SORTIERTE BEZEICHNER
62 REM SORTIERTE BEZEICHNER
63 REM SORTIERTE BEZEICHNER
64 REM SORTIERTE BEZEICHNER
65 REM SORTIERTE BEZEICHNER
66 REM SORTIERTE BEZEICHNER
67 REM SORTIERTE BEZEICHNER
68 REM SORTIERTE BEZEICHNER
69 REM SORTIERTE BEZEICHNER
70 REM SORTIERTE BEZEICHNER
71 REM SORTIERTE BEZEICHNER
72 REM SORTIERTE BEZEICHNER
73 REM SORTIERTE BEZEICHNER
74 REM SORTIERTE BEZEICHNER
75 REM SORTIERTE BEZEICHNER
76 REM SORTIERTE BEZEICHNER
77 REM SORTIERTE BEZEICHNER
78 REM SORTIERTE BEZEICHNER
79 REM SORTIERTE BEZEICHNER
80 REM SORTIERTE BEZEICHNER
81 REM SORTIERTE BEZEICHNER
82 REM SORTIERTE BEZEICHNER
83 REM SORTIERTE BEZEICHNER
84 REM SORTIERTE BEZEICHNER
85 REM SORTIERTE BEZEICHNER
86 REM SORTIERTE BEZEICHNER
87 REM SORTIERTE BEZEICHNER
88 REM SORTIERTE BEZEICHNER
89 REM SORTIERTE BEZEICHNER
90 REM SORTIERTE BEZEICHNER
91 REM SORTIERTE BEZEICHNER
92 REM SORTIERTE BEZEICHNER
93 REM SORTIERTE BEZEICHNER
94 REM SORTIERTE BEZEICHNER
95 REM SORTIERTE BEZEICHNER
96 REM SORTIERTE BEZEICHNER
97 REM SORTIERTE BEZEICHNER
98 REM SORTIERTE BEZEICHNER
99 REM SORTIERTE BEZEICHNER
100 REM SORTIERTE BEZEICHNER

```



## Sortierverfahren

Aus der Fülle der Möglichkeiten

..füllt es schwer, etwas Geeignetes auszuwählen. Da ist die Rede von Sortierung durch Zerlegen, Mischen, da wird von einzusetzenden Blumen, Listen, Kaufen (engl. heap) geredet, da schwirren Fachtermini wie Binary- und Maximum-Sort herum. Und keiner weiß, was im konkreten Fall zu tun ist - möchte man hinzufügen! Immerhin hat spätestens mit dem Heimcomputer die "Sorttheorie" das Feld wissenschaftlicher Abhandlungen verlassen. Viele wissen inzwischen "Gut sortiert ist halb gefunden" und halten nach geeigneten Verfahren Ausschau. Die Theorie hat aus dem Zirkel spezialisierter Computermagier Eingang in die Heimprogrammierung gefunden. Sogar populäre MC-Magazine veröffentlichen Beiträge zum Thema "Sortieren" (Einen recht guten Überblick bringt etwa das "P.M. Computerheft" 8/85 - Seite 88).

Sortierungen sind unter zwei Voraussetzungen erforderlich

1. man möchte im Datenbestand schnell etwas finden
2. durch die Eingabe ist die gewünschte Ordnung ist nicht vorgegeben

Ohne auf Feinheiten einzugehen, bei Sortierverfahren stellt sich folgendes Dilemma ein: entweder ist der Algorithmus kompliziert und erfordert (mit zusätzlichen Listen etc.) extra Speicherplatz oder das Verfahren ist einfach, aber langsam.

Man mag sagen "Gut, ich nehme den einfachen Bubble-Sort und warte ab". Wer nach jeder Neuingabe/Update Zigarettenpause macht, schadet nicht nur seiner Gesundheit, sondern verliert auch die Lust an ernsthaftem Computern. Sind gar Mehrfachsorts erforderlich (z.B. Schallplatten nach Interpret, Komponist und Genre), ist Bubble und Co schnell am Ende. Sgn. effekte Verfahren haben andererseits auch Nachteile. Meist muß ein "Baum" oder eine ähnliche Struktur zu Hilfe genommen werden. Versuchen wir einen Mittelweg, wobei wir kurz die grundsätzlichen Maßgrößen mit ins Spiel bringen.

Die Effektivität von Sortverfahren basiert auf Zahl der Austausch- bzw. Vergleichsoperationen plus notwendigem Zusatzspeicher. Austausch und Vergleich sind sich ersetzende Größen. Schnellere Sorts brauchen meist Zusatzfelder, viele Variablen und beträchtlichen Programmaufwand. Da sich "interpretiertes" BASIC dazu schlecht eignet, stellen viele trickreiche Verfahren sich selber ein Bein. Für BASIC beherrzige man die Devise "small is beautiful".

Wir wollen uns einigen wenigen Sorts widmen, die kurz programmiert sind, ausreichend 'Speed' und wenig Speicheraufwand haben. Fangen wir mit dem Einfachsten an und zeigen schrittweise

Wege der Verbesserung.

## Einfach aber langsam - der BUBBLE-Sort

Der Bubble (Blasen)-Sort hat seinen Namen, weil zwei Folgeelemente miteinander verglichen und bei "größer" vertauscht werden, sodaß das größte wie eine Luftblase zur "Oberfläche" (= Ende der Datenreihe) steigt. Er ist allgemein am verbreitetsten. Die Routine dazu liegt in einem Testrahmen, um Verbesserungen aufzuzeigen. Sie basiert auf folgenden Variablen:

TS = ursprüngliche 36 Testzeichen    E = Anzahl zu sortierender Elemente (36)  
AS = nachher sortierter String    N = äußere Schleifenvariable  
CS = Austauschfeld    M = innere Schleifenvariable

Weg bei BUBBLE: Vergleich der Nachbarelemente bis zum vorletzten Element; Austausch, wenn der "innere" Vergleich "größer" signalisiert. Nachteil - Es wird viel verglichen und ausgetauscht, daher langsam.

```

10 REM BUBBLE SORT
12 LET TS="00110101010101010101010101010101"
13 SPOS=78:XDURKUJ
14 LET E=LEN AS
15
16 REM BUBBLE EINFACH
17
18 LET AS=TS
19 FOR N=1 TO E-1
20   FOR M=1 TO E-N
21     IF AS(M)>AS(M+1) THEN LET
22       CS=AS(M)
23       LET AS(M)=AS(M+1)
24       LET AS(M+1)=CS
25   NEXT M
26 NEXT N
27 PRINT AS
28 REM 16 SEC
29
30 REM BUBBLE SCHNELLER
31
32 LET AS=TS
33 FOR N=1 TO E
34   FOR M=2 TO N+1 STEP -1
35     IF AS(M)>AS(N) THEN LET C
36       =AS(M)
37       LET AS(M)=AS(N)
38       LET AS(N)=C
39   NEXT M
40 NEXT N
41 PRINT AS
42 REM 13 SEC
43
44 REM BUBBLE OPTIMIERT
45
46 LET AS=TS
47 FOR N=1 TO E-1
48   FOR M=2 TO N+1 STEP -1
49     IF AS(M)>AS(N) THEN LET C
50       =AS(M)
51       LET AS(M)=AS(N)
52       LET AS(N)=C
53   NEXT M
54 NEXT N
55 PRINT AS
56 REM 12 SEC

```

Wir machen nun die erste Verbesserung, die meßbare Zeitersparnis bringt. Einmal drehen wir die Richtung im Inneren (mit negativem Step) von "oben nach unten" um. Zum anderen holen wir noch etwas Laufzeit mit Wegfall der Rechenoperationen M+1 heraus.

Die zweite Verbesserung geht diesem Weg weiter. Der letzte Durchgang wird ausgespart; weiter erhöht Zusammenpacken zu nur zwei Programmzeilen die Effektivität. Bei Inline-Statements hat der Interpreter weniger zu tun (Zeilennummer suchen etc.). Übrigens ist ein Tip, um Programme schneller zu machen..

#### Der effektive SHELL-Sort

Der SHELL-Sort ist das erste effektive Verfahren. Es tauscht und vergleicht aufgrund bisheriger Ergebnisse. Der Trick besteht in einer "Distanzsteuerung": Es wird nicht linear hochgezählt, sondern "gezielt" fortgeschritten und ausgetauscht. Kein Wunder also, wenn die BUBBLE-Zeit locker halbiert wird. SHELL bleibt unkompliziert und einfach zu programmieren - ohne viel Speicheraufwand. Das Verfahren ist für den Hausgebrauch wohl das beste, universell einsetzbare.

Wie arbeitet der SHELL-Sort? Die Sortspanne (= Bereich der zu vergleichenden Elemente) verkleinert sich und wandert je nach Vergleichsergebnis. Dazu werden zwei Schleifen bemüht: Die äußere gibt den Gesamtbegrenzer ab, die innere legt die "Halbierungsdistanz" auf den aktuellen Zähler (K). War das laufende Element (mit K indiziert) größer als das Vergleichsargument (mit J indiziert), erfolgt Austausch. Aber im Unterschied zu BUBBLE läuft die Positionierung anders. Die Spanne wird verkleinert, das laufende Element (K) wird nach "oben" gesetzt. Die wichtigste Rolle spielt dazu die "Halbierungsvariable" M; Sortende ist erreicht, wenn es kein Intervall mehr gibt, d.h. M = 0. (Wer es genau wissen will, ist aufgefordert, sich einmal die Logik aufzumalen und durchzugehen).

Die Variablen bei sonst unverändertem "Testrahmen" sind:

E	=	Arraygröße	J	=	lfd. Vergleichsargument (Subskript)
M	=	Halbierungsvariable			
N	=	Begrenzer innere Schleife	K	=	lfd. Anfangsargument (Subskript)
A	=	lfd. Element innere Schleife			

Bekanntlich gibt es nichts, was sich nicht zu verbessern ist. Wir haben laufzeitsteigernde Maßnahmen getroffen, ohne die Logik zu verändern. - Zuerst wurde das Hochzählen von J (mit 'Aufsetzen' auf A) in eine schnelle FOR-NEXT-Schleife gebracht, womit der Interpreter besser zurecht kommt. Die M-Subtraktion erscheint als negative STEP-Variable. Weiter kann der Ja-Zweig beim "Kleiner"-Vergleich in ein Statement (eine Zeile !) zusammengefaßt werden (Für Kenner - die NEXT J - Adresse landet nicht auf dem Stack und baut das Memory zu; vielmehr initialisiert das System jeden FOR-Start neu, Vorsicht bei Übertragung auf andere Dialekte, nicht alle MC's sind so gutmütig wie SINCLAIR...). Und noch eine Schallermach-Trick: Gemäß der Regel "Multiplizieren geht leichter als Dividieren"

wurde M mit \* .5 halbiert. Immerhin bringt das alles nicht viel an Geschwindigkeit, zeigt aber Wirkung. - Nun aber zum Hochgeschwindigkeitssort...

```

10 REM SHELLSORT
12
14 LET T$="INQYTFUC7DER2B553X
  ZRQW99010P0LM4KJHU"
16 LET A$=T$
18 LET E=LEN A$
20 LET M=E
22 LET M=INT (M/2)
24 IF NOT M THEN PRINT A$
  STOP
26 FOR J=A TO N STEP -M
  LET U=A
  LET K=M+J
  IF A$(K)<A$(J) THEN LET C$=
    A$(J):LET A$(J)=A$(K):
    LET A$(K)=C$
  LET J=J-M
  IF J=0 THEN GO TO 26
28 NEXT J
29 DO T=9.82
  REM T=9.82 SEC
30 REM ...VERBESSERT
31
32 LET A$=T$
34 LET M=E
  LET M=INT (M*.5)
  IF NOT M THEN PRINT A$
  STOP
36 LET N=E-M
  FOR J=A TO N
    LET K=M+J
    IF A$(K)<A$(J) THEN LET C$=
      A$(J):LET A$(J)=A$(K):
      LET A$(K)=C$
    LET J=J-M
  NEXT J
38 NEXT A
39 GO TO 34
  REM 7,3 SEC

```

#### Ganz schnell - der QUICKSORT

Der QUICKSORT (fortan QS) ist die anerkannt schnellste Methode für ungeordnete Daten. Sie wurde vom Amerikaner HOARE vor Jahren entwickelt und läßt sich auch beim Heimcomputer nutzen. Allerdings benötigt QS etwas Aufwand und einen zusätzlichen Zahlenarray. Man sollte den Sort nur im Fall umfangreicher Daten anwenden (.. oder gleich MC-Routinen bemühen). Wir haben das Verfahren in einen Demonstrationsrahmen gepackt. Teststring AS ist auf 30 Zeichen für SINCLAIR-Zeilendruck ausgerichtet worden. So stehen die Tauschfelder gut vergleichbar untereinander. Der Interessierte kann verfolgen, wie QS im einzelnen arbeitet. Für praktische Einsätze ist der PRINT (in Zeile 164) zu entfernen und eine Variablenanpassung vorzunehmen.

Man erwarte keine "Logik", vielmehr folgen die erforderlichen

```

12 10 REM QUICKSORT-DEMO
14 REM TESTSTRING
15 LET A$=""
16 FOR N=1 TO E
17   LET A$=A$+CHR$(40+INT(RND
18   ) * 26))
19 NEXT N
20 PRINT A$
21 DIM B$(E,2)
22 LET B$(1,2)=1
23
24 REM SORTIERUNG
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52 LET L=B$(P,1)
53 LET R=B$(Q,2)
54 LET D=P-1
55 LET I=P
56 LET A$=A$(INT((L+R)*.5))
57 IF A$(I) < A$ THEN LET I=I+1
58 IF B$(J,2) < A$ THEN LET J=J-1
59 GO TO 155
60 IF I < J THEN LET T=A$(I)
61 LET A$(I)=A$(J)
62 LET A$(J)=T
63 LET J=J-1
64 LET I=I+1
65 IF I < J THEN GO TO 155
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164 IF I < R THEN LET P=P+1
165 LET B$(P,1)=I
166 LET B$(P,2)=R
167 PRINT A$
168 IF L < R THEN GO TO 154
169 IF L > R THEN GO TO 152
170 PRINT A$

```

[illegible]

Numerkreise sind aber selten geschlossen, von Textschlüsseln ganz zu schweigen. Wer wird schon für Umsatzzahlen der Jahre 1970 - 1985 fast 2000 Elemente definieren? Lückenlose Belegung der 'range' mit 16 Jahreszahlen geht von Differenzen zu einer Basis aus, um direkt zu adressieren. Bleiben wir beim Beispiel.

Daten des Jahres 1985 etwa werden so direkt abgefragt, wenn die Umsätze in U(16 Elemente) liegen:

```
LET BASIS = 1969
INPUT "JAHR ?";J
PRINT "UMSATZ ";J;" = ";U(J-BASIS)
```

Bei nicht diskreter Verteilung (oder Text) fällt die Zuordnung Item - Schlüssel schwer. Es gibt aber Verfahren, die weiterhelfen (HASH). Auf sie soll wegen ihrer Komplexität nicht weiter eingegangen werden. Man kommt auch auf einem anderen Weg zum Ziel. Nach Sortierung der Daten wird die Reihenfolge ausgelagert; so hat man auch zu ein und demselben Datenbestand verschiedene Sortfolgen parat. Die betreffende Ordnung wird als Zeiger auf die Daten gespeichert. Dadurch kann direkt (per Subskript) auf einzelne Items zugegriffen werden. Womit sich das Problem 'Wiederfinden' auf die Suchprozedur verlagert. Wieso? Um festzustellen, ob Suchargument gleich Dataitem ist, wird via Datenzeiger verglichen. Danach geht es zurück zum Zeigerarray, um erneut Item mit Argument auf gleich zu prüfen usw. Von einem "Direktschluß" Argument - Index kann nicht mehr die Rede sein. Generell gilt - die Suchprozedur ist abhängig von der Datenanordnung. Liegt keine solche vor, bleibt nur 'lineares' Durchprüfen aller Daten. Ist hingegen eine Sortierung vorhanden, lassen sich schnellere Verfahren anwenden.

#### Linear contra binär

'Linear' heißt sukzessives Durchsuchen der Daten. Da im Schnitt das halbe File durchgegangen wird, ist die Linearsuche das umständlichste Verfahren. Es ist lediglich gerechtfertigt, wenn keine Ordnung (sprich Sortierung) vorliegt oder eine kleine Folgen vorliegt (z.B. Anzeigetexte). Bei sortierten Reihen ist Linearsuche Zeitverschwendung. Die effektivere Art nennt sich Binärsuche. Sie bedeutet im Wesentlichen folgendes:

- + Aufteilen der Suchstrecke in halbe Distanzen,
- + Aufsetzen auf der zuaddierten Distanz ('oben' bzw. 'unten') abhängig vom Vergleichsergebnis,
- + das Ganze solange, bis Argument gefunden oder Distanz Null ('nicht gefunden')

Da sich die Distanzen schnell verkleinern, kommt das Verfahren mit wenigen Vergleichen aus. Bei 100 Datenelementen sind maximal 7 Steps erforderlich; die Linearsuche braucht im Schnitt 50 Suchschritte. Um z.B. Element NR 99 zu finden, läuft die Binärsuche so ab:

STEP	1	2	3	4	5	6	7
Index	100	50	75	88	95	99	
Ergebnis	<	>	>	>	>	=	
des Ver-							

gleiche					
Offset	-50	+25	+13	+7	+4
(INT-Run- dung)					found

Zer Potenzen spielen (indirekt) eine wichtige Rolle. Mit ihrer Hilfe kann man sich bei gegebener Elementzahl ausrechnen, wieviel Steps anfallen. In der Mehrzahl der Fälle braucht man weniger als die Maximalstapzahl. Der Durchschnitt liegt unter der Hälfte davon, falls die Sortmenge nicht gerade gleich der nächsten Zer Potenz ist. Folgende Hilfetabelle gibt darüber Auskunft.

Zer	0	1	2	3	4	5	6	7	8	9
Potenz	1	2	4	8	16	32	64	128	256	512
Elemente	1	3	7	15	31	63	127	255	511	1023
max.Steps	1	2	3	4	5	6	7	8	9	10
Durchschnitt	1	1	1,5	2	2,5	3	3,5	4	4,5	5

Die Binärsuche ist das schnellste Suchverfahren für den Hausgebrauch; einzige Voraussetzung - eine sortierte Itemfolge. Natürlich gibt es weitere Verfahren, für Heimcomputeristen dürfte das folgende noch von Interesse sein.

#### Blocksuche

Die Blocksuche oder ein 'indexsequentielles' Verfahren steht zwischen 'linear' und Direktzugriff. Es läuft prinzipiell so: Aus dem Argument wird ein Indexwert gewonnen, z.B. bei Text der Code des ersten Zeichens. Mit Index greift man auf eine Blockadresse direkt zu. Mittels Adressarray erfährt man den Start der Dataitems, die zum Block gehören. Zum Beispiel bei Namensuche den Anfang aller Daten mit "M". Insgesamt 26 Adressen zeigen, wo "M" startet und "N" beginnt. Der Buchstabenblock kann nun im File linear durchsucht werden, bis "Name" gefunden oder "N" erreicht ist. Man kommt u.U. bei gleichstarker Besetzung der Blöcke mit weniger Steps als "binär" zurecht.

Für alle Suchverfahren folgen entsprechende Programmroutinen. Sie wurden zum Paket geschnürt. "Material" ist der sortierte Stringarray AS, der von den Sorten bekannt ist. Er muß vorhanden bzw. eingelesen sein. Die übrigen Variablen seien kurz erklärt, ehe das Programmangebot folgt:

Allgemein: AS (E,S) = sortierter Datenarray  
E = Elementzahl (hier 100)  
S = Suchargument (Settellig)

Binärsuche: U = Untergrenze Suchspanne  
O = Obergrenze Suchspanne  
M = Mitte (jeweils Hälfte von U+O)

Indexseq. Suche:  
 CS = Index f. Buchstabengrenze  
 (Initialisierung s. Zeile 36-43)  
 M = Code 1. Buchstabe Suchargument  
 (als Subskript verwendet)  
 O = Obergrenze Buchstabenverteilung  
 U = Untergrenze Buchstabenverteilung

Linearsuche: M = Laufvariable

```

10 REM SUCHVERFAHREN
11
12
13
14
15
16 REM LADEN SORTSTRING A$
17 LOAD "M";2;"A$STRING" DATA
18 A$(1)
19 LET E=100
20 INPUT "SUCHWORT (5 STELLEN) "
21 B$
22 IF B$="" THEN STOP
23
24 REM BINÄRSUCHE
25
26 LET U=0
27 LET O=E+1
28 IF B$ > A$(E) OR B$ < A$(1) THEN
29   GO TO 100
30
31 LET M=INT ((O+U)*.5+.5)
32 IF B$ < A$(M) THEN GO TO 120
33 IF B$ > A$(M) THEN LET O=M
34 GO TO 30
35
36 LET U=M
37 IF (O-U) < 1 THEN GO TO 24
38
39 REM INDEXSEQ-SUCHE
40
41
42 REM EINRICHTEN INDEX
43 DIM C$(26)
44 FOR N=1 TO 26
45   LET C$(N)=CHR$(N)
46 NEXT N
47
48 FOR M=1 TO E-1
49   IF A$(M,1) < A$(M+1,1) THEN
50     LET C$(CODE A$(M,1)-64)=C$(CODE A$(M+1,1)-64)
51     LET C$(CODE A$(M+1,1)-64)=C$(M)
52   END IF
53 NEXT M
54
55 LET CS=CODE A$(M,1)-64
56
57 REM SUCHE
58 INPUT "SUCHBEGRIFF (5 ST.) ";
59 B$
60 IF B$="" THEN STOP
61 LET M=CODE B$(1)-64
62 IF NOT CODE C$(M) THEN GO TO 120
63 LET O=CODE C$(M)
64 IF M > 1 THEN LET U=CODE C$(M-1)
65 IF NOT U THEN LET M=M-1
66 GO TO 50
67
68 FOR M=U TO O
69   IF A$(M)=B$ THEN GO TO 120
70 NEXT M

```

```

55 REM Z.VERGLEICH-
56 INPUT "SUCHBEGRIFF (5 ST.) ";
57 B$
58 IF B$="" THEN STOP
59 FOR M=1 TO E
60   IF A$(M)=B$ THEN GO TO 120
61 NEXT M
62 PRINT B$;" NOT FOUND"
63 STOP
64 PRINT B$;" FOUND AT NO ";M
65 GO TO 45

```

## FORMEN DER SPEICHERUNG

### Der universelle Datenzeiger

#### Speicherungsgrenzen

Gut 40 K RAM stehende dem Sinclair-Programmierer zur Verfügung; sie lassen den Glauben an nahezu unbegrenzte Datenspeicherung aufkommen. Abgesehen vom Programm gibt es jedoch natürliche Grenzen bei der Speicherung. Trickreiche "Verpointierung" kann sie mildern, nicht jedoch aufheben. Hierzu ein paar grundsätzliche Gedanken, ehe wir zu Datenzeigern und Sonderformen der Speicherung übergehen.

Zur Textorganisation wurde im vorigen Kapitel bereits etwas gesagt; wie sieht es mit sparsamer Zahlenspeicherung aus? Zahlen sind Speicherfresser. Bei 5 Bytes pro Floating Point (ohne Variablennamen) kommt peu à peu Einiges zusammen. Viele Anwendungen brauchen weniger als 8 Stellen Genauigkeit: Monat und Tag kommen mit der 'range' 1-12 bzw. 1-31 aus, Printstellen benutzen 0-22 und 0-31, Plotpositionen belegen nur den Bereich 0-255 bzw. 0-175 usw.

Ein erster Hebel zu ökonomischer Speicherung sind (künstliche) Integerzahlen. Mit den segensreichen Klauseln CODE bzw. CHR\$ lassen sich Zahlen des Bereichs 0-255 in einem Byte unterbringen, für INTEGERS von 1-65535 genügen 2 ganze Bytes. Das erste Byte erhält dazu die 256er Genzzahl, das zweite den 256er Modulus. Programmtechnisch ist das am besten mittels einer definierten 'function' zu realisieren. Sie liefert fürs erste Byte -> CHR\$(INT(Zahl/256)) beim zweiten Byte -> CHR\$(Zahl - (INT(Zahl/256))\*256).

Das Ganze hat nur den Nachteil, der häufig beim Speicherplatzsparen auftritt, - die Verarbeitung wird langsam... Man kann etwas anderes tun, um Daten wirtschaftlich zu speichern: Zeichenform (originär oder als Codes überführt) werden sie so angeordnet, daß nur der tatsächlich benötigte Platz belegt wird. Da wird mancher sofort an einen Riesenstrich denken - aber Vorsicht! Wenngleich das komfortable Stringlicing des SINCLAIR solches nahelegt, es geht nicht unbegrenzt. Bei etwa 14.500 Bytes ist Ende (mit Tricks etwas später). Selbst wer mutig die Praxis der Zeichenketten - Verlängerung betreibt, wird

die Grenzen schnell bemerken: 1. wird das Handling zusehens langsamer; 2. laufen nicht mehr alle Stringoperationen. Grund: intern fordert der Rechner Zwischenspeicher an und kommt bei Umbesetzungen dann mit 'out of memory' raus. Da liegt die Idee des Riesenstringarrays (ersatzweise) nicht fern. Für ihn gilt Nachteil 2 gleichermaßen... Was also tun bei viel Daten? Die interne Verschieberei hält sich in Grenzen, wenn (bei Zeichen) Blöcke gebildet werden. Um den User-RAM in Basic voll zu nutzen, sei die Idee einer RAM-Disk kurz angetippt. Diese in groben Zügen (nicht zu verwechseln von MC-Lösungen gleichen Etiketts!):

Zwecks sparsamer Adressierung über Codes sei eine Block-Höchstzahl von 255 gewählt (Tracks genannt). Jeder teilt sich auf in Sektoren a 130 Bytes, sodaß 33.150 Bytes Daten untergebracht und organisiert werden. Eine geschickte Adressierung vorausgesetzt werden variabel lange Sätze in jedem Sektor gespeichert. Das geht solange, bis der Block voll ist bzw. der aktuelle Satz (engl. Record, hier mit unterschiedlich langen Feldern) nicht mehr reingeht. Von der aktuellen Satzlänge hängt ab, wie gut die Speicherausnutzung ist. Nach Erfahrung des Autors ist das das beste Verfahren, mit Speichergrenzen des SINCLAIR zurecht zu kommen. Die folgende Tabelle zeigt die jeweilige Ausnutzung für einige Recordlängen zum Zweck eigener Überlegungen:

RECLEN REC/TRACK REC's used MEM in % (33.150 = 100 %)

10	13	2.550	76,9
20	6	1.530	92,3
30	4	1.020	92,3
40	3	756	92,3
50	2	510	76,9
60	2	510	92,3
70	1	255	53,8
80	1	255	61,5
90	1	255	69,2
100	1	255	76,9
130	1	255	100,0

Offenkundig ist die Speicherausnutzung von der Recordlänge abhängig. Viele kurze Sätze bringen im allgemeinen bessere Resultate als "unpassende" lange. Neben dem Vorteile einer im Schnitt 80% Belegung hat das Verfahren noch zwei Pluspunkte:

1. schneller Zugriff durch den Blockindex (s.o. Blocksuche)
2. voll variable Feld/Satzlängen

Allerdings ist Handhabung beträchtlicher Verwaltungsaufwand erforderlich. Ohne Datenzeiger ist das kaum zu schaffen. Sie müßten aufs einzelne Feld wie folgt zeigen:

- Angabe des 'tracks', in dem der Record liegt
- Position im 'sector', wo der Satz beginnt
- Über Längfelder schließlich zum Feld

Genug der Theorie, quasi als Anregung zum Thema "Speicherformen" für fortgeschrittene Leser gedacht. Zeigen wir nun konkret, wie man mit Datenzeigern arbeitet. Ein praktisches Beispiel soll die Verwendung von Kurztexten in der Form einstelliger Codes (im Datenfile) veranschaulichen. Das Paket "KUERZEL" ist raffiniert als die Datenzeiger bislang (wie in HEINBUDGET und REPDATA).

Platz sparen mit Pointern (Programm KUERZEL)

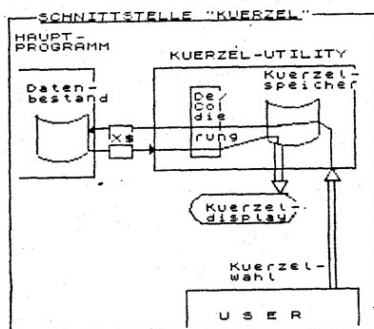
Pointer sind ein universelles Mittel, Daten zu verwalten. Egal, ob ein Zeiger auf Nachfolger zeigt oder den Weg in "Bäumen" weist; egal, ob er bei den Daten steht oder ausgegliedert ist, - mit diesem Mittel lassen sich nahezu beliebige Strukturen handhaben. Kurzum - mit Pointern kann man fast alles machen. Dabei ist das Prinzip alltäglich: "Ich weiß etwas nicht, aber ich kenne jemanden, der es weiß".

Im folgenden greifen wir ein Problem auf, das dem Heimcomputereisten oft begegnet. Man will im Datenfile Merkmale wie Autor, Titel, Jahr, Stichwort 1, Stichwort 2 etc. speichern. Für erstere braucht man Klartexte, bei Stichworten kann man auf eine Sammlung zurückgreifen. Also setzt man eine Zahl bzw. einen Buchstaben als "Ersatz" ein (immerhin 26 Varianten). Mehr erfordert zusätzliche Zeichen, oder? Nicht unbedingt, kann doch in einem Byte der Code von 0-255 gespeichert werden. Man muß nur entsprechende da/codieren.. Statt nun eine Tabelle zu bemühen, reicht ein "Kürzelstring", der die Texte hintereinander enthält. Will man eine Hierarchie, also Oberbegriffe dazu haben ("Titel"), geht das auch. Dann hüpf man von Titel zu Titel, um beim gewünschten "in die Tiefe" zu gehen. Alles läuft über Pointer, wie Programm KUERZEL zeigt. Es ist als Utility (einzubindendes Hilfsprogramm) aufgezogen und bietet ein hohes Maß an Bedienerfreundlichkeit (Cursorcontrol). Schauen wir es uns näher an.

Programm KUERZEL ermöglicht, Datenspeicher einzusparen durch Gebrauch einstelliger Codes. Der Benutzer bemerkt das nicht, da bei Aufruf lediglich der Kürzelklartext erscheint. Statt der Kategorien "Erwachsener, Jugendlicher, Senior" etwa wandert nicht die Langbezeichnung ins File, sondern nur der Code. Nicht allein wird so Speicher gespart, auch die Eingabe ist einfacher. Der User bestimmt per Tastendruck das Kürzel der Manuleiste am Screenfuß, Langtextinput entfällt. Das Programm realisiert daher zwei Forderungen: 1. Kürzelstring statt Bezeichnertabelle, 2. Auswahl der Kürzel mittels Pointer. Der Lösungsweg ist wie folgt.

Zu Anfang werden bei Kürzeleingabe die Längen mit gespeichert. Sie werden addiert, sodaß nach Abschluß die Anfangsstellen bekannt sind. Ein Datenzeiger wählt (nach der Initialisierung) den Start einer jeweiligen Gruppe an, um anschließend die Kürzel bereit zu stellen. Sie sind in Gruppen unter einem Titel organisiert. Der User springt erst von Titel zu Titel, ehe er die Kürzelleiste "herunter" geht. Der Gebrauch vertrauter Cursortasten bewirkt Seitwärtssprung bei den Kürzeln

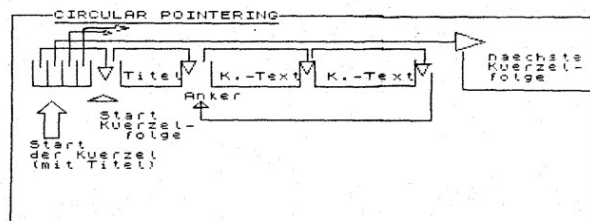
("Blättern"), Space quittiert die Wahl, eine andere Taste bringt die Titellebene. Die gewählte Kürzelnummer wird in XS als Code gespeichert und von Unterroutine K-IN dem Aufrufprogramm übergeben. Das wiederum kann via XS ein Datenitem K-OUT zur Verfügung stellen, wo es decodiert wird. Die Logik zeigt folgendes Schaubild.



Es können maximal 31 Kurztexte mit einer Gesamtlänge von 255 Bytes verwaltet werden; 31 Texte, um die Codes im Datenfile unterhalb des druckbaren Bereichs zu halten (einfache Abfrage !); 255 Stellen Länge, um die Verwaltung mit Pointern einstellig zu bewerkstelligen. Die Längen je Titel (= Oberbegriff) sind in Positionsstring JS, KS selber umfaßt alle Texte (Titel + Kürzel). Variable IEND erteilt Auskunft, wieviel Titel (=Kürzelgruppen) existieren. - Es gibt im Programm zwei Arten von Pointern :

- einfache Vorwärtszeiger (JS)-> jeweils die Startposition eines 'Titels', gespeichert als Code
- Ringpointer bei Kürzeln eines Titels (in KS als Code angelegt)

Die 'circular pointer' sind trickreicher: Jeder Zeiger weist auf seinen Nachfolger, nur der letzte zeigt wieder auf den Anfang (Anker genannt). Das kommt der Handhabung des Benutzers entgegen. Er wählt die Kürzel durch, bis wieder das erste erscheint. Das Prinzip ist in folgender Darstellung aufgezeigt.



Noch eine gute Hilfe für den User - beim Initialisieren (der Kürzeltexte) zeigt ein Balken oben, wieviel Platz frei ist, ein Raster unten, wieviel Kürzel noch eingegeben werden können. Abschließend das Programm, das routinenweise aufzurufen ist (!), und weiter Einzelheiten.

#### Funktionen

Zeile 200-208	Funktionendefinition
210-244	K-IN = Wahl des ins File zu übernehmenden Kürzels
248-254	K-OUT = Anzeige des Klartextes gem. Code (in XS)
258-322	K-DEF = Definition der Kürzel (Titel + Texte)
Unterroutinen	
264-270	Dimensionierung
274-284	Titel eingabe
288-296	Kürzeltexte
298-308	Andruck + Anlage der Längenfelder
312-318	Speichern Titelfeldlängen
322	Fehlerrückmeldung ("Überlauf")
324-328	Belegungsplot
330-346	Andruck Kürzelliste

#### Variablen

Strings	
XS	Kürzelcode bzw. OUT-Text
KS	Kürzeltexten (Texte + Pointer)
JS	Start Textpositionen in KS
IS	User-Input
CS	Dimensionierungshilfsstring

#### num.Variablen

N,M	Zähler
X	akt. Stelle in KS
IEND	Anzahl 'Titel' (=Kürzelgruppen)
LEN	Längenmesser (b. Textinput)
Z,S	Zeilen- + Spaltenzähler



## 151



# ERGEBNIS DER ZUORDNUNG

```

=====
TROCKEN ==> MEINE
          ==> DAMEN
          ==> UND
          ==> HERREN
          ==>
          ==> 7-810007
=====

```

## Der Screen als Speicher

Mit dem Display File als Datenspeicher eröffnet sich die reale Möglichkeit, fast volle 48 K RAM zu benutzen. Unmöglich? Keineswegs, man muß nur mit etwas Überlegung zu Werk gehen. Auf dem Bildschirm stehen normalerweise 704 Zeichen, den INPUT-Bereich nicht gerechnet. Eigentlich sind es etwas mehr, weil das System Farbattribute und ein unsichtbares CR (Carriage Return) ans Ende einer jeden Druckzeile einfügt..

In Wirklichkeit aber enthält das zuständige Display File Informationen über 176 \* 256 Pixels = 45056 Plotpunkte. Ein ungeahnter Datenspeicher, wenn man auf volle Grafik verzichtet. Ab der festen Adresse 16384 liegen zuerst 6.144 Bytes für den sichtbaren Screenbereich, daran schließen sich noch 768 Bytes Attributspeicher an. Insgesamt stehen 6.912 Bytes zur Disposition, wobei man zweckmäßigerweise die Attribute nicht benutzen wird. Adresse 16384 - 22527 käme also in Frage für Daten. Jetzt muß natürlich der Einwand kommen - und wo bleibt die Printanzeige? Dagegen kann man Vorkahrungen treffen, um eine "bildschirmlose Zeit" erträglicher zu machen:

1. Man läßt alle Anzeigen über die INPUT-Zone laufen; eine Methode 'Laufschrift' oder Textdisplay mit Cursorsteuerung (wie in HEIMBUDGET, DRAWUTIL oder bei KUERZEL) wären erster Ersatz.
2. Man erreicht bereits eine hohe Informationsdichte, wenn ein Teil des Screens zur Anzeige bleibt (vgl. die Scrolls in HEIMBUDGET).
3. Wenn Farbspiele nicht stören, kann den Attributspeicher ersatzweise bemühen oder eine Kombination von "ausgeblendeten" Zeilen mit Attributteilbereichen.

Wohlgemerkt, das Display File ist ein echtes File und kann als solches benutzt werden. - Dem Verzicht auf Grafik stehen bei SCREENDATA eine ganze Menge Vorteile gegenüber:

- echte 48 K Ram für Programm und Daten in BASIC!
- feste, vom System vorgegebene Adressen; wonach sich MC-Freaks sonst die Finger lecken..
- Eindringungschance von Basic aus in die Maschinenebene

Daß die Idee des Screenspeichers gar nicht so abwegig ist, zeigen manche Utilities; in BETABASIC beispielsweise werden Zwischenergebnisse von RENUMBER auf den Bildschirm gelegt.. Wenden wir uns der Programmlösung zu.

SCREENDATA ist als Demoprogramm aufgezogen. Es sollte nur das Prinzip dargestellt werden, nicht mehr. Zuerst wird ein echtes Datenfile installiert mit der stattlichen Größe von 38.400

Verwendungen. Man kann z.B. wiederkehrende Texte, ja sogar Stringzahlen (bis 8 Stellen - Vorsicht bei STRs und Dezimalpunkt!), dort unterbringen. Floskeln liegen nahe, es sind lediglich vereinbarte Aufrufcodes auf die UDG-Belegung zu lenken. Wie geht man vor?

Da das System die UDG-Belegung vorbereitet hat, braucht man sich nicht mit echten Speicheradressen abzugeben; "symbolische" Adressierung genügt. - Zunächst fragt unsere Routine den User, welchen UDG-Buchstaben eine Floskel erhalten soll. Dann wird der Floskeltext Byte für Byte auf das UDG gelegt. Jeder Buchstabe (von BS, 8 Zeichen lang) wird an die (USR-)Adresse gepaket, u.z. in seiner Codeform. Das wars denn schon..

Das Programm kann gelöscht werden, die maximal 21 Floskeln bleiben erhalten. - Der Aufruf ist Sache einer Programmzeile (s. Zeile 26). Der in IS eingetragene Buchstabe dient per PEEK USR (und Fortzählung), den Buchstabencode rückzugenerieren. Man soll und braucht wohlgemerkt nicht das UDG (mit "graphics"-Taste) selber aufrufen; dort steht nur die Information für die Floskel. Der Screenausdruck zeigt das recht anschaulich. Er zeigt auch, was die Routine generiert, wenn keine Floskelbelegung auf einem UDG lag - nämlich undruckbare Zeichen. Die Routine ist so klein und pfiffig, daß sie bequem in eigene Programme eingebaut werden kann. Alle Heimcomputeranwendungen, die Menuprints, Datenkürzel, Texttabellen (z.B. Monatskurznamen) verwenden, werden UDG-Floskel gebrauchen können. Auch für die Übergabe von 'Params' eröffnet sich ein weites Feld.

```

10 REM UDG-BUCHSTABEN-LEISTUNG
12
13 DIM BS(8)
14 INPUT "UDG-BUCHSTABEN (A-U)"; IS;
   TEXT BS
   IF IS="" THEN GO TO 20
15 FOR N=0 TO 7
   POKE USR IS+N, CODE BS(N+1)
   NEXT N
16 GO TO 13
17
18 REM AUFRUF
19
20 PRINT "ERGEBNIS DER ZUORDNUNG"
21
22 INPUT "UDG": IS
23 IF IS="" THEN STOP
24 PRINT IS; " ==> "
25
26 FOR N=0 TO 7
   PRINT CHR$(PEEK(USR IS+N))
   NEXT N
27 PRINT
28 GO TO 24
29
30 REM WEITERER VORTEIL - INFO
   RMATION BLEIBT USERS' PROG
   RAM HIN AUS ERHALTEN (NEUES
   CHUEZT!)
31 PRINT "ABCDEFGHIJKLMNORST
   U

```

# ERGEBNIS DER ZUORDNUNG

```

=====
MEINE
DAMEN
UND
HERREN
=====
?@#1234567

```

## Der Screen als Speicher

Mit dem Display File als Datenspeicher eröffnet sich die reale Möglichkeit, fast volle 48 K RAM zu benutzen. Unmöglich? Keineswegs, man muß nur mit etwas Überlegung zu Werk gehen. Auf dem Bildschirm stehen normalerweise 784 Zeichen, den INPUT-Bereich nicht gerechnet. Eigentlich sind es etwas mehr, weil das System Farbattribut und ein unsichtbares CR (Carriage Return) ans Ende einer jeden Druckzeile einfügt..

In Wirklichkeit aber enthält das zuständige Display File Informationen über 176 \* 256 Pixels = 45056 Plotpunkte. Ein ungeahnter Datenspeicher, wenn man auf volle Grafik verzichtet. Ab der festen Adresse 16384 liegen zuerst 6.144 Bytes für den sichtbaren Screenbereich, daran schließen sich noch 768 Bytes Attributspeicher an. Insgesamt stehen 6.912 Bytes zur Disposition, wobei man zweckmäßigerweise die Attribute nicht benutzen wird. Adresse 16384 - 22527 käme also in Frage für Daten. Jetzt muß natürlich der Einwand kommen - und wo bleibt die Printanzeige? Dagegen kann man Vorkerkungen treffen, um eine "bildschirmlose Zeit" erträglicher zu machen:

1. Man läßt alle Anzeigen über die INPUT-Zone laufen; eine Methode 'Laufschrift' oder Textdisplay mit Cursorsteuerung (wie in HEIMBUDGET, DRAWUTIL oder bei KUERZEL) wären erster Ersatz.
2. Man erreicht bereits eine hohe Informationsdichte, wenn ein Teil des Screens zur Anzeige bleibt (vgl. die Scrolls in HEIMBUDGET).
3. Wenn Farbspiele nicht stören, kann den Attributspeicher ersatzweise bemühen oder eine Kombination von "ausgeblendeten" Zeilen mit Attributteilbereichen.

Wohlgemerkt, das Display File ist ein echtes File und kann als solches benutzt werden. - Dem Verzicht auf Grafik stehen bei SCREENDATA eine ganze Menge Vorteile gegenüber:

- echte 48 K Ram für Programm und Daten in BASIC!
  - feste, vom System vorgegebene Adressen; wonach sich MC-Freaks sonst die Finger lecken..
  - Einbindungschance von Basic aus in die Maschinenebene
- Daß die Idee des Screenspeichers gar nicht so abwegig ist, zeigen manche Utilities; in BETABASIC beispielsweise werden Zwischenergebnisse von RENUMBER auf den Bildschirm gelegt.. Wenden wir uns der Programmlösung zu.

SCREENDATA ist als Demoprogramm aufgezogen. Es sollte nur das Prinzip dargestellt werden, nicht mehr. Zuerst wird ein echtes Datenfile installiert mit der stattlichen Größe von 38.400

Bytes. Damit sind knapp 2000 Records mit je 20 Stellen Zeichen als "Stamm" angelegt. Nachdem zu Testzwecken das binkäre Suchen von "HAIER" noch mal gezeigt ist, geht es an den Aufbau des Bildschirms - Datenspeichers. Dort sollen nämlich die Zusatzinformationen (Status über Herrn HAIER) hinkommen. Daß das Datenfile 1920 Records besitzt, hat einen tieferen Sinn: Bei einer Zeile "Zusatz" (= 32 Zeichen) lassen sich im Screenbereich (ohne Attributspeicher) genau 192 Eintragungen unterbringen. Modulo 192 ist die Größe, die ein abgespeicherter Screen immer hat. Werden nun die 'Teiler' berücksichtigt, denn machen genau 10 Screens das im User-RAM angelegte Maximum von 1920 Sätzen aus. Durch die Integerumrechnung (vgl. Zeile 44) werden jedem Datenrecord genau 32 Stellen Zusatzinformation zugeordnet. Der INT-Teil einer Satznummer adressiert geschickt den Zugriff per SCREENS-LOAD: die Zahl N1 wendet nämlich in den Ladenamen, sodaß die Routine Blöcke von 10 \* 192 verwaltet. Was nur mit Microdrive ohne viel Zeitverlust machbar ist..

Führen wir noch einmal die Steps einzeln vor Augen (Wem der Kopf schwirrt, braucht sich nur ums eigentliche Datenfile AS zu kümmern; er kann diesen Programmtell "aufbohren" und sich auf den Rest von SCREENDATA so verlassen, es funktioniert!).

1. Aus Datensubscript (= Recordnummer) M wird der Integer zur Basis 192 gebildet; er gibt an, welcher Screen zu laden bzw. mit welcher Screen-Nummer der Bildschirm zu serven ist (=N1).
2. Der Modulus liefert die Screen-Adresse (=N2); genauer, den Start der 32 Zeichen Zusatzinformation zum aktuellen Record.
3. Vom Start des Display Files (16384) wird nun in 32er Schritten die Speicherstelle bestimmt, ab welcher die Zusatzinformation in Codeform steht; 32 mal wird der Character daraus gepeekt bzw. dorthin gepaket.

Nachdem die Zeichen in IS gesammelt sind, können sie gedruckt bzw. bei Eingabe weiter verarbeitet werden. Und das wars auch schon... Abschließend noch die notwendigen Angaben zur Programmdokumentation neben demselben.

Noch ein Wort: Mit SCREENDATA lassen sich nicht allein 38.400 Zeichen Daten verwalten, das Verfahren ermöglicht darüber hinaus 1920 \* 32 = 6144 Bytes Zusatzinformation pro Screen. Da mit 10 Bildschirmen gearbeitet wird (die gehen bequem auf ein Microdrivecartridge), ergeben sich ganze 61 K an Zusatzinformationen. Insgesamt hat der User damit indirekt 99.840 Bytes im Zugriff. Er kann mit knapp 2000 Sätzen über 97 K (!!!) Daten verwalten.

Und da gibt es Leute, die behaupten, mit dem SINCLAIR ließe sich keine Datenverarbeitung betreiben.

## Programmaufbau

- Zeile
- 16-21 Initialisieren (Testdaten)
    - Dimensionieren
    - Testdaten belegen
    - Längenausrichtung Suchstring (BS)

22-36 Binärsuche  
 37-49 Datenspeicherung auf Screen  
 - Dimensionieren + Aufnahme Input (IS)  
 - Laden Speicheradresse  
 - Einlesen Zusatzinfos  
 57-62 Retrieval Screen-Infos  
 - Laden Screen  
 - Herauspeken Infos + Ablage in IS  
 - Andruck

#### Variablen

AS(1920,20)

IS Testdaten  
 IS Userinput/Ausgabe-Infos  
 BS Demo-Suchstring  
 SS Save/Load-Name

U,0,M Aufteiler f. Binärsuche  
 (M auch RecNr)

N1,N2 Adressverschlüssler

NA Startadresse Zusatzinfos (entspr. RecNr)  
 E Anzahl Records in Datenfile (Begrenzer)  
 N Laufvariable

```

10 REM TESTARRAY
12
14 REM D E M O
16
18 REM TESTARRAY EINRICHTEN
19 DIM BS(1920,20)
20 REM BS IS SPEICHERUNG VON NR FUER DATEN
21 NAME (BS) = SUCHBEGRIFF
22 LET BS(1919) = "MAIER"
23 LET BS(1920) = "ZZZZZZZ"
24 NAME BS = "MAIER"
25 LEN BS < 20 THEN LET BS = BS
26
28 REM BINÄRSUCHE
29 LET U = 0
30 LET O = E + 1
31 GO TO BS AS(E) OR BS AS(1) THE
32
34 LET M = INT ((O+U) * .5 + .5)
35 IF BS AS(M) THEN GO TO 36
36 IF BS AS(M) THEN LET O = M
37 GO TO 38
38 LET U = M
39 IF (O-U) < 1 THEN GO TO 24
40
42
44 PRINT "NOT FOUND"
45 STOP
46
48 PRINT BS, "FOUND AT "; M
49

```

```

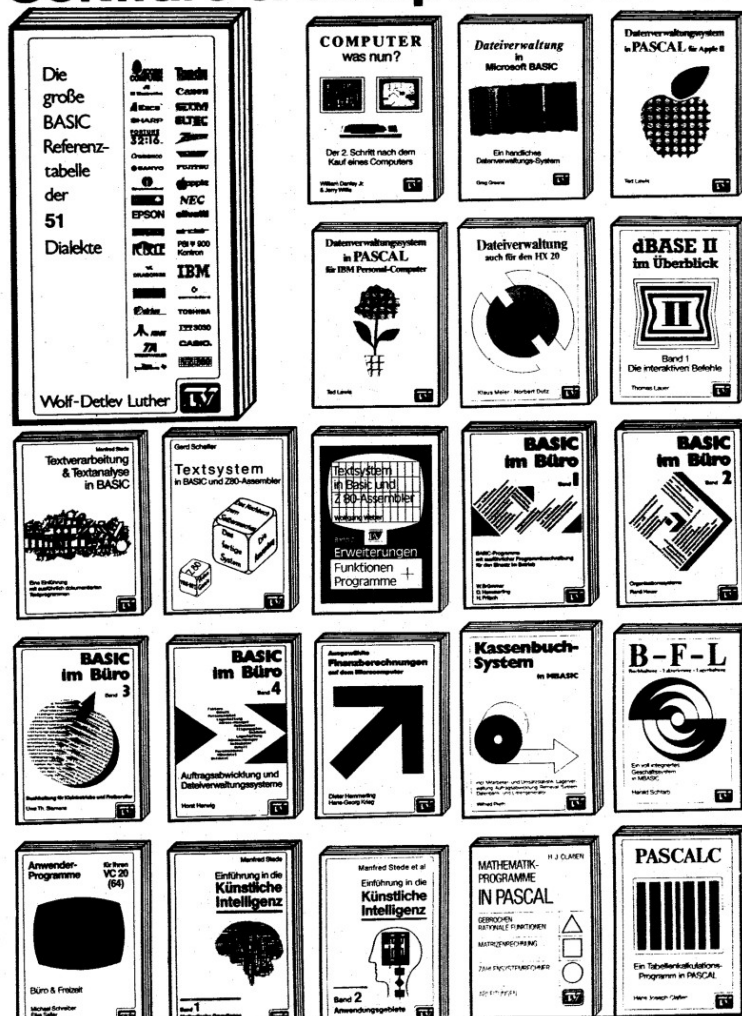
37 DIM IS(32)
38 INPUT "DATA(32 STELLEN)"; IS
39 REM DATEN IN SCREENSPEICHER
40 BRINGEN
42
44 CLS
45 LET N1 = INT (M/192)
46 LET N2 = M - N1 * 192
47 LET SS = "S" + STR$ N1
48 LET NA = 16384 + (N2 - 1) * 32
49 FOR N = 0 TO 31
50 POKE NA + N, CODE IS(N+1)
51 NEXT N
52
54 REM WEITERE DATEN...
55
57 SAVE "M"; 2; SS SCREENS
58
60 REM ZURUECKHOLEN (VEREINFACH
61 T, DA ALLE VARIABLEN BESETZT
62 BLEIBEN)
63
65 LOAD "M"; 2; SS SCREENS
66
68 DIM IS(32)
69 LET M = 1
70 FOR N = NA TO NA + 31
71 LET IS(M) = CHR$ (PEEK N)
72 LET M = M + 1
73 NEXT N
74 PRINT BS, IS

```

## Literaturverzeichnis

- LOGAN, Ian Understanding your Spectrum, UK Lincoln 1982  
ders. Das Microdrive Universum, München 1984
- TONS, Trevor The Spectrum Pocket Book, UK Surrey 1983
- STEWART, Ian  
+ JONES, Robin Machine Code and better Basic, UK Cheshire 1982
- JONES, Dilwyn Delving Deeper into your ZX Spectrum, UK London 1983
- PENNELL, Andrew  
ZX Microdrive-Buch, Stuttgart 1984
- JAMES, Mike Der Weg zur ZX-SPECTRUM-Meisterschaft, Würzburg 1985
- LAWRENCE, David  
The working Spectrum, UK London 1982
- MERZ, Jochen Maschinencode-Handbuch für den ZX Spectrum,  
Osnabrück 1983
- ders. Mikrodrive-Handbuch für den ZX Spectrum, Osnabrück
- HARTNELL, Tim  
+ JONES, Dilwyn Programming your ZX SPECTRUM, UK London 1982
- SOFTWELL, Tom ZX 81/Spectrum-Anwendungen, 6531 Gensingen 1984  
(Hrsg.)

## Software & Computer-Bücher



# Software & Computer-Bücher

