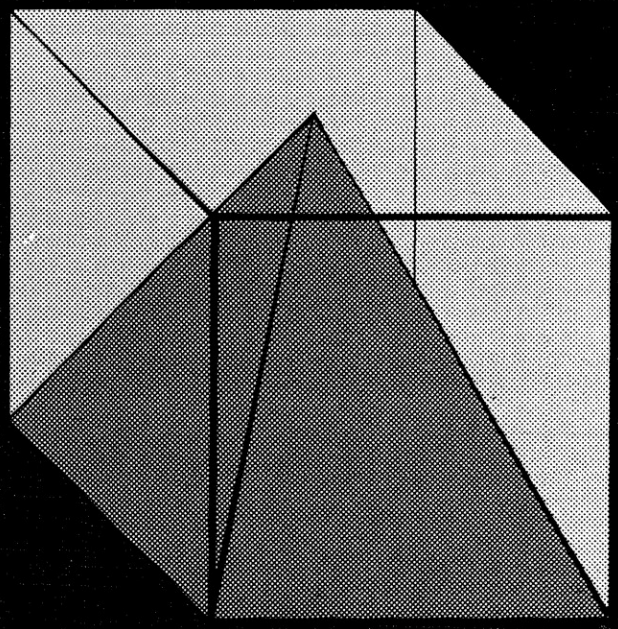
Wer sich ernsthaft mit dem SINCLAIR beschäftigt, wird dankbar sein, wenn er mehr als nur „Spiele" zu Gesicht bekommt. Die Grenzen von ZX81 und Spectrum liegen jenseits von STARTREK. Immer mehr Jugendliche und Erwachsene fragen, was man darüber hinaus mit einem „Micro" anstellen kann. Viele fangen an, selber in BASIC, der verbreitesten 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 Profi am Werk, 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.

SINCLAIR BASIC



FJ
5420

Tom Softwell

Sinclair BASIC

Programmiertips & -tricks

für ZX 81 und Spectrum

Titelseite: Peter Spann, Karlsruhe

KAPITEL II: G R A F I K

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

KAPITEL IV: V E R A R B E I T U N G V O N D A T E N

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 STARTREK. - 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 Machine 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 Verwaltung 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 + VCHARS generieren Riesenbuchstaben bzw. Vertikalschrift (mit einem völlig neuen Screengefühl)
TEXTVER bringt eine komplette Textverarbeitung incl. Formatierung und Blockoperationen
REPDATA verwaltet Daten beliebigen Typs gemischt(Text + Zahl)

und stellt sie in frei definerten 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, Hardcopys, Variablenliste, Benutzungshinweise etc. So kann der Lesen 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 SINCLAIT die Aufschrift tragen

Der Bundesminister für Forschung und Technologie :

VORSICHT, DER HC KANN IHR FAMILIENLEBEN GEFAHRDEN !

Kapitel I : Die Sprache

EIGENSCHAFTEN

Stärken

Syntax und Semantik

Diss Kapitel ist als Einführung zu den Anwendungen in SINCLAIR-Basic zu verstehen. Man erwarte 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 Merkposten für Programme bzw. Routinen des Buches - ohne Anspruch auf Vollständigkeit.
Eine Sprache - und das ist SINCLAIR-Basic nun mal - ist bestimmt von Sytax 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 Programmlösungen. Statementklauseln können so starr sein, daß nur ein Haufen Spezialfälle übrig bleibt. Ist die Sprache befehlsarm, aber die Verwendung (bei wenig Vorschriften) 'frei', läßt sich viel mit ihr anstellen. Wenn jedoch ein Basic befehls- und anwendungsstarr 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 (seine) Eigenschaften zusammenpassen, 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 Seltenheit 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, eins seiner besten Eigenschaften - seine Methode sollte Standard werden." Vom 'Standard' Microsoft allerdings ist das Basic etliches entfernt. - Da gibt es die Stringaufteiler LEFT$, RIGTH$ und MID$ beispielsweise. Jahrelang kritiklos übernommen und gedankenlos "geschluckt" kam Sir Clive und zeigte, daß man mit einfacher TO-Klausel bessser arbeiten kann. Drei Befehle weniger, aber vielseitiger, einleuchtender und eingabefreundlicher. Das ist typisch fürs SINCLAIR-Basic. Übrigens, wer zur Konvertierung ein Umwandlunghilfe braucht - die folgenden Functions erledigen L-eft, R-igth und M-id:

```
DEF FN L$ (X$,X) = X$ ( TO X)
DEF FN R$ (X$,X) = X$ (LEN X$ - X + 1 TO )
DEF FN M$ (X$,X,Y) = X$ (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 SCREEN$ zur Zeichenbestimmung u n d Bildschirmladen/Saven, leicht zu definierende UDG's - ohne Adressrechnerei (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 Flexibiltät, seine Narrensicherheit, ehe wir kritisch Schwächen aufzeigen. Auf Einzelheiten der Sprache wird noch im nächsten Kapitelteil einzugehen sein.

Flexibiltät

Eine Programmsprache 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 INKEY$ READ PRINT LPRINT
vordefinierte(mathematische) Funktionen	FN X() LN SIN ABS usw.
String- + Arrayhandling	DIM + Subskribierung
Zuweisungen und Datenkonvertierung	LET STR$ CHR$ CODE VAL VAL$
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:


```
LET X$ = X$(n1 TO n2-1) + X$( TO n1-1) + X$(n2 TO) + Y$
```

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 I$ (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 behandelt 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 HCs, 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-Zeilenkorrektur. 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 I$ (5 TO 2)
```

Das gilt auch für Index 0, wo ansonsten eine Fehlermeldung kommt : (N=0) PRINT I$ (TO N)

Wird einem Kettenglied andererseits ein Leerstring zugewiesen, behandelt das System ihn wie Space (wie im allgemeinen Procrustes größen läßt !):

```
(A$="") LET I$(5) = A$
PRINT I$  -> **** ***
```

Print +
Plot Positionierung

Sowohl PRINT AT wie PLOT Y,X funktionieren auch bei gebrochenen Zahlen. Insbesonders 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 gesichtert, 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 Unaufmerksamkeit 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 Handicap

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 Programmarbeit als solche erleichtern. Arbeitshilfen wie TRACE,RENUMBER etc. haben das Stadium von Zusatzsoftware oder Clubtips verlassen. Gravierender sind hingegen Befehle, die SINCLAIR erst seinem 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 unerklärbaren 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 ims 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 Exponten 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 Hase 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 Ärgernis. 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 Stringandruck 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
"Ärgernis" gehören:

+ Auf das Konto Schlamperei bei Gleitkommazahlen geht die ungenaue 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 SCREEN$-Funktion ist eine gute Sache. Man kann mit ihr sogar eine kleinen Textprozessor "bauen", indem immer angesteuerte Zeichen zurück geprintet werden. Aber Vorsicht ! PRINT SCREEN (0,0) + SCREEN$ (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 Werbemanagern 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 unzwischen 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 I$ = "": LET U = NOT PI: LET X = U: LET Y = X
DIM A$(5): LET B$ = A$
```

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

Beim Initialisiern 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 I$)
```

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

```
IF I$ <> "" THEN LET X = VAL I$
```

Vielseitiges VAL

Die Extensionen von VAL und dem weithin ungebrauchten VAL$ 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 VAL$ ist allein durchs Handbuch nicht zu eruieren. Arbeiten wir zuerst das heraus, ehe wir auf so interessante Dinge wie indirekte Feldnamensbestimmung (per VAL$) eingehen.
VAL gibt den numerischen Wert, der als Zeichenfolge in einem String steht, VAL kann aber auch eine Vielfalt von Rechenvorschriften verarbeiten (wie eine 'function'). VAL A$

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

```
LET ZAHL = 1000: LET A$ ="ZAHL": PRINT VAL A$ bringt keinen
'error', sondern den Inhalt von ZAHL zutage => 1000
```

VAL$ demgegenüber liefert das Ergebnis als Zeichenkette zurück, setzt bei Zahleninhalt infolgedessen Anführungszeichen voraus. PRINT VAL$ "A$" ist gleichbedeutend mit PRINT A$, druckt also "ZAHL" (als Zeichenfolge). PRINT VAL$ 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 I$ = "A * 5 + SQR B"
```

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

```
LET A$ = "": LET SUM = 0 : DIM A(5) => mit Werten gefüllt...
.
FOR N = 1 TO 5: LET A$ = A$ + "+ A(" + STR$ N +")": NEXT N
LET SUM = VAL A$: 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 VAL$ 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 A$, B$ oder C$ an:

```
LET A$="HIER DER ERSTE": LET B$="ICH BIN NR 2": LET C$="NR 3 "
LET X$="ABC": INPUT "STRINGNR ?";N: IF NOT N AND N>3 THEN STOP
LET Z$= VAL$ (X$(N)+"$"): PRINT Z$
```

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 S: IF N=S THEN READ X$:PRINT X$: 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 eins der vielen ON .. GOTO-Surrogate zu stricken:

```
DATA 10,23,44,231,32
.
=>Sprungqalifizierer 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 Programmiertechnik. Insbesonders die IF-losen Abfragen werden uns zuerst beschäftigen. Nach ein wenig Systematik kommen wir dann zu einer Reihe arbeitserleichternder Tricks.

Abfragen und Vergleiche

Die Boolschen 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 Vergleichergebnis "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 sgn. Boolschen 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 Boolscher 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 Boolsche 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:

```
X$ AND Y$ ergibt (Leerstringtest !) X$, wenn Y$ <> ""
          ergibt "", wenn Y$ = ""

X$ AND Y  ergibt X$, 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 Boolscher Operatoren komplett:

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

IF X$ THEN...     => Aktion, wenn X$ <> ""
IF NOT X$ THEN..  => Aktio, wenn X$ = ""
```

Natürlich lassen sich IF-Direktabfrage und Operatoren miteinander kombinieren, Opertoren verketten 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 * (I$=1) + 220 * (I$=2)
```

3.'Verifikator' mit AND/OR

```
INNPUT ("TEXT" AND T);I$
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>9)
```

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

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

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 Boolscher Operator

```
GOTO (100 AND A<10) + (222 AND A>9)
```

Was wiederum vereinfacht werden kann

```
GOTO 100 + (122 AND A>9)
```

bzw. `GOTO 100 + 122 * (A>9)`

IF-lose Abfragen

In loser Folge sollen praktische Anwendungsfälle behandelt werden, um zu zeigen, was man alles (ohne IF und damit oft kürzer und kompakter) mit Boolschen 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-lose 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 ?";I$
PRINT ("FIRMA" AND I$="6");("NAME" AND
      I$="2");("ANSCHRIFT" AND I$="1")
```

Damit werden gleich vier Vorteile gleichzeitig wahrgenommen:

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

Schalter

Schalter sind Merkposten, die im Programm Auskunft über einen Zustand oder ein (vorheriges) Ergebnis geben. Der Möglichkeiten gibt es viele, z.B. ob ein Upro ok war 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 "Brigth" 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 ?";S$;" ANZAHL ?";N
.
IF S$ = "S" THEN LET D(1) = D(1) + N
IF S$ = "U" THEN LET D(2) = D(2) + N
IF S$ = "C" THEN LET D(3) = D(3) + N
IF S$ = "A" THEN LET D(4) = D(4) + N
.
```

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

```
.
LET I = (S$="S") + 2*(S$="U") + 3*(S$="C") + 4*(S$="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 Wertetransformation. 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 hinzuzuadieren oder aber man arbeitet mit dem bekannt vielseitige VAL. Wenn die Zeilennummern also 100, 200 und 876 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 "100200876"(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 (A$(ENDE)) wird mit dem Suchwort (B$) verglichen. Da N bei "nicht gefunden" immmer 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  A$(N) = B$ THEN LET N = ENDE + 1 + N
NEXT N: LET N = N - ENDE - 1
IF N THEN PRINT B$;" 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 A$(N) = B$ THEN GOTO ... <FOUND>
```

Unsere Routine erledigt das ohne GOTO indirekt gleichermaßen.

```
FOR N = 1 TO ENDE
IF A$(N) <> B$  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 B$ = A$(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 verästelte 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 sgn. logische Konstrukte - da nicht im 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 wiederspricht dem Prinzip zeilenloser Programmierung. (Das 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)

```
                                          < Bestandteile>
FOR N = 0 TO 1
                                          DO WHILE
LET N = NOT (BEDINGUNG)                   (BEDINGUNG)

IF (BEDINGUNG) THEN                           .
                                              .
    (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 Tips 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
   Variablennamen sind je kürzer, je besser. Am besten arbeite 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 Boolsche Logik entlastet den Compuer von aufwendiger Rechnerei, 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, Drucksteuercodes 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 Heimprogrammierer, 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 gleichartiger GOTOs.

3. Statements kompakt halten - also viele Befehle in eine Zeile,Prints mit Druckkomma und TAB zusammenfassen,direkte Indizierungen (statt 100 mal LET X = Y(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 - RAMTOP auf tatsächlichen Programmumfang herabsetzen..

BESSERES BASIC MIT BETABASIC

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 scheut, unbesehen 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 Eingebens abgesehen : die verschiedenen Routinen passen in den seltensten Fällen zueinander. Nur wenige MC-Programmchen sind allokierbar, 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 vermessen, 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 T A B A S I C. Das Erzeugnis ist auch in Deutschland erhältlich,die neueste Version 3.0 läßt kaum noch Wünsche offen. Das gilt insbesonders 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 aufzeigen, 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 SQ).

Die Schwerpunkte der zahlreichen Hilfen, Befehlserweiterungen und Unterstützungsfunktionen lassen sich wie folgt klassifizieren. Damit ist im folgenden keineswegs alles, was BEATBASIC 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 notwenige Vorraussetzung 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 Listandruck eine Option ohne Zeilenummer gewählt werden kann.

Grafik

Zunächst enthält die Kassette neben den üblichen Scrolls, Rolls, 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; einsame "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 Controllcodes geschaffen wurden. Letztere lassen sich auch in normalem Text verwenden, wie auch jeder String pixelgenau plazierbar ist..

Das ist noch lange nicht alles. Screenteile sind in Strings abspeicherbar, Bildschirmteile im Hauptspeicher ablag- und abrufbar. Figuren können mit FILL gefärbt, Farbattribute ohne Neuzeichnen des Screen 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.

Arbeitserleichterungen

Unter die Rubrik Arbeitserleichterungen 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 Zusammhang 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 programmtechnischen 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 M$ wird MEMORY, aus T$ wird TIME usf. Das Handbuch erkennt dies an - wie auch eventuelle Schwierigkeiten mit Printsteuerzeichen -, und gibt Hilfestellung. - Noch ein Merkposten bei Nicht-BETABASIC's: Programme sind zur Basicerweitung 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 höllisch 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 Variablennamens 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 eingängisten 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 Hochauflösung (HiRes abgekürzt) ist es, Daten optisch zu aufzubereiten. Grafik heißt, vereinfacht ausgdrückt, Zahlen in Strecken umzuformen. Von Spielereien und "Kunst" abgesehen liegt Kurven, Säulen, 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 Plazierung 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 I n i t i a l i s i e r u n g (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 M$. Der Januar bekommt eine 3 (31 Tage = Basis 28 + 3), der April mit 30 Tagen eine 2 (Basis 28 + 2) und so fort. Eine Schleife (in Zeile 48) errechnet und speichert die Monatszahlen als Codes im M$. Dort stehen sie im direkten Zugriff zur Abfrage bereit.

Vorbereitet wird der M o n a t s p r i n t mit Einsetzen des Startwochentages eines Jahres (vom 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 L a y o u t (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 I$ enthält dazu erst den "Balken", dann die WochtagsKürzel. Sie werden - einmal geladen - nach Andruck abgehackt (Zeile 204); das ist die einfachste Methode, einen String in gleichen "Stücken" ohne viel Rechnerei abzuarbeiten. Solang der gewählte Monat (noch) nicht da ist, wird im Upro 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 [redacted]
  8
 10 REM ****INITIALISIEREN***
 20 CLS
    LET JA=0
 30 REM MONATSENDTAGE
 40 INPUT "JAHR ->";JAHR
 41 IF JAHR<1900 OR JAHR>2000 T
    HEN GO TO 40
 42 LET Z=ABS (1984-JAHR)
    LET Z=Z/4-INT (Z/4)
 45 LET M$="3"+STR$ Z+"32323323
    30"
 48 FOR N=1 TO 12
     LET M$(N)=CHR$ (VAL M$(N)+
     48)
    NEXT N
 50 REM MONATSPRINT
 51 IF JA=0 THEN INPUT ("WOCHEN
    TAG 1.JAN.";JAHR);"HILFE:1=
    MO,2=DI,3=MI,4=DO,5=FR,6=SA
    ,7=SO";JA
 52 IF JA<1 OR JA>7 THEN LET JA
    =0: GO TO 51
 53 INPUT "JAHLMONAT=ZAHL (0=GAN
    ZES JAHR)";UM
    IF UM>12 THEN LET UM=0
 54 RESTORE
 56 FOR M=1 TO 12
     CLS
     LET SP=7
     FOR N=1 TO CODE M$(M)
      IF UM=M OR UM=0 THEN PRIN
      T AT 4+JA*2,SP+(N<10); IN
      VERSE (JA+7);N
 62    LET JA=JA+1
       IF JA>7 THEN LET JA=1
       LET SP=SP+4
 64   NEXT N
      GOSUB 200
    NEXT M
 66 INPUT "FOLGEJAHR ? JA=J";I$
    IF I$="J" THEN LET JAHR=JAH
    R+1
     CLS
     GO TO 42
 68 STOP
 90 REM *** E N D E ******
 99
100 REM >>UNTERPROGRAMM<<<
101
199 REM LAYOUT
200 IF UM=0 THEN GO TO 202
201 IF UM<>M THEN READ T$
     RETURN
202 LET T$="[redacted]"
    PRINT AT 0,0;T$;AT 4,0;T$;A
    T 20,0;T$;AT 5,5;"|";
    LET T$="MODIMIDODFRSASO"
203 FOR N=1 TO 19
     PRINT AT N,0;"■";AT N,31;"
     ■";
    NEXT N
204 FOR N=6 TO 18 STEP 2
     PRINT ;AT N,2;T$( TO 2);"
     ";AT N+1,5;"|";
     LET T$=T$(3 TO )
    NEXT N
206 READ T$
    PRINT AT 2,3;T$;TAB 24;JAHR
208 INPUT "HARDCOPY ? J=JA/RETU
    RN=NEIN/S=STOP";I$
    IF I$="J" THEN COPY
209 IF I$="S" THEN STOP
210 RETURN
```

```
214 REM MONATSTEXTE
218

220 DATA "JANUAR","FEBRUAR","MA
    ERZ","APRIL","MAI","JUNI","
    JULI","AUGUST","SEPTEMBER","
    "OKTOßER","NOVEMBER","DEZEM
    BER"
```

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, Beschriften Wochentage + Monatsbezeichnung
210
220 DATA-Festwerte = Monatsbezeichnungenß

Variablenliste

JAHR Vorgabejahr
JA Wochentag des 1. Tages von JAHR
N Schleifenvariable (Druck + Monatstageszähler)
M Zähler Monate
SP Spaltenzähler
WM Wahlmonat
Z Zählvariable

I$ User-Input
M$ Monatstage (als Codes)
T$ Druckstring

Säulengrafik

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 auflösender 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 getreues 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 entwicken und alles ist gegen Überlauf zu sichern. Das wird uns im Folgenden beschäftigen; zeigen wir erst einmal, wie einfache Säulengrafik auf Schirm und Papier zu bringen ist.

Wir bleiben bei Monaten und gehen von einer simplen Umsatztstatistk in Säulenform aus. Was nicht so schwierig wie Fläche (sgn. Tortengrafik) ist , aber schon anspruchsvoller als Kuven auf dem Bildschrim. Immerhin enthält Programm "Sasulengrafik" schon so interessante Dinge wie Vorjahreszahlen und automatische Justierung nach 'range' (= Wertesausprägung zwischen einem Minimum + Maximum). Wegen der Säulenbreite 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 HEIMBUDGET; hier soll lediglich der Weg zur "Säule" 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 Werteinheiten ergeben 7 Pixel)

3. Generieren der Y-Werte (als Vertikalstriche) von Koordinatenursprung aus und Fortschreiben der Horizontalachse

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

"Saeulengrafik" 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 136):
Konstante f = Differenz Max-Min geteilt durch 151 Pixels (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..

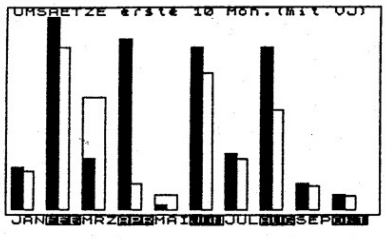
Natürlich gehört zu "Saeulengrafik" noch mehr, z.B. richtige Menusteuerung und optisch ansprechende Ausgabe der Zahlen mit Vergleichswert (Vorjahr). Bei der Grafik selber ist zu beachten, daß aktuelle Monatswerte immer als gefüllter Balken erscheinen, während der Vergleichwert als Schatten daneben steht. 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 Saeulengrafik
  12
  14 DEF FN g()=4+(a(i+1,X)-min)
     *f
  16
  17 LET d$="JANFEBMRZAPRMAIJUNJ
     ULAUGSEPOKT"
     REM BEISPIEL f.10 Monate
  18
  19 REM Mini-Menu
  20 CLS
     PRINT '"        [illegible]"
     ''"      OPTIONEN:"''; PAPER
      7;"  1-Werte eingeben/lade
     n"'"  2-Programm/Werte Save
     n"'"  3-Werte ausgeben"'"
     4-Grafik generieren"'',"(Re
     turn=STOP)"
  22 INPUT "Bitte waehlen.."; LI
     NE i$
     IF i$>"0" AND i$<"5" THEN G
     O SUB 100*VAL i$
  24 IF i$<>"" THEN GO TO 18
  26 STOP
  28
```



```
  98 REM [illegible]
  99
 100 INPUT "Anzahl (Doppel-)Wert
     e>Max.10< ";z
     IF z<1 OR z>10 THEN GO TO 1
     00
 110 INPUT "Werte [illegible]aden/[illegible]ingeben
      ?";b$
     IF b$="e" OR b$="E" THEN GO
      TO 120
 112 IF b$="l" OR b$="L" THEN IN
     PUT "Ladename array";b$
      IF b$<>"" THEN LOAD *"m";1
      ;b$ DATA a()
       GO TO 122
 114 RETURN
 120 DIM a(z,2)
     LET c$=""
 122 LET min=999999
     LET max=0
 123 LET x=1
 124 FOR i=1 TO z
 125  IF c$<>"" THEN LET w=a(i,x
     )
       GO TO 131
 126  INPUT (i);",";("Vgl." AND
     x=2);"Wert ? ";w
 128  IF w<0 THEN GO TO 126
 130  LET a(i,x)=w
 131  IF w>max THEN LET max=w
 132  IF w<min THEN LET min=w

 134 LET x=2 OR x=2
     IF x=2 THEN GO TO 125
 136 NEXT i
     LET f=151/(max-min)
 138 RETURN
 140

 200 REM [illegible]
 202

 210 INPUT "Saven [illegible]rogramm/[illegible]rray
     ";b$
     IF b$<>"P" AND b$<>"p" AND
     b$<>"A" AND b$<>"a" THEN GO
      TO 210
 220 INPUT "Name ?";c$
     IF c$="" THEN GO TO 220
 230 IF b$="P" OR b$="p" THEN SA
     VE *"m";1;c$ LINE 10
      RETURN
 240 SAVE *"m";1;c$ DATA a()
 242 RETURN

 300 REM [illegible]
 302

 310 INPUT "Anzahl Werte geladen
     /eingegeben","(wenn nicht=0
     )";z
 312 IF NOT z THEN RETURN
 320 CLS
     PRINT '"        [illegible]
     [illegible]"''"Monat Umsatz Vgl.Vo
     rj;"'"  ________________________
 330 FOR i=0 TO z-1
 332  PRINT d$(i*3+1 TO 3*(i+1))
     ;
 334  PRINT TAB 7;a(i+1,1);TAB 1
     5; INVERSE 1;a(i+1,2)'
 336 NEXT i
 340 INPUT "copy ?[illegible]/[illegible]";b$
     IF b$="y" OR b$="Y" THEN CO
     PY
 350 RETURN

 400 REM *** Generieren der
     [illegible]
```

```
410 CLS
    PRINT " UMSAETZE erste 10 M
    on.(mit VJ)"
412 PLOT 4,8
    DRAW 0,167
    DRAW 251,0
    DRAW 0,-167
    DRAW -251,0
414 LET x=0
416 FOR i=0 TO z-1
418  LET w=i*3+1
420  PRINT INVERSE x;AT 21,w;d$
     (w TO w+2);
422  LET x=NOT x
    NEXT i
426 REM DRAW-Umrechnen d.Werte
430 FOR i=0 TO z-1
432  LET x=1
434  FOR j=i*24+8 TO i*24+16
436   PLOT j,12
      IF a(i+1,x) THEN DRAW 0,F
      N g()
438  NEXT j
     LET x=2
440  PLOT j,12
     IF a(i+1,x) THEN DRAW 6,0
      DRAW 0,FN g()
      DRAW -15,0
      DRAW 0,-FN g()
442 NEXT i
444 INPUT AT 0,0;"ScreenCOPY?=>
    Taste (No=RETURN)";c$
    IF c$="" THEN RETURN
446 COPY
450 RETURN
```



Monatsergebnisse

Monat	Umsatz	Vgl.Vorj.
JAN	22.3	
FEBR	34.55	
MRZ	23	
APR	32.7	
MAI	19.3	
JUN	32	
JUL	23.45	
AUG	32	
SEP	21	
OKT	20.1	

Angaben zum Programm

Aufbau

Zeile

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

Variablen

a(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$	Ladename
d$	Monatskurztext
i$	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. Letzere 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 boolsche Variable (wie in PASCAL und anderen Hochsprachen bekannt). Lautet die Screenuntersuchung JA (TRUE), so wird der Wert 1 rückgeliefert; ist kein Pixel gesetzt, erhält 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 Vergleichergbnis 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 Zielkordinaten. 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; Zielkoordinate - 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
    ATEN
 22  LET XA=80
     LET YA=170
     LET XE=190
     LET YE=140
 24  LET XZ=30
     LET YZ=60
```



```
 25
 26 PRINT AT 1,10;"INSCREEN COP
    Y";AT 2,10;"ist in Basic";A
    T 5,10;"sehr ";CHR$ 6; "LO
    NGSAM"
 27
 28
    REM FEHLERCHECK
 30 IF XA>=XE OR YA<=YE OR (XZ+
    (XE-XA))>255 OR (YZ-(YA-YE)
    )<0 THEN PRINT "FEHLER"
    STOP
 32
 33 REM VERSCHIEBEDISTANZ
 34
 35 LET X=XA-XZ
    LET Y=YA-YZ
 36
 37 REM COPY-PLOT
 38
 40 FOR M=YA TO YE STEP -1
     FOR N=XA TO XE
      IF POINT (N,M) THEN PLOT
      N-X,M-Y
 42  NEXT N
    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 Excurs 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 SCREEN$). - Der Screen ist in drei Bereiche aufgeteilt, wobei die Attribute getrennt hintenan liegen. Der SCREEN$-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, Hervorzaubern 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" 16384,4096

Dadurch wird der oberen 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 "hexe" 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.

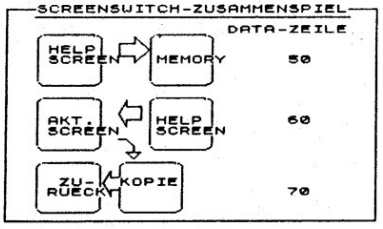
```
10  REM HEXLOAD(OHNE CHECKSUM)
12

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 "2100401100f0010010edb0c9"
58

59  REM akt.SCREEN <->HelpSCREEN
60  DATA "2100401100e0010010edb02100f0110040010010edb0c9"
68

69  REM switchback akt.SCREEN
70  DATA "2100e0110040010010edb0c9"
```

Wie arbeiten die ScreenSwitch-Routinen zusammen ?

Die drei MC's bewirken aufeinander abgestimmt folgendes:

- laden des Bildschirm (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 wir vielleicht deutlicher, wenn man das folgende Schaubild betrachtet.



Nach "Installation" der Routinen wird man überwiegend mit den

letzten beiden MC's arbeiten; ein Helpscreen ist in der Regel dann vorhanden. - Was kann das Verfahren ?

Vorteil :+ durch partiellen Screen-Switch wird Speicher gespart
+ die Darstellung "oben" erlaubt es, die Bildschirmarbeit "unten" weiterlaufen zu lassen
+ Screen-Swich hält noch den letzten Bildschirm in Reserve
+ Farbattribute bleiben, wie sie sind

Nachteil: - da zwei Screens gleichzeitig gehalten werden (d.h. der Help und der aktuelle), werden mit lockerer Hand 4 K RAM "verbraten"

Man sollte "Screen-Switch" daher nur anwenden, wenn man reichlich Speicher hat oder dichte Informationen als "Hilfe" parat halten will.

Scrolls

Bewegungen des Bildschirms im Ganzen (= Scrolls) werden inzwischen in jedem guten "toolkit" angeboten. Weswegen wir die folgenden MC's bringen, hat einen triftigen Grund: Wir brauchen sie zur Grundausstattung von Programm DRAWUTIL.
Der Zeilenscroll ist bei hochauflösender Grafik sekundär, wichtiger ist der Pixelscroll. Dabei wird der gesamte Screen um einen einzigen Grafikpunkt nach oben, unten rechts oder links versetzt. Mit Scoll up und left kommt man bei der Erstellung von Darstellungen, die über mehrere Bildschirmseiten laufen, ganz gut zurecht. Man unterbricht das Programm dazu, saved das Bild oder druckt es aus und "malt" weiter. So lassen sich auch umfangreiche Grafiken erstellen.
Die MC-Routinen lädt man sinnvollerweise an diese Adressen, um sie gleich für DRAWUTIL nutzbar zu haben:

Scroll left => Adresse 32600
Scroll up => Adresse 32700

Warum nicht Scrolls in Basic - mag man fragen ? Die Sprache ist einfach ungeeignet, sie wird wegen des Interpreters zu langsam. Erst in Ergänzung mit MC's wird Basic schnell und flexibel, ohne daß man die Übersicht verliert. Die folgenden Scroll-Routinen erschienen uns kurz und brauchbar, deshalb wurden sie ausgewählt (Dank an HAPPY COMPUTER 3/85,S.98 ff).

```
10  REM HEXLOADER
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
    LET C=0
30  READ C$,X
    REM X=CKECKSUM
```



```
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 C=C+E
     LET START=START+1
    NEXT B
46  IF X<>C THEN  PRINT '"FEHLER =>DATA-ZEILE ";N
48

49  REM SOFT-SCROLL(DOWN)
50  DATA "11ff5621ff570ec006201a7779fe02200297122b1b10f3d511e0
00a7ed52e3a7ed52ebe10d79e607fe002008d511e00719d1180bfe012007e521
e00719ebe179e63ffe0020067cd6076718bcfe0120b87ad6075779fe0120afc9
",9862
55

59  REM SOFT-SCROLL (UP)
60  DATA "1100412100400ec006201a7779fe02200297122313i0f3d511e0
0019e319ebe10d79e607fe00200ad511e007a7ed52d1180efe01200ee5eb11e0
07a7ed52ebe179e63ffe0020063e07846718bbfe0120b73e07825779fe0120ae
c9",9228
62

89  REM LINKS-SCROLL (PIXEL)
90

91  DATA "0ec021ff570620b7cb162b10fb0d20f5c9",1028
92
```

Bildschirmhilfe DRAWUTIL

Was muß ein Grafik-Utility alles können ?

Ein Utility soll eine Hilfe sein, welches die Screenarbeit erleichtert; es muß mehr als ein wenig Plot + Draw können. Ohne mit kommerziellen Produkten konkurrieren zu wollen, ein solches Basic-Programm hat zwei Grundforderungen zu erfüllen

1. schnelle bzw. präzise Bewegung auf dem Bildschirm
2. möglichst viele Grafik-Funktionen in direktem Zugriff

Ersteres bedeutet etwa einen leicht zu steuernden Pixel-Cursor, rasch anzuwählende Menupunkte (ohne die Grafik zu überlagern) und Verschieben des Screen in unterschiedlichen Ausschnitten. Wünschenswerte Funktionen sind Zeichnen von Strich,Kreis und Box, Plazieren von Text sowie Copy von Bereichen. Mehr ist sicher naheliegend (z.B. Fill, Erzeugen von Pattern, Zoom-Effekte, Farbe), in Basic jedoch kaum zu realisieren oder erfordert Joystick, wenn nicht Grafik-Tablett.
Programm DRAWUTIL kann schon eine ganze Menge. Es bleibt auf dem Boden der Tatsachen, arbeitet also in Basic mit zwei unterstützenden Maschinencode-Routinen (s.o.). Stellen wir die

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 plazieren
- 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 INKEY$) die Steuerung immer sichtbar. Mit der Spacetaste kommt man von einer Funktion zurück auf die Hauptmenuleiste.

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 Menuzweiges zwei Geschwindigkeiten vorwählen; also einmal langsam, dann wieder schnell (4-fache Geschwindigkeit) auf dem Bild herumfahren. Mit "Space" übt man eine Option aus. Diese (Draw, Un/plot, Circle, Scroll, Text) sind zudem recht flott - trotz Basic. Wie kommt das ?

Nun einmal wurde die schnelle INKEY$-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. Überlaufsicherung 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

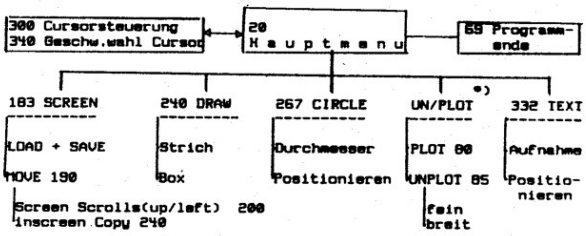
(Zeile 342) Hier ist der Variableninput angewandt; die Inputantwort S = Slow oder F = Fast ist bereits mit einer vorbelegten Variable (s. Zeile 17) vorbereitet. Gibt der User S oder F ein, erhält G (wie Geschwindigkeit) als Wert 1 bzw. 5

(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.

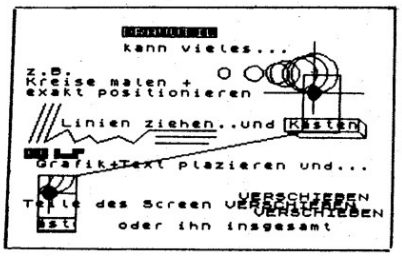
Genug der Tricks, wenden wir uns wieder dem Programm im Ganzen zu. Zeigen wir die Funktionen und ihr Zusammenspiel, ehe der Rest der Dokumenation folgt.

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



Das Zusammenspiel der Funktionen interessiert den eigentlichen Benutzer nur mittelbar. Für ihn ist wichtig zu wissen, wie er DRAWUTIL zu bedienen hat. Daher kurz einige wichtige Benutzungshinweise:

- Start mit RUN,um die MC's zu laden; Restart ins Programm mit GOTO 20
- Rücksprung auf die Steuerleiste und eine höhere Ebene mit SPACE, dito um zwischendurch den Cursor zu deaktivieren
- Cursor-Geschwindigkeit wird bei den Hauptästen vorgewählt und bleibt gesetzt;notfalls kann man mit UN/PLOT zwischendurch 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 Berdich 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 Screen sind in Microdrive-Syntax (eine Änderung für Tape ist einfach)

```
   6 REM D R A W U T I L
         UTILITY Z. BEARBEITUNG
             D.BILDSCHIRMS (ZEIC
     HNEN,VER-      SCHIEBEN,SCRO
     LL + TEXT)
   8 DEF FN Z()=INT (175-Y)/8-.5
   9 DEF FN S()=INT (X/8)+.5
  10
  11 REM LADEN MC-ROUTINEN F.SCR
     OLL-UP + SCROLL-LEFT
  12 LOAD *"M";1;"SCRUP"CODE
     REM ADRESSE 32600
  13 LOAD *"M";1;"SCRLEFT"CODE
     REM ADRESSE 32700
  14 PRINT #1;AT 0,0; FLASH 1;"*
      * * D R A W U T I L * * *"
     ; FLASH 0;"OPTIONS/PARAMS="
     ; ""CURSORSTART=ENTER"
     PAUSE 200
  16 REM INITIALISIEREN
  17 LET F=5
     LET S=1
     LET G=F
  18 PLOT 4,4
     DRAW 247,0
     DRAW 0,167
     DRAW -247,0
     DRAW 0,-167
  19 LET X=128
     LET Y=88
  20 REM HAUPTMENUE
  21
  22 PRINT #1;AT 0,0;"SCREEN$ DR
     AWCIRCLEPLOTUNPLOTTEXTENDE=
     >BUCHSTABE(SONST CURSOR)
     "
  23 PAUSE 0
     LET S$=INKEY$
     IF S$>"Z" THEN RANDOMIZE US
     R 4317
```



```
  24 IF S$="S" OR S$="D" OR S$="
     P" OR S$="T" OR S$="E" OR S
     $="U" OR S$="C" THEN GO TO
     26
  25 GO SUB 300
     GO TO 20
  26 IF S$="E" OR S$="S" THEN GO
      TO 28
  27 PRINT #1;AT 0,0;" ====>> ";
     GO SUB 342
  28 GO TO CODE S$
  67 GO SUB 267
     GO TO 20
  68 PRINT #1;AT 0,0;"[illegible]=>DRAW
     STRICHBOX(KASTEN)",
     PAUSE 0
     LET S$=INKEY$
     GO TO 168
  69 STOP
  80 PRINT #0;AT 0,0;"[illegible]",,"
     CURSOR ";"[illegible]" AND G=S;"[illegible]
     [illegible]" AND G=F;" (RETURN=SPACE
     )",
     GO TO 25
  83 GO SUB 183
     GO TO 22
  84 PRINT #0;AT 0,0;" ",,,,
     GO SUB 332
     GO TO 20
  85 PRINT #1;AT 0,0;"[illegible]",,
     "FEIN" AND G=S;"BREIT" AND
     G=F;" (RETURN=SPACE)",
  88 GO SUB 300
     GO TO 22
 168 PRINT #1;AT 1,0;"(RETURN=SP
     ACE)",,
     IF S$<>" " THEN GO SUB 240
 169 IF S$=" " THEN GO TO 25
 170 GO TO 68
 183 PRINT #1;AT 0,0;"[illegible]
     [illegible]",,"LOADSAVE BILD MOVE",
     PAUSE 0
     LET S$=INKEY$
     IF S$=" " THEN RETURN
 184 IF S$="M" THEN GO SUB 190
 185 IF S$="L" THEN INPUT AT 1,0
     ;"NAME ? ";S$
      LOAD *"M";1;S$SCREEN$
 186 IF S$="S" THEN INPUT AT 1,0
     ;"NAME ? ";S$
      SAVE *"M";1;S$SCREEN$
 187 GO TO 183
 190 PRINT #1;AT 1,0;"";AT 1,0;"
     SCREEN INSCREEN(RETURN=SP)
     "
     PAUSE 0
     LET S$=INKEY$
     IF S$=" " THEN RETURN
 192 IF S$="S" THEN PRINT #1;AT
     1,0;"[illegible]=UP [illegible]=LEFT (SPACE=RE
     TURN) "
      PAUSE 10
      GO SUB 200
 193 IF S$="I" THEN GO SUB 220
 194 GO TO 190
 200 PAUSE 0
     LET I$=INKEY$
     IF I$=" " THEN OVER 0
      RETURN
 201 OVER 0
     PLOT 0,0
     DRAW 255,0
 202 OVER 1
     PLOT 0,0
     DRAW 255,0
 203 IF I$="<" THEN RANDOMIZE US
     R 32700
 204 IF I$="↑" THEN RANDOMIZE US
     R 32600
 206 GO TO 200
```

```
221 LET S$="B"
    OVER 1
    GO SUB 240
    PAUSE 100
    OVER 1
    GO SUB 250
222 LET XE=X
    LET YE=Y
224 PRINT #0;AT 0,0;"VERSCHIEBE
    -KOORD.",
    GO SUB 300
228 IF ((X+(XE-XA))>250) OR ((Y
    -(YA-YE))<4) THEN PRINT #0;
    AT 1,14;"KOORD.FEHLER  =>"
     PAUSE 10
     GO TO 220
236 FOR M=YA TO YE STEP -1
     FOR N=XA TO XE
      IF POINT (N,M) THEN PLOT
      N-(XA-X),M-(YA-Y)
237  NEXT N
    NEXT M
    RETURN
240 REM DRAW+BOX
241 PRINT #0;AT 0,0;"CURSOR=>1.
    KOORD.",
    GO SUB 300
    LET XA=X
    LET YA=Y
242 PRINT #0;AT 0,15;" 2.KOORD.
    "
    GO SUB 300
243 PRINT #0;AT 1,0;"ALLES OK ?
     JA=SPACE",
244 PAUSE 0
    LET I$=INKEY$
    IF I$<>" " THEN GO TO 241

250 PLOT XA,YA
    IF S$="D" THEN DRAW X-XA,Y-
    YA
     RETURN
252 IF S$="B" THEN DRAW X-XA,0
     DRAW 0,Y-YA
     DRAW XA-X,0
     DRAW 0,YA-Y
     PLOT XA,YA
255 RETURN
267 INPUT AT 0,0;"CIRCLE =>DURC
    HMESSER  ?";M
    GO SUB 300
    IF (X+M)<=255 AND (X-M)>=0
    AND (Y+M)<=175 AND (Y-M)>=0
    THEN CIRCLE X,Y,M
268 RETURN
297 REM CURSOR
299


300 OVER 1
    GO SUB 322
    PAUSE 5
    GO SUB 322
    IF INKEY$=" " THEN OVER 0
     RETURN
302 IF S$="P" OR S$="U" THEN LE
    T XE=X
     LET YE=Y
303 LET X=X+G*((INKEY$="8")-(IN
    KEY$="5"))
    LET X=X+G*((X<F)-(X>250))
304 LET Y=Y+G*((INKEY$="7")-(IN
    KEY$="6"))
    LET Y=Y+G*((Y<F)-(Y>170))
306 IF S$="P" THEN GO SUB 324
307 IF S$="U" THEN GO SUB 325
308 GO TO 300
321 REM CURSOR-KREUZ
```



```
322 PLOT X,Y+F
    DRAW 0,-10
    PLOT X-F,Y
    DRAW 10,0
    PLOT X,Y
    RETURN
324 OVER 0
    PLOT X,Y
    DRAW XE-X,YE-Y
    PLOT X,Y
    RETURN
325 IF G=F THEN GO TO 328
326 OVER 1
    IF POINT (X,Y) THEN PLOT X,
    Y
327 RETURN
328 OVER 0
    PRINT AT FN Z(),FN S();" "
    RETURN
332 INPUT AT 0,0;"[illegible]EINGABE
    =>ENDE:RETURN ";I$
    IF I$="" THEN RANDOMIZE USR
     4317
     RETURN
334 PRINT #1;AT 1,0;"TEXTPOSIT.
    M.CURSOR",
    GO SUB 300
337 IF LEN I$>(31-FN S()) THEN
    LET I$=I$( TO 31-FN S())
338 PRINT AT FN Z(),FN S();
339 PRINT I$
    LET Y=Y-8*(Y>=12)
340 GO TO 332
342 PRINT #0;AT 1,0;"CURSOR [illegible]LO
    W [illegible]AST",
    PAUSE 0
    LET G=VAL INKEY$
344 RETURN
```

Programmbeschreibung

Module

Zeile 6 - 19 Initialisieren
- + Funktionen definieren
- + Laden MC's (Scroll up + left)
- + Erstmenu und Rahmen

Zeile 20 -67 Hauptmenu
- + Steuerleiste anzeigen,Abfrage u. Ansteuern Optionen (per "Codesprung"):

Zeile 67->267	
CIRCLE	Durchmesser + Start festlegen Kreis zeichnen
Zeile 68->240	
DRAW	Wahl Strich oder Box Positionierungen Erzeugen Figur
Zeile 80->300	
PLOT	in 'Cursor' integriert
Zeile 83->183	
SCREEN	Unterabfrage

```
                    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 Machine 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)
+ Säulengrafik 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 (sgn. Monatsposten)
+ leichtes Update mit der Möglichkeit, Werte "zeilenweise" vor-
  und rückzutragen
+ einfache Werteingabe durch vorgegebene Bezeichner

Gerade die Anwendung einer speziellem 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 Screentricks (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 Bezeichnern 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 Menuezweig 'Jahresübersicht' werden sogar zwei verschiedene Diagramme zusammengeschoben, um die dazugehörigen Werte auf dem unteren Bildschirm mit darzustellen.
Bei alle dem wird Bedienungskomfort groß geschrieben:

Neben Selbstverständlichkeiten wie automatischer Skalierung aller Werteplots, stellengerechter Tabulierung bei Zahlen (incl. Rundung) und maßstabsgerechter Grafik wurde dem sgn. Benutzerhandling verstärkt Rechnung getragen. Dazu ein paar Beispiele :
- bei "Anwahl" von Einzelposten (s.Option EINZELENTWICKLUNG, Menueast JAHRESUEBERSICHT) wird dem Benutzer die Skala aller Bezeichner im INPUT-Bereich präsentiert; mittels Cursor-Tasten fährt 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. MONATSUEBERSICHT), 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 MONATSUEBERSICHT 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 geschwärzt. 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 odie 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 Monatskolumnen sind abwechselnd normal und "bright" gehalten. So ist leicht zu verfolgen, zu welchem Monat ein Plotwert gehört.

Wie ist das Programm aufgebaut ?

Die übliche Menuesteuerung zeigt als Hauptäste des Progamms:
```
INITIALISIERN
UPDATE MONAT
AKT.UPDATE-STATUS
MONATSUEBERSICHT
ferner - SAVE,LOAD und ENDE
```

INITIALISIEREN dient der Besetzung der Params, Datenarrays werden dimensioniert und mit Erstwerten besetzt, sowie das Programm mit den nötigen Initialwerten versehen. (Näheres zu diesem und den andren Menuezweigen 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 upgedateten Monat zurück zu schalten und Korrkturen 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 Updatedurchgang 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, wieweit man bereits upgedatet hat.
MONATSUBERSICHT ist wie die JAHRESUEBERSICHT 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 Werteplot 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 ersterer 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 Etatposten dran,der "unten" hinkommt. Skalierung und Beschriftung richtet sich nach dem jeweilgen Maximum, sodaß immer eine maßstabsgetreue 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 Menuezweige 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 Datenarry 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

M o d u l e
(Nummmer = Programmzeile)

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

50 - 69 UPDATE
  52- 65 Eingabe Monatszahlen:

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

20 - 44 AKT.UPDATE-STATUS
  20 - 24 Layout:
    Scroll,Kopf,Einnahmen/Ausgabentrennung
  25 - 27 Andruck Besetzung:
    Hell/Dunkelsteuerung,Besetzungscheck,Rahmenplot

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

80 - 99 JAHRESUEBERSICHT

  85 - 91 Summenentwicklung:
    85-88 Grafik Monatssummen + -salden
    88-91 Grafik mit Absolutzahlen

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

U P R O 's

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

Funktionen

FN T$(X) Tabulieren der gerundeten Zahl X mittels ausgerichteter 'leading spaces'
FN M$(X) Konversion Monatszahl in Monatsbezeichnung aus M$-String

V a r i a b l e n l i s t e

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)      Monatssaldo Einn./Ausg.

I$(N+3,15)Bezeichner eines jeden Postens(15 Stellen lang)
```

Stringfelder

```
M$        Monatsnamen (alle 12 Monate = Länge 12*9)
J$        Jahr
I$        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.Werteplot
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 HEIMBUDGET
11
12
13 GO TO 100
14
15 DEF FN T$(X)="          "( TO
   8-LEN STR$ (INT X))+STR$ I
   NT (ABS X+.5)
16 DEF FN M$(X)=M$((X-1)*9+1 T
   O X*9)
18 REM AKTUELLER STATUS
19
20 LET I$="EINNAHMEN"
21 GO SUB 32
   PRINT
   GO SUB 34
   DRAW 0,-8
22 FOR X=1 TO N
23  IF SCREEN$ (21,0)<>" " THE
    N GO SUB 32
```



```
24 IF X=E+1 THEN LET I$="AUSGA
   BEN"
   GO SUB 34
   DRAW 0,-8
25 PRINT TAB 0;T$(X);TAB 16;
   LET F=1
   FOR Y=1 TO 12
    LET I$="
    IF N(Y,X) THEN LET I$="*"
26  PRINT BRIGHT F;I$;
    LET F=NOT F
   NEXT Y
   DRAW 0,-8
27 NEXT X
   DRAW 0,-8
   DRAW 112,0
   PAUSE 0
   RANDOMIZE USR 3330
   RETURN
31 REM KOPF STATUSBERICHT

32 RANDOMIZE USR 3330
   CLS
   PRINT TAB 4;"[redacted]
   [redacted]";"POSITIONEN" "  M O N
   A T E","  J[redacted]
   PLOT 0,143
   DRAW 250,0
   PLOT 130,150
   DRAW 0,-16
   RETURN
33 REM EINSCHUB("EIN-/AUSGABEN
   ")
34 PRINT BRIGHT 1;I$; BRIGHT
   0;TAB 16;
   LET F=1
   FOR Y=1 TO 12
    PRINT BRIGHT F;" ";
    LET F=NOT F
   NEXT Y
   RETURN
35 REM SUMMENGRAFIK JAHR

36 LET F=FAK*N(1,X)
37 PLOT 71,95-80*Y+F
   FOR Y=2 TO 12
    DRAW 16,FAK*(N(Y,X)-N(Y-1,
    X))
   NEXT Y
   RETURN
38 FOR X=1 TO 12
    LET N(13,Y)=(N(13,Y) AND (
    X>1))+N(X,Y)
    IF N(X,Y)>MAX THEN LET MAX
    =N(X,Y)
39 NEXT X
   RETURN
40 RANDOMIZE USR 3330
   CLS
   LET I$=" 1[redacted]
   [redacted]"

41 FOR X=0 TO 19
    PRINT TAB 0;
    FOR Y=1 TO 6
     PRINT "  "+CHR$ 19+CHR$ 1
     +" ";
    NEXT Y
   NEXT X
   PRINT AT X,8;I$;AT 0,8;"[redacted]
   [redacted]";U$
43 PLOT 71,175
   DRAW 0,-80
   DRAW 164,0
   DRAW 0,+80
   DRAW -164,0
   DRAW 0,80
44 RETURN
45
```

Screentricks (im Programm)

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. Erläutern 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 eine dritten Punkt werden eine Reihe sonstiger Programmkniffe 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-Zahlenn 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-Nachkomma-Stellen 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 M(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 zuguta, 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, wenngleich 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 Pixeldistanzen "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 Programmtricks

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

EineIF-like Progammierung 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) Ansprungzeile zur Programmfortsetzung
```

Derselbe Effekt wird in Zeile 62 in nur e i n e m 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 INKEY$ = "" THEN PAUSE 0
IF INKEY$ <> "" 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

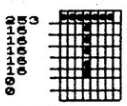
Zeichen werden beim SINCLAIR (und anderen Heimcomputern) pünktchenweise als Matrix generiert. Jeder Buchstabe, jedes Sonderzeichen und jede Zahl entstehen aus einer 8 x 8 - Matrix. Die Besetzung der Pixel (= schwarzer Punkt höchster Auflösung auf dem Bildschirm) wird von der Information gesteuert, die 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 sgn. 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 < äußerst einfach (mit BIN) die Besetzung der Zeichenmatrix 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 generiert 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 a 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 I$ gespeicherten Zeichens bekommt man daher so:

```
(CHARS ist gleich Adresse 23606 + 23607)
CHARSTART = PEEK 23606 + 256*PEEK 23607
    CHAR = 8 * CODE I$ + 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) enschlüsselt 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 vrgeschoben 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 T$; da um Faktor 8 je Pixel vergrößert wird, ist N*8 einzusetzen..

```
 8 REM BIG PRINT
10 REM 8-fach Vergroesserung v
   on Buchstaben
11 LET start=10
12 LET i$="Look"
13 FOR n=0 TO LEN i$-1
14   LET t$=i$(n+1)
15   LET zeile=start
     PRINT AT zeile,n*8;
16   LET char=8*CODE t$+PEEK 23
     606+256*PEEK 23607
18   FOR y=0 TO 7
20     LET x=PEEK (char+y)
22     FOR z=7 TO 0 STEP -1
24       IF (x-2↑z)>=0 THEN  PRI
       NT "■";
       LET x=x-2↑z
       GO TO 28
26       PRINT " ";
28     NEXT z
30     LET zeile=zeile-z
31     PRINT AT zeile,n*8;

32 NEXT y
34 NEXT n
```

Look

Wem 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 + 46). 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 Buchstabenvergroesse-
         rung 2-fach
12

14 REM Startzeile Zeile 10
16 LET start=10
18 LET i$="Test 2 fach"
   REM Teststring
20 LET zeile=175-start*8
22 LET charstart=PEEK 23606+25
   6*PEEK 23607
24

26 FOR n=0 TO LEN i$-1
28   LET t$=i$(n+1)
30   LET zei=zeile
     LET sp=n*16
32   LET char=8*CODE t$+charsta
   rt
34   FOR y=0 TO 7
36     LET x=PEEK (char+y)
38     FOR z=7 TO 0 STEP -1

40 IF (x-2↑z)>=0 THEN PLOT sp,
   zei
     PLOT sp,zei+1
     PLOT sp-1,zei
     PLOT sp-1,zei+1
     LET x=x-2↑z
42 LET sp=sp+2
44 NEXT z
46 LET zei=zei-2
   LET sp=sp-16
48 NEXT y
50 NEXT n
```

Test 2 fach

Weitere Manipulationen

Mit der angelistete Plotroutine 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 Scriftverbreiterung.

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.

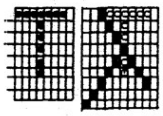
Um den alternierenden Plot zu erreichen, wird b als Ein/Aus-Schalter geführt (vgl. Zeile 43). Dadurch wird etwas wie "enlarged condensed" bei Normalpapierdurckern realisiert. Man experimentiere ruhig ein wenig mit der folgenden Routine.

```
10 REM BREITSCHRIFT

16 INPUT "startzeile";start
   LET sp=0
18 INPUT "22-25 Zeichen/Zeile
   ";i$
20 LET zeile=175-start*8
22 LET charstart=PEEK 23606+25
   6*PEEK 23607
```

```
25 FOR n=0 TO LEN i$-1
26  LET t$=i$(n+1)
30  LET zei=zeile
31  IF t$=" " THEN LET sp=sp+6
    GO TO 50
32  LET char=8*CODE t$+charsta
    rt
33  LET b=0

34 FOR y=0 TO 7
35  LET x=PEEK (char+y)
36  FOR z=7 TO 0 STEP -1
40   IF x THEN IF (x-2↑z)>=0 T
     HEN PLOT sp,zei
      PLOT sp+b,zei
      LET x=x-2↑z
42   LET sp=sp+1+b
43   LET b=NOT b
44  NEXT z
45  LET zei=zei-1
    LET sp=sp-12
46 NEXT y
50 LET sp=sp+11
   IF sp>239 THEN LET sp=0
   LET zeile=zeile-8
   IF zeile<0 THEN LET zeile=
   175
55 NEXT n
```

statt hoch + gross BREIT

Selbstverständlich kann man noch andere Effekte erzielen. Das "Pixelverfahren" ermöglicht beispielsweise, Schrift zu drehen. Im folgenden Listing wurde zu dem Zweck die Zeilen/Spaltenfortzählung asymmetrisch gestaltet. Die Spalte wird um eins erhöht, gleichzeitig jedoch wird die Zeile bei jedem Pixel nach unten verschoben. Und der Effekt ? Wir haben flugs Schrägschrift auf dem Schirm. Wird jedes Pixel nach unten verschoben, erhält die Schrift "Schräglage" (s. Zeile 42).



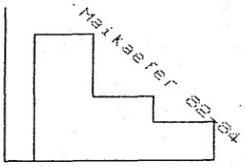
Zu beachten ist noch, daß bei einer neuen Zeile der Pixelabstand nicht um 8, sondern nur um 7 erhöht wird. Die Spalten werden entsprechend nicht um -8, sondern um -9 rechts vom Rand positioniert, wie das kleine Schaubild zeigt.

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 [illegible]
11
12
14 INPUT "Startzeile ?";zeile
16 INPUT "Startspalte ?";spalt
   e
```



```
17 PRINT AT zeile,spalte; OVER
   1;"■";
   INPUT "kein Ueberlaufcheck!
   Alles o.k.?=>RETURN";i$
   PRINT CHR$ 8; OVER 1;"■"
   IF i$<>"" THEN GO TO 14
18 INPUT "Zeichen ";i$
20 LET zeile=175-zeile+8
21 LET sp=8*spalte+8
22 LET charstart=PEEK 23606+25
   6*PEEK 23607
24
25 LET zei=zeile
26 FOR n=0 TO LEN i$-1
27  LET t$=i$(n+1)
32  LET char=8*CODE t$+charsta
    rt
34  FOR y=0 TO 7
35   LET x=PEEK (char+y)
36   FOR z=7 TO 0 STEP -1
40    IF x THEN IF (x-2↑z)>=0
      THEN PLOT sp,zei
       LET x=x-2↑z
42    LET sp=sp+1
      LET zei=zei-1
44   NEXT z
46   LET zei=zei+7
     LET sp=sp-9
48  NEXT y
49  LET sp=sp+15
    LET zei=zei+1
55 NEXT n
```



Wird ein Zeichen ganz gedreht, kommt es um 90 Grad verkantet auf den BUldschirm. Die Vorteile einer solchen VERTIKALSCHRIFT, mit der wir uns im Folgenden beschäftigen, sind beträchtlich:

> Skalen können koordinatengerecht beschriftet werden

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

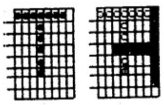
> Fuktionen lassen sich "open endad" (immer weiter nach unten fortgeschrieben) darstellen

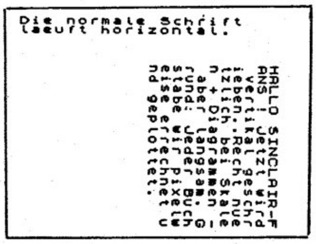
> Der Bildschirm bekommt ein angenehmes 32 Zeilen-Format

> Durch Hardcopies auf Normalpapierdrucker lassen sich bei mehrere 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.

Die folgende Routine wandelt auf vertrautem Pfad. Der Trick "Vertikal" läuft so: Nach jedem Zeilenplot wird nicht die Spalte um 1 verschoben, sondern die Spalte (s. Zeile 42). Es wird also nach unten geplottet. Bei einer neuen Pixelreihe wird die Zeile wieder aufgefüllt und die Spalte um 1 nach links verschoben (s. Zeile 40 + 46). Kurzum - die bisherigen Zeilen-/Spalten-Funktionen sind vertauscht. Das Ergebnis sieht dann so aus.






```
10 REM [illegible]
11

12

14 INPUT "Startzeile ?";zeile
16 INPUT "Startspalte ?";spalt
e
18 INPUT "bis 704 Zeichen(=Sei
te)";i$
20 LET zeile=175-zeile*8
21 LET sp=8*spalte
22 LET charstart=PEEK 23606+25
6*PEEK 23607
24

25 LET zei=zeile
26 FOR n=0 TO LEN i$-1
28   LET t$=i$(n+1)
30   IF t$=" " THEN GO TO 50
32   LET char=8*CODE t$+charsta
rt

34 FOR y=0 TO 7
36   LET x=PEEK (char+y)
38   FOR z=7 TO 0 STEP -1
40     IF x THEN IF (x-2↑z)>=0 T
HEN PLOT sp,zei
       LET x=x-2↑z
42     LET zei=zei-1
44   NEXT z
46   LET zei=zei+8
     LET sp=sp-1
48 NEXT y
49 LET sp=sp+8
50 LET zei=zei-8
   IF zei<8 THEN LET zei=zeile
    LET sp=sp-8
    IF sp<8 THEN LET sp=255
55 NEXT n
```


Die Generierung eines neuen Zeichensatzes

Wie ist vorzugehen ?

Der bisherige Weg ist ebenso umständlich wie langsam. Wenden wir uns daher der Generierung von "Sets" zu, auf welche der Rechner blitzschnell zugreift. Dazu bedarf es zweier Schritte:

1.Generierung eines neuen Zeichensatzes (als Bytefolge abgelegt)

2.Aufruf desselben durch "Verbiegen" von CHARS und Andruck

Die einzelnen Schritte zur Lösung von Vertikalschrift sind wie folgt:

- Reservieren eines Bereichs, wohin das neue Set kommt
- Umsetzen bisheriger Bitpositionen in eine eigene 8 x 8 Buchstabenmatrix
- Einpoken der Matrix (pro Buchstabe) an die neue Adresse

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 8 Bytes ergibt 768 Acht-Bit-Positionen; wozu 8 Wertigkeiten pro Pixel zu errechnen sind, was 6144 Arithmetrikoperationen ausmacht. Kein Wunder also, wenn die Generierung des Vertikal-Sets ca. 15 Minuten in Basic dauert. Das Ende der Kaffeepause wird mit ein paar Beeps signalisiert...

```
10 REM [illegible]
11

12 REM GENERIEREN VON VERTIKAL
   -BUCHSTABEN(Aendern des Cha
   rachtersatzes)
13

15 CLEAR 64599
   REM Destination
16 LET m=64600
   LET n=256+PEEK 23606+256*PE
   EK 23607
   REM Quelle
17

18 REM Umbesetzen Bitpositione
   n in Tabelle n(8);danach Ei
   npoken der gedrehten Charac
   ter
19

20 FOR n=n TO n+768 STEP 8
22   DIM n(8)
24   FOR y=0 TO 7
26     LET x=PEEK (n+y)
28     FOR z=7 TO 0 STEP -1
30       IF x THEN IF (x-2↑z)>=0
        THEN LET x=x-2↑z
          LET n(8-z)=n(8-z)+2↑y
32     NEXT z
     NEXT y
     LET z=1
34

36   FOR m=m TO m+7
       POKE m,n(z)
       LET z=z+1
     NEXT m
   NEXT n
```

```
30 FOR n=1 TO 5
   BEEP 1,0
   BEEP 1,8
   BEEP .6,3
   BEEP .5,2
   BEEP 1,0
   NEXT n
   PRINT FLASH 1;"Generierung
   beendet"
40 REM Umlegen des Char.Sets m
   itPOKE 23606,88:POKE 23607,
   251(ZURUECK =) 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 "VCHAR" CODE 64600,768
```

Um nun an die Vertikalschrift zu kommen, muß Systemvariable CHARS auf Adresse 64600 "umgebogen" werden:

```
POKE 23606,88: POKE 23607,251
```

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

Arbeiten mit VCHAR (Vertikalcharcters)

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 VCHAR einzulesen. Das geschieht mit LOAD VCHAR,CODE. Nach Verbiegen des Char-Vektors (s.o.) kann dann ein vorhandener Text vertikal geprintet werden. Die Printroutinene beinhaltet zwei Vorschläge:

1.Anzeige eines vorhandenen Textes (in A$) mit eigener (auf "vertikal" umgestellten) Input-Aufforderung. - Der erste Teil des Printprogramms richtet Spalten und Zeilen neu aus. Da der Textstring A$ nicht einfach - mit PRINT A$ - 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 Screenseiten 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
    ikalcharakters
 12
 14 REM memo 1.CLEAR 64599 2.VC
    HARS-CODES laden 3.Textstri
    ng A$ laden 4.Variable CHAR
    S auf Adr.64600 umlenken
 16
 18 LET s=31
    LET z=0
    LET i$=""
 20 FOR n=1 TO LEN a$
 22  PRINT AT z,s;a$(n)
 24  LET z=(z+1) AND (z<21)
 26  LET s=s-(z=0)
 28  IF s<0 THEN INPUT "NRUTER!
     etiesS euen";i$
      IF i$="" THEN CLS
       LET s=31
       GO TO 32

 30 IF i$<>"" THEN LET n=LEN a$
 32 NEXT n
 34
 36 REM Tabellenloesung
        (Vorteil
    Moegl.,den Text zu 'splitte
    n' + in mehreren Schueben a
    uszudrucken)
 38 LET dim=44
 40 DIM t$(dim,32)
 42 LET n=1
 44 FOR s=32 TO 1 STEP -1
 46  FOR z=1 TO dim
 48   LET t$(z,s)=a$(n)
      LET n=n+1
 50   IF n>LEN a$ THEN LET z=di
      m
       LET s=1
 52  NEXT z
 54 NEXT s

 56 INPUT "PRINT ? =>RETURN";i$
    IF i$<>"" THEN GO TO 62
 57 LET s=22
    LET n=1
 58 FOR n=n TO s
     PRINT t$(n)
    NEXT n
 60 IF dim>n THEN COPY
     LET s=dim
     GO TO 58
 62 COPY
110 POKE 23606,0
    POKE 23607,60
```

```
Dieser Test zeigt,wie  Vertikalbuchstaben ein
gesetzt werden.Man dre he also den Bildschirm
 und geniesse ein voel lig neues Screen-Gefue
hl.Die Einsatzmoeglich keiten sind aber noch
vielseitiger:beispiels weise ist es nun moegl
ich Druckausgaben unbe grenzter Breite zu rea
lisieren.Man muss nur  den Text in mehreren S
chueben ausdrucken.Abe r auch Grafiken mit 25
6 pixels koennen verti kal in beliebiger Laen
ge erstellt und beschr iftet werden..Und Viel
es 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..

Wem 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 "geshiftet". Ein neues Pixel umzeichnet zum vorhandenen die Buchstabenkontur und erbringt so die propere Schrift.

```
 990 CLEAR 64568
 991 FOR f=65338 TO 65365: READ
a: POKE f,a: NEXT f
 992 RANDOMIZE USR 65338
 993 DATA 33,0,61,17,57,252,1,0,
3,126,203,47,162,18,19,35,11,121
,176,32,244,33,57,251,34,64,92,2
01
```

Zum Schluß noch einen Hinweis: Wer wieder seinen normalen Zeichensatz haben will - auch im 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 erfolt zunächst per INPUT. Hinter dem Befehl verbirgt sich mehr als man gemeinhin ahnt - ein kompletter Zeileneditor, der von System zur Verfügung gestellt wird. Arbeitet man mit Eingabemasken, kann man nur neidisch auf die guten Insert und Deletefunktionen 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 Gedanke 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 ?";N
```

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

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

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

```
 10 REM [illegible]

 16 LET z=6
    LET s=14
    LET len=9
    DIM i$(len)
 17 PAUSE 0
 20 PRINT AT 2,3;"************
    **********";AT 3,3;"*  I N
    P U T  *";AT 4,3;"*_____
    _____*";AT 5,3
    +"*                  *";A
    T 6,3;"* Umsatz :
     *";AT 7,3;"*        .....
    .....*";AT 8,3;"**********
    **********"
 21 PRINT AT z,s;i$
 22 PAUSE 0
    REM  Loesung mit Input-Sim
    ulation
    DIM i$(len)
    LET p=1
    LET s=14
 24 PRINT AT z,s;i$;AT z,s-1+p;
    OVER 1; FLASH 1;" ";
 26 PAUSE 0
    LET x=CODE INKEY$
    LET p=p-(x=8)+(x=9)
 29 IF p<1 THEN LET p=1
 30 IF x=13 THEN GO TO 34
 31 IF x>31 THEN LET i$(p)=CHR$
    x
 32 LET p=p+(x>31)
    IF p>len THEN LET p=len
 33 GO TO 24
 34 PRINT AT z,s;i$
```

```
**********************
*  I N P U T  *      *
*_____________*      *
* Umsatz : 1234       *
*                     *
**********************
```

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	I$ - 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 SCREEN$-Funktion)
>volles Insert und Delete, solange die Eingabe nicht abgeschlossen ist

```
 10 REM inscreen- INPUT
12 LET X=15
   LET Y=11
   LET LA=5
14 LET X0=15
   LET XN=X0+LA-1
16 PRINT AT Y,X;".........";( T
   O LA)
18 LET FLASH=30
   LET INSERT=40
   LET DELETE=50
20 GO SUB FLASH
21 PAUSE 0
   LET I$=INKEY$
22 IF CODE I$=13 THEN STOP
24 IF I$>=" " THEN GO SUB INSE
   RT
25 LET X=X+(X<XN AND CODE I$=9
   )
26 LET X=X-(X>X0 AND CODE I$=8
   )
27 IF CODE I$=12 THEN GO SUB D
   ELETE
   GO TO 26
28 GO TO 20
29
30 IF X>X0 THEN PRINT AT Y,X-1
   ;SCREEN$ (Y,X-1)
32 PRINT AT Y,X; FLASH 1;SCREE
   N$ (Y,X); FLASH 0;SCREEN$ (
   Y,X+1)
   RETURN
40 FOR I=XN-1 TO X STEP -1
41  PRINT AT Y,I+1;SCREEN$ (Y,
    I);
42 NEXT I
43 PRINT AT Y,X;I$
44 LET X=X+(X<XN)
   RETURN
50 FOR I=X TO XN-1
51  PRINT AT Y,I;SCREEN$ (Y,I+
    1);
52 NEXT I
54 PRINT AT Y,XN;"."
55 LET I$=CHR$ 8
   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 Vorgabelänge LA sein muß.
- Die Screen$-Funktion (Zeile 30) printet das Zeichen so, wie es vorliegt, vor die aktuelle Stelle; dann wird das aktuelle Zeichen angedruckt, Flash an- und ausgeschaltet und das Folgezeichen geprintet. Sinn: Synchronisation der Positionierung und damit richtige Anzeige nach Delete/Insert.
- Bei Insert (Zeile 40) druckt die Schleife rückwärts - bis zur aktuellen Stelle; dann kommt das neue Zeichen, und der Spaltenzähler wird erhöht. Da ein Zeichen hinzukommt, startet der Andruck vor der letzten bisherigen Datenstelle (XN - 1); gezeigt wird diese jedoch an "alter" Position (I + 1). Damit ist Platz für das nächste neue Zeichen geschaffen..
- Delete läuft ähnlich. Da ein Zeichen wegfällt, zählt die Schleife (Zeile 50) bis zur vorletzten Datenstelle (XN -1).Jedes Folgezeichen wird von rechts her vorpositioniert (I + 1). I$ wird vorgesetzt; der Punkt am Ende signalisiert, daß ein Zeichen frei geworden ist. (Merke: Bei Code-12 enthält I$ tatsächlich dieses Zeichen, die Spalte muß folglich um 1 reduziert werden, damit die Spaltenanzeige wieder stimmt)
- Eine Überlaufsicherung 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. Erstere 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 [illegible]
 11
 12 REM "ERSATZLOESUNG "(STATT
    MC)
 13 FOR N=0 TO 21
     FOR M=0 TO 31
      PRINT AT N,M;"▮";
     NEXT M
    NEXT N
 14 INPUT AT 22,0;AT 0,0;"?";X$
 16 IF X$<>"" THEN GO TO 14
```

```
20 REM MASCHINENCODE-LOESUNG
21
22 CLEAR 29999
24 LET CHECKSUM=0
26 FOR X=3E4 TO 3E4+45
    READ Y
    POKE X,Y
    LET CHECKSUM=CHECKSUM+Y
   NEXT X

28 IF CHECKSUM<>4531 THEN PRIN
   T "INPUT-ERROR ! TRY IT AGA
   IN"
    STOP
29 CLS
   LET X$="*"
   PRINT AT 0,0;
30 PRINT ;TAB 3;X$;"INPUT ?";
   INPUT #USR 3E4;X$
   IF X$<>"" THEN GO TO 30
31
32 DATA 151,205,1,22,38,0,62,
   2,215,164,215,164,215,56,10
   7,92,214,3,111,6,5,41,16,26
   3,56,136,96,61,79,9,62,8,21
   5,43,124,181,32,246,68,77,6
   2,158,50,65,92,201
33 REM MEMO  BEI AUFRUF OHNE H
   ASH-SIGN ERSCHEINT INPUT -P
   ROMPT MIT FUEHRENDER NULL
```

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 TEXTVER) 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, wenngleich 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 Wortlängen. Texte zu formatieren, heißt sich mit dem Problem "Spacing" auseinander zu setzen. Was ist darunter zu verstehen ? Wer formatiert, muß Wortzwischenräume berücksichtigen. Jede druckgerechte Form von Text hat zumindest sicherzustellen, daß es nicht am Rand zu Wortbrüchen kommt (engl. word wrapping). Die Spaces zwischen Worten ließen sich vermeiden - und damit die Programmierung vereinfachen -, wenn man die Wortlängen von vornherein hätte. Erster Gedanke fürs Texthandling - statt Spaces mitzuschleifen, werden an deren Stelle Wortlängen (als Codes) im Text gespeichert. Das kostet kein Byte zusätzlich und erleichtert Ausgabeoperationen erheblich, von Dingen wie Wortcursor u.ä. ganz zu schweigen. Allerdings hat die Idee einen Nachteil - Längen(de)codierung kostet Zeit...

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

"Texterbeit" anregen. - Was geschieht im Programm ?

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

Länge W o r t Länge W o r t ...

Der Nachteil, daß T$ nicht mehr direkt angedruckt werden kann ist aufgewogen durch die leichterer Formatierung (2.Programmteil ab Zeile 46). Bei vorgebener Breite (Variable BR mit Bildschirmformat 32 Zeichen/Zeile) wird einfach von Längenfeld zu Längenfeld gesprungen. Liegt die Breite im Rahmen von BR, werden alle Zeichen bis zur "Lücke" angedruckt und ein Space angefügt. Ist jedoch ein Überhang erreicht, erfolgt Zeilenvorschub und Löschung von N. Dann wird weitergezählt. Zu beachten ist, daß Textstring T$ nach jedem Zeilendurchgang abgehackt wird. Am Ende ist T$ leer. Möchte man ihn erhalten, ist vorab zu sichern (z.B. mit LET A$ = T$).

```
 8 REM        TEXTLEN
 9
   REM ------------
10 REM INPUT Text mit Wortlaen
   gen
12
14 PRINT "     Texteingabe (Ende
   mit £)"''
20 LET t$=""
   LET a$=""
   LET n=1
30 PAUSE 0
   LET i$=INKEY$
   IF CODE i$<32 OR CODE i$>12
   8 THEN GO TO 30
32 IF i$="£" THEN GO SUB 40
   GO TO 45
34 IF i$=" " THEN GO SUB 40
36 PRINT i$;
   IF i$<>" " THEN LET a$=a$+i
   $
38 LET n=n+1
   GO TO 30
40 LET a$=CHR$ (n-1)+a$
   LET t$=t$+a$
   LET a$=""
   LET n=0
   RETURN
46 PRINT '''     Formatierung'''
47 REM BREITE frei waehlbar
48 LET n=0
   LET br=32
56 LET n=1+n+CODE t$(1)
59 IF n<br THEN PRINT t$(2 TO
   1+CODE t$(1));" ";
   GO TO 64
60 PRINT
   GO TO 48
64 LET t$=t$(2+CODE t$(1) TO )
65 IF t$="" THEN PRINT
   STOP
66 GO TO 56
```

Wer den Vorteil des direkten Stringandrucks (von IS) behalten möchte, wird die folgende Routine für 'Flattersatz' zu schätzen wissen. Sie arbeitet ohne "Längen" und gestattet den direkten Andruck 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) besetzt. Dann wird von vorn " weitergehandelt". 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
AUS. BEI DIESEM SGN. FLATTERSATZ
WIRD DAFUER GESORGT, DASS KEINE
WORTBRUECHE (ENGL. WORD
WRAPPING) VORKOMMEN. DIE ROUTINE
PRUEFT DAS AS UND FUEGT AN
PASSENDER STELLE EINEN
ZEILENVORSCHUB EIN(CR=CODE 13).
VORTEIL ZUSAETZLICH - DER EINMAL
AUFBEREITETE TEXT KANN BELIEBIG
(UND SCHNELL !) OFT GEPRINTET
WERDEN...
```

```
 1000 REM [redacted]
1002
1010  INPUT "DRUCKBREITE";BR
      LET BR=BR+1
      FOR N=BR TO LEN T$ STEP BR
1020   LET N=N-(T$(N)<>" ")
       GO TO 1020+(T$(N)=" ")
1021   LET T$(N)=CHR$ 13
      NEXT N
      PRINT T$
```

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

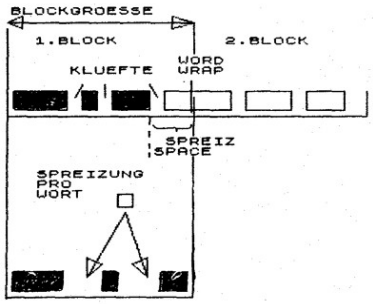
Das sauberste Druckbild liefert der Blocksatz; darunter ist die sowohl links- wie rechtsbündige Textwiedergabe zu verstehen. Blocksatz krönt gewissermaßen die Druckausgabe für den Hausgebrauch. Wer schnellere und noch bessere Formatierung haben will, wird sich in der Regel an kommerzielle Produkte (in Maschinencode) wenden. Was ist von einer BASIC-Routine BLOCKSATZ zu verlangen ? Kurzum ein bündiger Textandruck 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.

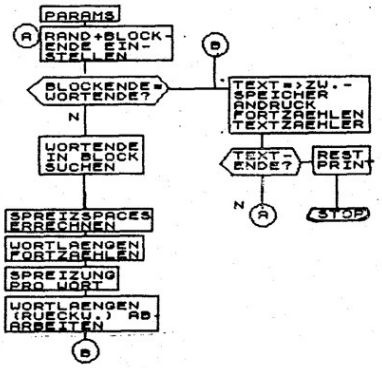
Gegeben ist zunächst ein Textstring IS; Druckbreite BR und Druckbeginn (= 1. Spalte) DB werden vom User gesetzt. Zuerst wird Zwischenstring AS mit der echten Breite dimensioniert; AS nimmt in Verlauf der ersten FOR-NEXT-Schleife die Textteilstücke auf. Da die Logik ziemlich verzwickt ist, sei erst die 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 Einfügens) 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.

PROBLOGIK "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:
- I$ = Text
- BR = Druckbreite ('Block')
- DB = Spalte Druckbeginn
- DIM A$(BR-1+DB)= dimens. Zwischenstring echter Breite
- I$ = Userinput + Wortzähler
- I = Start d. Daten in A$
- E = Ende e. Textblocks
- ST= Stelle rel. Wortbeginns
- N = Zählvariable

```
10 REM [illegible]
12
14 REM TEXTSTRING VORH.MIT T$;
   WORTLAENGEN WERDEN MIT I$ A
   USGERICHTET
16 LET BR=38
   INPUT "DRUCKBREITE?(RETURN=
   38 ST.) ";I$
   IF I$<>"" THEN LET BR=VAL I
   $
20 INPUT "LINKER RAND(DRUCKBEG
   INN:RETURN=1.ST.)";I$
   LET DB=1
   IF I$<>"" THEN LET DB=VAL I
   $
21
22 DIM A$(BR-1+DB)
   REM **S T A R T ************
   ********************
```



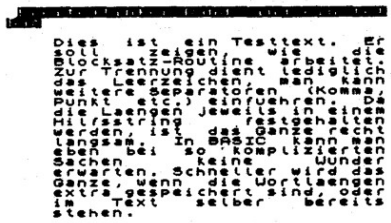
```
25 FOR N=1 TO LEN T$-BR
    LET T=DB
    LET E=N+BR-1
    IF T$(E+1)=" " THEN LET A$
    (T TO )=T$(N TO E)
     PRINT A$
     LET N=E+1
     GO TO 40
26  IF T$(E)<>" " THEN LET E=E
    -1
    GO TO 28
30  LET S=N+BR-E
    LET E=E-1
    LET I$=CHR$ 0
32  FOR E=E TO N STEP -1
     IF T$(E)=" " THEN LET I$=
     I$+CHR$ 0
      GO TO 34
33   LET I$(LEN I$)=CHR$ (CODE
     I$(LEN I$)+1)
34  NEXT E
36  LET S=S/(LEN I$-(LEN I$)1)

38 FOR E=LEN I$ TO 1 STEP -1
    LET ST=CODE I$(E)+1
    LET A$(T TO )=T$(N TO N+ST
    -2)
    LET T=T+ST+S
    LET N=N+ST
   NEXT E
39 PRINT A$
   LET N=N-1
40 NEXT N
   LET T=DB
   LET A$(T TO )=T$(N TO )
   PRINT A$
42 REM **********************
   ********** E N D E
```

[illegible]

```
Dies ist ein Testtext. Er soll z
eigen, wie die Blocksatz-Routine
arbeitet. Zur Trennung dient le
diglich das Leerzeichen, man kann
n weitere Separatoren (Komma, P
unkt etc.) einfuehren. Da die La
engen jeweils in einem Hilfsstri
ng festgehalten werden, ist das
ganze recht langsam. In BASIC ka
nn man eben bei so komplizierten
Sachen keine Wunder erwarten. S
chneller wird das Ganze, wenn di
e Wortlaengen extra gespeichert
sind, oder im Text selber bereit
s stehen.
```

[illegible]

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.) einfuehren. 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.) einfuehren. 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.) einfuehren. 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.

PROGRAMMENTWICKLUNG VON "TEXTVER"

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. Verweilen 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 Alphabeth 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 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 mal 16 (= 4 Bits) ginge. Dazu müßte man ein Steuerbyte vor nachfolgende 8 Alphazeichen legen. Es würde angeben, ob jeweils die erste oder die zweite Hälfte der 32 Möglichkeiten der Bitkobination besetzt

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

Nun zur Datenorganisation, ehe wir die Programmentwicklung von TEXTVER schildern.- Verfolgt man die gängige Maxime - ein Zeichen = ein Byte - weiter, stellt sich die Frage: Wie soll Text verarbeitet werden, als e i n 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 Stringslicing genau so gut möglich (wie TEXTVER 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 anders heißt als - man hangele 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" gängige 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 Menue mit Aufzeigen aller Funktionen; allgemein - ein einfaches Handling ohne viel Umschalterrei 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 abgehakt 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 TEXTVER 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 verlangsamt 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 Allgemeinsten 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 druckt 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 Userreaktionen, auf daß das "Bild" dann verschwinde.

```
 98 REM [illegible]
 99
100 CLS
    PRINT '''TAB 6;"[illegible]
    ■";''TAB 1;"E D I T I N G";'
    TAB 3;"CURSOR/\ \/ < >";TAB
     3;"CAPS SHIFT 0 = DELETE";
    TAB 2;"(ZEILENLOESCHEN=""NO
    T"");TAB 3;"# = INSERT";TA
    B 4;"("""=LEERZEILE)";TAB
    3;"STOP=PROGR.ENDE"''TAB 1;

100 CLS
    PRINT '''TAB 6;"[illegible]
    ■";''TAB 1;"E D I T I N G";'
    TAB 3;"CURSOR/\ \/ < >";TAB
     3;"CAPS SHIFT 0 = DELETE";
    TAB 2;"(ZEILENLOESCHEN=""NO
    T"");TAB 3;"# = INSERT";TA
    B 4;"("""=LEERZEILE)";TAB
    3;"STOP=PROGR.ENDE"''TAB 1;
    "H A N D L I N G";TAB 2;"(U
    MSCHALTER=""OR"")"''TAB 3;"
    CURSOR /\ \/ =BLAETTERN";TA
    B 3;"K-OPIEREN";TAB 3;"S-AV
    E/L-OAD";TAB 3;"P-RINT DRUC
    KER";TAB 3;"C-LEAR FILE"
102 FOR N=0 TO 21
    IF N=0 OR N=21 THEN PRINT
    AT N,0;"[illegible]
     GO TO 104
103 PRINT AT N,0;"■";AT N,27;"
    ■"
104 NEXT N
105
106 PAUSE 0
    CLS
```


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 Betrachten der Modulangebots 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, Genaueres ist in einem eigenen Schaubild aufgezeigt.

```
108 GO SUB SEITENANDRUCK
109 GO SUB GETZEICHEN
110 IF Z=197 THEN GO SUB HANDLI
    NG
     GO TO 100
112 IF Z=35 THEN GO SUB INSERT
     GO TO 108
113 IF Z=195 THEN GO SUB ZEILEN
    LOESCH
     GO TO 108
114 IF Z>31 AND Z<128 THEN GO S
    UB ZEICHENVER
     GO TO 109
116 GO SUB CURSOR
    GO TO 109
```

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)

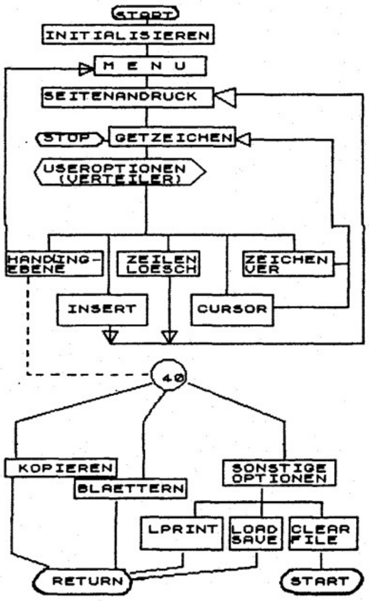
Normalerweise wird man aber nach Rücksprung beim Zeichenlesen landen, im Programm GETZEICHEN geheißen. Nach jedem Tastendruck muß notgedrungen die Abfrage folgen, ob Text- oder Steuerzeichen eingegeben wurden. Ist letzteres der Fall, wird entweder auf die HANDLING-Ebene verzweigt oder aber INSERT bzw. 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 Cursoreingaben abzuprüfen. 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

Menuanzeige
Seitenandruck
oder GETZEICHEN

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 sgn. 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.

STEUERUNGSLOGIK



Was entnehmen wir daraus ?

Neben einer Reihe Details - z.B. Zeilenlöschen wird durch NOT (= Code 195), INSERT mit dem Hash-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 werden.

```
118
119 REM INITIALISIEREN
120

121 INPUT "WIEVIEL SEITEN ?(MAX
    . 20)";MAX
    IF NOT MAX OR MAX>20 THEN G
    O TO 121
122 LET G=MAX*704
    DIM T$(G)
    LET MAX=MAX-1
    LET X=0
    LET SEITE=X
    LET Y=X
    LET POS=1
124 LET GETZEICHEN=10
    LET [illegible]=10
    LET SEITENANDRUCK=50
    LET CURSOR=25
    LET ZEILENLOESCH=36
    LET HANDLING=4[illegible]
    LET INSERT=30
    LET ZEICHENVER=15
126 GO TO 100
```

Wichtigste Aufgabe von INITIALISIEREN ist Dimensionierung des Datenarrays T$. 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 entsprechnd 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 Varialen 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, w a s geschieht und nicht sogleich das w i e. 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 Tatstatur lesen, den Code auf STOP abprüfen, - sonst nichts.

```
 5 REM TEXTVER
 6 GO TO 120
 8
 9 REM [illegible]
10 PAUSE 0
   LET Z=CODE INKEY$
   IF Z<>226 THEN RETURN
11 STOP
```

In der Regel wird das Zeichen auf dem Screen dargestellt werden, sofern es druckbar ist, also als Textdatum in T$ 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 [redacted]
13 OVER 0
   PRINT AT Y,X;T$(POS);
   RETURN
```

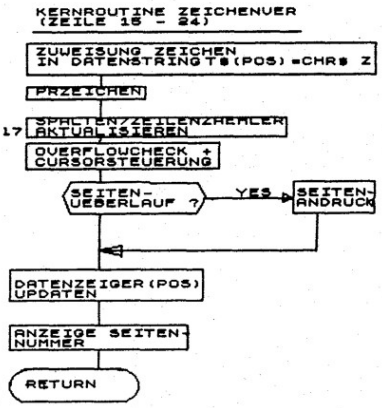
Ebenso simpel ist der Andruck einer Seite aus dem Textstring T$ heraus. Das Teilstück wird einfach aus der laufenden SEITE ermittelt; als Hilfsgröße dient Variable S.

```
48
49 REM [redacted]
50 CLS
   LET S=SEITE*704
   PRINT T$(S+1 TO S+704);
   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 T$
3. Fortzählen 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 / Spaltensynchronisation; hier werden die Cursor-Steuerzeichen verarbeitet, sodaß der Teil Ansprungpunkt 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 / Spalet auf 0,0 gesetzt und die folgende Seite angedruckt. Hilfsvariable S in Zusammhang mit der trickreichen SINCLAIR-Logik macht es möglich.

Abschließend muß noch der Datenzeiger POS, welcher den Bezug zum Datenarray T$ fürs Printen 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 [redacted]
15 LET T$(POS)=CHR$ Z
   GO SUB PRZEICHEN
16 LET X=X+1
   GO TO 18
17 LET X=X+(Z=9)-(Z=8)
   LET Y=Y+(Z=10)-(Z=11)
18 LET Y=Y+(X>31)-(X<0)
   IF X<0 THEN LET X=31
19 IF X>31 THEN LET X=0
20 LET S=(Y>21)-(Y<0)
   LET SEITE=SEITE+S
   LET Y=Y+(S=0)*21*(S<0)
   IF SEITE<0 THEN LET SEITE=0
21 IF SEITE>MAX THEN LET SEITE
   =MAX
22 IF S THEN GO SUB SEITENANDR
   UCK
23 LET POS=SEITE*704+X+1+Y*32
   IF POS>0 THEN LET POS=0
24 PRINT #1; INVERSE 1;AT 0,11
   ;"SEITE ";SEITE+1;
   RETURN
```

Mit der eleganten Verwendung boolscher Operatoren gelingt das Kunststück , ohne viele IF's die gesamte Überlaufsicherung 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"
                erbigt X, wenn Y "false" (bestätigt X)

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

Boolsche 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 [redacted]
26 GO SUB PRÜFZEICHEN
   IF Z=12 THEN LET T$=T$( TO
   POS-1)+T$(POS+1 TO G)
     LET Z=0
     GO SUB SEITENANDRUCK
27 IF Z=13 THEN LET X=32
     GO TO 16
28 GO SUB 17
   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 T$ "implantiert" werden ? Und wie kann man den User optisch darauf aufmerksam machen, daß er sich im "Einfüge-Modus" befindet ?

Würde man Einfügungen Zeichen für Zeichen - wie bei DELETE - verarbeiten, käme das Ganze recht langsam. Also legen wir einen Zwischenstring 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 sei ENTER; wenn nichts eingegeben wurde, soll eine Leerzeile "insertet" werden. Ansonsten können wir für Fragen des Überlaufs Modul ZEICHENVER anspringen.

```
29 REM [redacted]
30 LET A=POS
   LET X$=""
   OVER 0
   GO SUB 16
31 GO SUB GETZEICHEN
32 IF Z<>13 THEN LET X$=X$+(CH
   R$ Z AND Z>31)
     PRINT INVERSE 1;AT Y,X;X$
     GO TO 31
33 IF X$="" THEN LET X$="
   "
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$ ins File eingefügt, INVERSE wieder zurück gesetzt und abschließend ein Seitenandruck gebracht (s. Hauptsteuerung).

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

```
35 REM [redacted]
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 [redacted]
40 PRINT #1;AT 0,0; BRIGHT 1;"
   HANDLING";
   PAUSE 0
   LET A$=INKEY$
   IF A$<>"K" THEN GO SUB 130
   RETURN
41 LET A=POS
   LET X$=""
   PRINT #1; BRIGHT 1;AT 0,19;
   "KOPIE=>ENDE";
   REM [redacted]
42 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 Zielposion. "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 anzuwählen und gleichermaßen mit ENTER zu quittieren. Die Prozedur endet mit Anlage der Textkopie in T$ (aus Zwischstrng X$) und Seitenandruck. - 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 8) und "right" (Code 9) abgefragt werden, um einen Seitenwechsel zu provozieren. Mittels vertrauter Cursortasten kann beliebig im File geblättert werden. Alle anderen Tasten bewirken RETURN und damit - allgemein bei HANDLING - die Menuanzeige.

```
130 REM [illegible]
132 IF A$=CHR$ 8 OR A$=CHR$ 9 T
HEN LET Y=22-(23 AND (A$=CH
R$ 8))
    GO SUB 20
    PRINT #1; BRIGHT 1;AT 0,19
    ;"BLAETTERN\\\/";
    PAUSE 0
    LET A$=INKEY$
    GO TO 132
```

Die Formatierung von Texten

Zwei Lösungen zur Auswahl

Formatierung heißt Textausgaben in eine lesegerechte Form zu bringen. Meist ist darin die Übergabe von Steuercodes für Normalpapierdrucker 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 ausgefranstes 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 X$;

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

```
FLATTERSATZ  Textstring T$
             Druckbreite Z
             Zwischenstring X$
             Zählvariablen A, E, N

LPRINT :     Textstring T$ ( 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 galt es, das Problem "word wrapping" (= Worttrennung) mittels eines Zwischenstrings in den Griff zu bekommen. Schwierigkeiten gibt es bei erstem Füllen von X$ und beim Ausgeben des Textrestes am Ende. "Vor-" und "Nachlauf" sind extra zu programmieren, um diese Besonderheiten abzudecken.

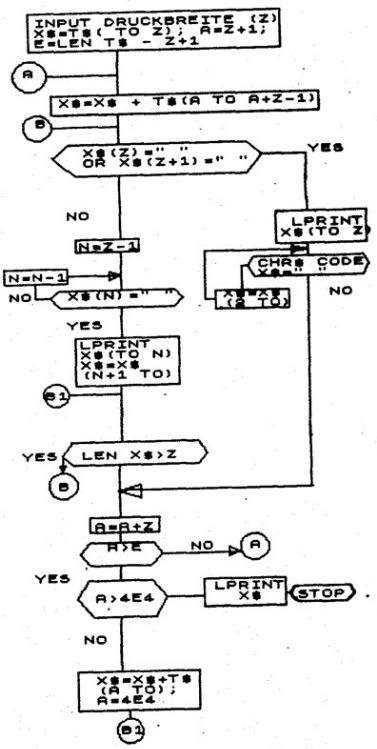
Im Hauptteil des Moduls läuft die Worttrennung am Zeilende nach folgendem Muster ab :

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 Ausbahme; 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 eleminieren, um ein linksbündiges Bild zu gewährleisten.

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

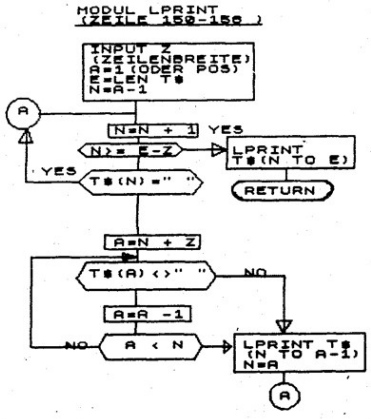
Das Ganze sieht programmiert so aus; weshalb der anderen Lösung (LPRINT) der Vorzug gegeben wurde, werden wir als nächstes zu diskutieren haben.

```
150 REM FLATTERSATZ
160
162 INPUT "DRUCKBREITE ?";Z
    IF NOT Z THEN GO TO 162
163 LET X$=T$( TO Z)
    LET A=Z+1
    LET E=LEN T$-Z+1
164
165 LET X$=X$+T$(A TO A+Z-1)
166
167 IF X$(Z)=" " OR X$(Z+1)=" "
    THEN GO TO 177
168 LET N=Z-1
169 IF X$(N)<>" " THEN LET N=N-
    1
    GO TO 169
170 LPRINT X$( TO N)
    LET X$=X$(N+1 TO )
171 IF LEN X$>Z THEN GO TO 167
172 LET A=A+Z
    IF A<=E THEN GO TO 165
173 IF A>4E4 THEN GO TO 176
174
175 LET X$=X$+T$(A TO )
    LET A=4E4
    GO TO 171
176 LPRINT X$
    RETURN
177 LPRINT X$( TO Z)
    LET X$=X$(Z+1 TO )
178 IF CHR$ CODE X$=" " THEN LE
    T X$=X$(2 TO )
    GO TO 176
179 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 (sie ist in der folgenden Logik der Übersichtlichkeit halber aufgelöst worden). Wenngleich jedes Zeichen von T$ 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 T$ 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 Worttail (d.h. keine Spaces mehr) "getroffen" ist. Die dauernde Initialisierung von FOR-NEXT bedeutet für den SINCAIR zwar eine Menge Schufterei, 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..



Die programtechnische Lösung ist einfach "herunterprogrammiert". Um das Ganze optimal ins Übrige Programm einzubetten, wurde noch ein wenig "Optik" miterledigt. 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 T$) gesetzt, kann jedoch auch eine zweite Position im Text selber erhalten; das war der Grund für die Zwischemspeicherung. Dem SINCLAIR-Programmierer stehen hier weitere Möglichkeiten der Gestaltung offen.. Doch nun das Listing selber.

```
150 REM FORMATIERTER [illegible]
151 PRINT #1; BRIGHT 1;AT 0,19;
    "LPRINT";
152 INPUT "DRUCKBREITE ?";Z
    IF NOT Z THEN GO TO 152
153 INPUT "1=ab POSITION,2=GESA
    MTFILE ";A
    IF A<1 OR A>2 THEN GO TO 15
    3
154 LET A=POS OR A=2
    LET E=G
    LET N=A-1
155 LET N=N+1
    IF N>=(E-Z) THEN LPRINT T$(
    N TO E)
    RETURN
156 IF T$(N)=" " THEN GO TO 155
157 FOR A=N+Z TO N STEP -1
    IF T$(A) <>" " THEN NEXT A
158 LPRINT T$(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 Dokumentaion ganz nützlich.

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

FORMATIERTER AUSDRUCK, erzeugt von Modul "LPRINT"

MEINE DAMEN UND HERREN,
ES IST MIR EINE BESONDERE EHRE, DIE HONORATIOREN 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 STOSS 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	
	INKEY$-Zeichenzuweisung (Z) und Stop-Abfrage	Z
12 - 13	PRZEICHEN	
	Andruck des Datenstrings (T$) mit laufender Position auf dem Screen	T$(POS) Y,X(=Zeile/Sp.)
14 - 24	ZEICHENVER	
	Zuweisung Zeichen in Datenstring + Andruck	(PRZEICHEN)
	Aktualisieren Zeilen/Spaltenwert gem. Cursor; Überlaufcheck	T$(POS) Y,X
	Fortschreiben Seiten-Nr. u. Datenpointer (POS)	SEITE,POS
	Andruck Seite bzw. Seiten-Nr.	(SEITENANDRUCK)
25 - 28	CURSOR	
	Andruck akt. Zeichen	(PRZEICHEN)
	Zeichendelete u. Zeilenvorschub	(SEITENANDRUCK)
	OVER-Print d. Cursorzeichens	(ZEICHENVER) Z,X,Y T$(POS)
29 - 34	INSERT	
	Zwischenspeichern akt. Posit.(in A)	(GETZEICHEN) A,POS
	Endeabfrage, sonst Zeichenspeichern in String X$	Z, X$
	Andruck (invers) d. eigegeb. Textes ggf. Leerzeilen-INSERT	Y,X
	Einfügen Hilfsstring in Textdaten	T$
35 - 38	ZEILENLOESCH	
	Textverkürzung ab akt. Position Positionscheck	T$,POS,X
39 - 45	HANDLING	
	Anzeige und Abfrage User-Option	(GETZEICHEN + CURSOR)
	"KOPIE"-Routine: - Marker Textende (Anfang=POS) setzen	A$,X$
	- Zwischenspeichern Text	A,POS
	- Zielpos. setzten	T$
	- Text kopieren + Seitenandruck	(SEITENANDRUCK)
49 - 50	SEITENANDRUCK	
	Löschen Screen, Errechnen Start/Ende mittels d. akt. SEITE	S,SEITE
	Andruck Daten auf ScreenSeite	
100 - 118	HAUPT-MENU <s. eigene Logik>	
	Anzeige d. Optionen	Z
	Andruck akt. Seite	(GETZEICHEN,
	Einlesen Zeichen + Steuern Verarbeitung	SEITENANDRUCK, INSERT,ZEILEN-LOESCH,ZEICHEN
	Cursor-Abfrage	VER,CURSOR)
119 - 126	INITIALISIEREN	
	Abfrage Seiten + Errechnen Dimensionierung Textarray (T$)	MAX,G
	Erstbesetzung Print- + Posit.variablen	Y,X,SEITE,POS
	Aufbau des Sprungzieltableaus	
129 - 132	BLAETTERN	
	Cursorcheck + Seite auf Überlauf setzen	(ZEICHENVER)
	Ansteuern Seitenandruck(in ZEICHENVER)	A$,Y

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

Variablenliste

Strings
- T$ Textdatenarray
- A$ Handling-Optionen u. Abfrage b. BLAETTERN
- X$ Zwischenstring f. KOPIE u. weitere Intext-Funktionen
- I$ Ladename b. LOAD/SAVE

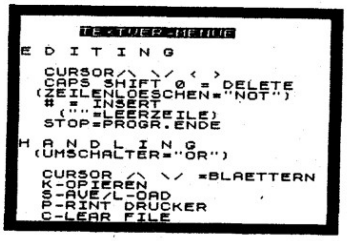
Konstanten
- GETZEICHEN=10
- PRZEICHEN=13
- SEITENANDRUCK=50
- CURSOR=25
- ZEILENLOESCH=36
- HANDLING=40
- INSERT=30
- ZEICHENVER=15

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 T$)
- MAX max. reservierte Seitenzahl
- G 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 Texterray 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 Kommandaufruf ein. Ende ist die mit ENTER quittierte Textstelle inclusive. 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 beachte man, daß Leerzeilen bei LPRINT nicht gedruckt werden.
> Blättern wird initialisiert mit < > Cursor,danach erst geht das eigentliche Paging gemäß Pfeilrichtung vonstatten.
> Da mit INSERT und DELETE das Textfile neu ausgerichtet wird, ist es zweckmäßig, die optische Feinarbeit (Absätze, 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.



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 Elements 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 Datenoragnisation. 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 Plazierung von Daten auf dem Bildschirm
Records sind logische Einheiten ("Sätze") aus einem geordneten Datenbestand ("File"), die dort mehrfach abgelegt sind

Daten können in verschiednen Typen auftreten. In BASIC gibt es davon nur zwei - Foating-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 Verbunddtyp '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 Gedanken zum künstlichen Verbundttyp wäre - man transformiert Zahlen in die Stringform (was mit STR$ ja geht). Dann sind Zahlen und Texte gemixt, 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)
GEHALT (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 D$ (RECORDs,3,20)). Nun taucht ein Problem auf : Die Verbindung von Satzelement (fortlaufend) zu Datenarray (nicht fortlaufend) ist schwierig herzustellen. Bei einem angenommenen aktuellen Record (z.B. R=7) läuft ein Andruck wie folgt:

lfd. Nr. Item(=Satz-element)		Datenelement mit gültigem Subskript
(1)	PRINT "NAME ";	D$ (R,1)
(2)	PRINT "ANSCHRIFT ";	D$ (R,2)
(3)	PRINT "ALTER ";	D (R,1)
(4)	PRINT "GEHALT ";	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 Zurodnung zu den Datarrays, würde P(4) = 1 bedeuten : Item Nr 4 ist in Datenelemt 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 = D$(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 sonstigem Rankenwerk - über eine 'Leiste' erreicht werden. Ein Zahlenarray, der Bezeichner-, Zeilen- und Spaltennummer enthält, ist naheliegend. Es gibt aber einen ökonomischeren Weg - über einen String (R$ wie Reportstring). Da Positionen nur die Werte 0 - 22 (Zeile) bzw. 0 - 31 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 Reportinfomationen auf. Wie ist das am besten zu organisieren ? Da die zur Erzeugung einer Darstellung notwenigen Angaben (in R$) hintereinander liegen, wäre es zu umständlich, bei Generierung den String voll "durchzunudeln". Die Angaben sind variabel, da ein Report einen lang ein anderer einen kurzen Titel hat, der eine Report mit 3 Bezeichnern arbeitet, während der andere z.B. Informationen von 7 Bezeichnern braucht usf. - Die beste (nicht unbedingt übersichtlichste) Lösung wird einen Datenzeiger bemühen. Er zeigt bei vorgegebener Reportnummer immer auf den Anfang der Informationen (in R$). 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

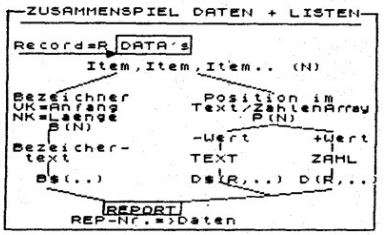
Ausgangsfrage ist - wie verknüpft man aktuelle Satzfelder mit der Plazierung auf einer Bildschirm-Liste ?

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 Plazierung von "NAME,ANSCHRIFT,GEHALT.." 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 plazieren

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

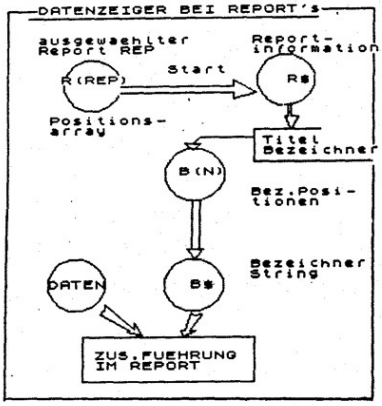
Ziel der vorgestellten Programmlösung ist es, eine (einfache)

Datenverwaltung mit vielfältiger Datendarstellung inReports zu verbinden. realisieren. Den simplen Grundfunktionen im Datenbereich (sgn. 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:

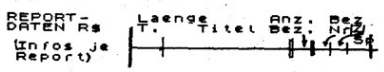
- freie Dimensionierung der Daten und Mischung von Zahlen- und Textfeldern (als "Record")
- unbeschränkte Vorgabe von Bezeichnern durch den Benutzer
- Entwurf beliebig vieler, verschiedener Reports
- freie Plazierung der Felder und Gestaltung der Reports
- Aufruf unterschiedlicher Reports beim Satzlesen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

Programmlösung

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 verwaltet 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 R$ liegen. Danach holt er sich seine Bezeichner, generiert das Layout und plaziert die Daten auf den Screen. Der konkrete Zusammenhang wird durch das folgende Schaubild deutlicher.

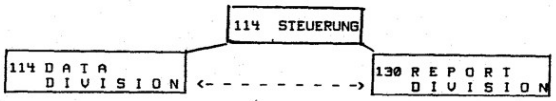


Interessant ist, welche Informationen genau zur Erzeugung eines Reports aus R$ kommen. Dazu die Aufschlüsselung der Eintragungen im Reportstring, wobei Zahlenangabe dort natürlich als Codes stehen.



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-Dokumenation).

Funktionalzusammenhang Module



118 INITIALISIEREN

10 BEZEICHNER
Eingabe
Sortierung Bez
Ausgabe nach Sort
Dimensionieren

140REPORTDEFINITION

50 NEUEINGABE RECORD

71 Anlisten Satz
+ Dateneingabe

86 UPD ATE

71 Anlisten Satz
+ Dateneingabe

46 LOESCHEN

90 STOEBERN (ohne Report)

120-124 LOAD/SAVE/ENDE

140 REPORT DEFINITION

160 BEZEICHNERLISTE

170 NEUEN REPORT ANFUEGEN

Redimensionieren Reportarray (-> REPORT LOESCHEN)
Report neu definieren(->REPORT DEFINITION)

180 REPORT LOESCHEN

172 Report-Select
+ Redimensionieren Reportarray

190 REPORTANDRUCK
(ohne Daten)
172 Report-Select
+ Print

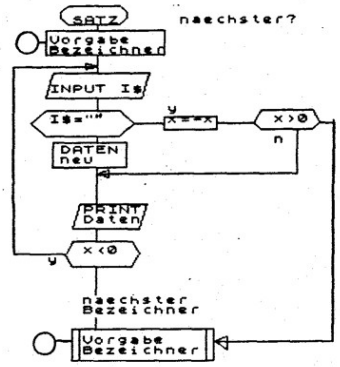
200 LESEN MIT REPORT

190 Reportandruck

+ 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 Progamm 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 RÜcksprungoption
Festlegen der Bezeichner (anfangs) + Liste
Update mit Input-Korrektur (Bezeichner invers)
Stöbern im File unter Cursor-Control ohne Report
..oder mittels definierter Reports

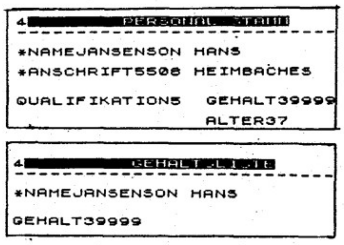

```
>>> H A U P T M E N U <<<
 DATA DIVISION
1-INITIALISIEREN
2-NEUEINGABE
3-UPDATE
4-LOESCHEN
5-STOEBERN
6-LOAD
7-SAVE
8-ENDE
9-=>REPORT DIVISION
```

```
>>>  M E N U  <<<
 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    GEHALT
5    QUALIFIKATION
```

```
    S A T Z 2
 ---------------------
 *ANSCHRIFT4405 ANGELROGGE
 *NAMESCHULZE, GERD
 ALTER36
34
 GEHALT38945
 QUALIFIKATION3
4
5
6
```

```
 A T Z 1
5000 KOELN 90
MAIER, HANS
35
48927
3
 A T Z 2
4405 ANGELROGGE,
SCHULZE, GERD
34
38945
5
 A T Z 3
4400 TELGDE
BAUMANN FRANZ
34
36709
9
```



Programmdokumentation

Programmaufbau

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

	-Löschen Zahl/Textfelder	D(R,N) D$(R,N) ZAHL,TEXT
NEUEINGABE RECORD (50-68)	-Aufsuchen nächster leerer Satz	N,Y,R,REC
	-ANLISTEN SATZ und Dateneingabe	ZAHL,TEXT D$(.),D(.)
ANLISTEN SATZ (71-84)	(ohne Report) Satzverarbeitung für Lesen,Update + Neueingabe	B$,I$
	-Vorgabe Bezeichner	D$(.)
	-Andruck vorh. Daten mit Typenselektion	D(.)
	-fakultative Dateneingabe	P(N),X,N
UPDATE (85-88)	-Recordanwahl	I$
	-Ändern der Daten im Untermodul ANLISTEN SATZ	R,REC
BROWSE (SIOEBERN ohne Report) (90-100)	-Festlegen Einsprungstelle im File	I$
	-Check auf "leer" (kein Andruck)	R,REC
	-Datenprint gem Item-Liste P(..)	ZAHL,TEXT
	-Cursorsteuerung fürs Blättern	N,BEND P(N) D(.),D$(.)
S T E U E R U N G Hauptmenu <DATA DIVISION> (110-129)	-Display Menu	
	-Abfrage Useroption und Unterprogramm-aufrufe	I$
	-Direktbearbeitung von LOAD/SAVE/Sprung in die REPORT DIVISION	
<REPORT DIVISION> MENUE (130-139)	-Display Report-Menu	
	-Abfrage Useroption + Unterprogramm-aufrufe	I$
	-ggf. Sprung zur DATA DIVISION	
REPORTDEFINITION (140-153)	-Dimensionieren Positionsarray R(..) d.h. Anzahl Reports gemäß User	REND R(REND)
	-Definieren Report mit Titel,Bezeichnern	N,M B(..)
	-Plazieren Bezeichner auf dem 'Layout' unter Cursorsteuerung	Y0,Y1 B$
	-Sichern Reportinfos in R$ incl. Positionierungen (Zeile,Spalte)	X,R$,Z$
	-Setzen Zeiger in R(..) auf Reportanfang (R$)	

```
BEZEICHNER     aktuelle Liste für Reportdefinition
    (160-164)  -Andruck                                  N,BEND
               -Copy-Anfrage                             BS(,)
                                                         B(N)

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

REPORT-SELECT
    (173-177)  Auswahl eines Reports auf der Input-      RS,R(..)
               leiste (Titelanzeige mit Cursorcontrol)   IS,REP
                                                         REND,X0,X1
REPORT LOESCHEN
    (180-183)  -Unteraufruf von REPORT-SELECT            R(REP)
               -Auswahl der betreffenden Reportinfos     REND
                (in RS)                                  RS,X0,X1
               -Stringverkürzung (von RS) und Redimen-   N,M,X
                sionieren Positionsarray R(N)

REPORTANDRUCK
    (190-199)  -Unteraufruf REPORT-SELECT                R,RS
               -Auslesen Item-Infos (aus RS)             P(..)
               -Generieren Report als Leerandruck        ZAHL,TEXT
                bzw. bei vorhandenen Daten Selektion     D(.),DS(.)
                von Zahlen- oder Textdaten               N,Y,X1

REPORTLESEN (analog zu SIOEBERN in der 'DATA DIVISION')
    (202-212)
               -Einsprungstelle im File festlegen        IS
               -Unteraufruf  REPORT-SELECT und           R,REC
                 REPORTANDRUCK (mit Daten)
               -Cursorsteuerung für satzweises Lesen
                bzw. Anwahl eines anderen Reports
                                 (Usroption '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 Bezeich-
                    ner,BezeichnerNr,Zeilen/Spaltenposition)

3.Datenzeiger
          P(BEND)   Verknüpfung BezeicherNr 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 Datenbe-
                     reich)
5.aktuelle Daten
          IS        Userinput
          R         bearbeiteter Record
          REP       gewählter Report

6.sonstige Variablen
          ZS        Löschstring bei REPORTDEFINITION
          X0,X1     Zwischengespeicherte Positionie-
          Y0,Y1      rungen
          N,M       Laufvariablen
          X,Y       Zahl- und Abfragevariablen
7.Funktionen
          FN B(X)   ..liefert den Vorkommawert im Posi-
                    tionierungsarray '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). Wen die alphabetische Folge stört (z.B. beim Update), kann sie durch Bezeichnerinput mit führendem Space beeinflussen. Dann erscheint fortan "_"Verfasser 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 "Leeranzeige";danach Korrekturen machbar
  bei Update - vorhandene Daten können mit Leer-

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

+ Das STOEBERN im File mit oder ohne Report erfolgt durch Cursorbewegung (Pfeiltasten ohne Shift); es gibt für den User 2 Arten der CursorControl - satzweises Blättern (Record vor = "8", zurück = "5") und Wahl des Reports mit den Up- und Down-Tasten ("7" bzw. "6"). 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"..

```
  2 REM REPORTE
4&2
  5 GO TO 110
    DEF FN B(X)=INT B(X)
    DEF FN G(X)=100*(B(X)-FN B(
    X))+FN B(X)
  6
  7 REM INITIALISIEREN
  8
  9 REM BEZEICHNER
 10 INPUT "WIEVIEL DATAITEMS=BE
    ZEICHNER";BEND
    IF NOT BEND THEN GO TO 10
 12 LET B$=" "
    DIM P(BEND)
    LET ZAHL=0
    LET TEXT=0
    DIM B(BEND)
    LET Y=0
    CLS
    PRINT "NR BEZEICHNUNG (*=TE
    XTDATEN)","----------------
    ----------"
 14 FOR N=1 TO BEND
     INPUT "BEZ.?";I$;" TEXTDA
     TEN ? Y-ES ";X
 16  LET I$=("*" AND X=Y)+I$
     PRINT N;TAB 3;I$
 20
 22  LET B(N)=LEN B$+.01*(LEN I
     $-1)
     LET B$=B$+I$
    NEXT N
24  LET B$=B$(2 TO )
26  REM SORTIERUNG BEZ.LISTE
28  FOR N=1 TO BEND-1
     LET M=N
     FOR X=N+1 TO BEND
      IF B$(FN B(M) TO FN G(M))
      >B$(FN B(X) TO FN G(X)) T
      HEN LET M=X
30   NEXT X
     LET X0=B(N)
     LET B(N)=B(M)
     LET B(M)=X0
    NEXT N
```



```
 34 REM AUSGABE SORT .BEZEICHNE
    R
 36 CLS
    PRINT "SORTIERTE BEZEICHNER
    ","----------------------"
    FOR N=1 TO BEND
     PRINT N;TAB 3;B$(FN B(N) T
     O FN G(N))
     IF B$(FN B(N))="*" THEN LE
     T TEXT=TEXT+1
      LET P(N)=-1*TEXT
 37  IF B$(FN B(N))<>"*" THEN L
     ET ZAHL=ZAHL+1
      LET P(N)=ZAHL
 38 NEXT N
 39 INPUT "K-OPIE ";I$
    IF I$="K" THEN COPY
 40 REM DIMENSIONIERUNG
 42 INPUT "WIEVIEL RECORDS ?";R
    EC

    IF NOT REC THEN GO TO 42
 44 DIM D(REC,ZAHL)
    DIM D$(REC,TEXT,20)
    RETURN
 45 REM [illegible]
 46 INPUT "REC-NR ?(ENTER=AKT/0
    =MENU)";I$
    IF I$<>"" THEN LET R=VAL I$
 47 IF NOT R OR R>REC THEN RETU
    RN
 48 FOR N=1 TO ZAHL
     LET D(R,N)=0
    NEXT N
    FOR N=1 TO TEXT
     LET D$(R,N)=" "
    NEXT N
    GO TO 46
 49 REM [illegible]
 50 LET Y=0
 51 FOR R=1 TO REC
     IF ZAHL THEN IF NOT D(R,1)
     THEN GO TO 54
 52  IF TEXT THEN IF D$(R,1,1)=
     " " THEN GO TO 54
 53 NEXT R
    PRINT FLASH 1;"DATEI VOLL"
    PAUSE 100
    RETURN
 54 GO SUB 71
 56 INPUT "NAECHSTER SATZ ?Y-ES
    ";A$
    IF A$=Y THEN GO TO 51
 58 RETURN
 70 REM ANLISTEN SATZ (OHNE REP
    )
 71 CLS
    PRINT "     S A T Z "; BRIGHT
     1;R; BRIGHT 0;"-------------
    ----------";
 72 FOR N=1 TO BEND
     PRINT INVERSE 1;B$(FN B(N)
     TO FN G(N));
 74 INPUT ("TEXT" AND (B$(N)="*
    ")) I$
    IF I$="" THEN LET X=-X
    GO TO 80+4*(X>0)
 76 IF P(N)<0 THEN LET D$(R,ABS
    P(N))=I$
 78 IF P(N)>0 THEN LET D(R,P(N)
    )=VAL I$
 80 IF P(N)<0 THEN PRINT D$(R,A
    BS P(N))
 81 IF P(N)>0 THEN PRINT D(R,P(
    N))
 82 IF X<0 THEN GO TO 74
 84 NEXT N
    RETURN
```

Sortierverfahren

Aus der Fülle der Möglichkeiten

..fällt es schwer,etwas Geeignetes auszuwählen. Da ist die Rede von Sortierung durch Zerlegen, Mischen , da wird von einzusetzenden Bäumen,Listen, Haufen (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 HC-Magazine veröffenlichen Beiträge zum Thema "Sortieren" (Einen recht guten Überblick bringt etwa das "P.M. Computerheft" 8/85 - Seite 88).

Sortierungen sind unter zwei Vorraussetzungen erfoderlich
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 Neueingabe/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 Meßgrößen 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 brauchenn meist Zusatzfelder, viele Variablen und beträchtlichen Programmaufwand. Da sich "interpretiertes" BASIC dazu schlecht eignet, stellen viele trickreiche Verfahren sich selber ein Bein. Für BASIC beherzige man die Deviss "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 verbreitesten. Die Routine dazu liegt in einem Testrahmen, um Verbesserungen aufzuzeigen. Sie basiert auf folgenden Variablen:

T$ = ursprügliche 36 Testzeichen	E = Anzahl zu sortierender Elemente (36)
A$ = nachher sortierter String	N = äußere Schleifenvariable
C$ = 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 T$="0S1I6M4EN3AOFZUCELY
 2T5P8H7BX9VDURKJ"
14 LET E=LEN A$
16
17 REM BUBBLE EINFACH
18
19 LET A$=T$
20 FOR N=1 TO E-1
22  FOR M=1 TO E-N
24    IF A$(M)>A$(M+1) THEN LET
       C$=A$(M)
       LET A$(M)=A$(M+1)
       LET A$(M+1)=C$
26  NEXT M
28 NEXT N
30 PRINT A$
   REM 16 SEC
32 REM BUBBLE SCHNELLER
34
35 LET A$=T$
36 FOR N=1 TO E
38  FOR M=E TO N+1 STEP -1
40    IF A$(M)<A$(N) THEN LET C
     $=A$(M)
       LET A$(M)=A$(N)
       LET A$(N)=C$
42  NEXT M
44 NEXT N
45 PRINT A$
   REM 13 SEC
46
48 REM BUBBLE OPTIMIERT
49
50 LET A$=T$
52 FOR N=1 TO E-1
     FOR M=E TO N+1 STEP -1
       IF A$(M)<A$(N) THEN LET C
      $=A$(M)
       LET A$(M)=A$(N)
       LET A$(N)=C$
54   NEXT M
   NEXT N
56 PRINT A$
   REM 12 SEC
```

Wir machen nun die erste Verbesserung, die meßbare Zeitersparnis bring. Einmal drehen wir die Richtung im Innern (mit negativem Step) von "oben nach unten" um. Zum andere 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 "Halbierungdistanz" 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 ereicht, 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
M = Halbierungsvariable
N = Begrenzer innere Schleife
A = lfd. Element innere Schleife (Subskript)
J = lfd. Vergleichsargument (Subskript)
K = lfd. Anfangergument für Vergleich (Subskript)

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 HC's sind so gutmütig wie SINCLAIR..). Und noch eine Schellermach-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$="INQYTFVC7DER2B5633X
   ZA0W9Q1OP8LM4KJHU"
16 LET A$=T$
18 LET E=LEN A$
19
20 LET M=E
22 LET M=INT (M/2)
   IF NOT M THEN PRINT A$
    STOP
24 LET N=E-M
   FOR A=1 TO N
    LET J=A
26 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 A
   GO TO 22
   REM 7,5 SEC
30 REM ...VERBESSERT
31
32 LET A$=T$
   LET E=LEN A$
   LET M=E
34 LET M=INT (M*.5)
   IF NOT M THEN PRINT A$
    STOP
36 LET N=E-M
   FOR A=1 TO N
    FOR J=A TO 1 STEP -M
     LET K=M+J
     IF A$(K)<A$(J) THEN LET C
     $=A$(J)
      LET A$(J)=A$(K)
      LET A$(K)=C$
     NEXT J
38 NEXT A
   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 A$ ist auf 30 Zahlzeichen 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

Angaben , um mit QS arbeiten können. kann. Folgende Dinge sind anzugleichen, sofern man nicht bei den benutzten Variablen bleibt:

- Elementzahl (E) mit zu sortierender Anzahl besetzen
- Dimensionieren Zwischenspeicher mit Elementzahl (B,E,2)
- Bereitstellen a) Zählvariablen (L,R,I,J)
  b) Austausch- und Zwischenfelder (bei Strings analog T$ + B$)
- Initialisieren Subskripte 'Sortanfang' + 'Sortende' (analog Zeile 18)

Verwendete Variablen und ihre Bedeutung:

A$ = zu sortierender String	E = Anzahl Elemente
T$ = Austauschfeld	B (E,2) = Zwischenspeicher 'Sortplätze'
B$ = zwischengespeichertes Vergleichsargument	P = Positionszähler
	L,R = Begrenzer 'links' bzw. 'rechts
	I,J = Zwischenzähler

```
 10 REM QUICKSORT-DEMO
 12
 14 REM TESTSTRING
 15 LET A$=""
    LET E=30
 16 FOR N=1 TO E
      LET A$=A$+CHR$ (48+INT (RN
      D*10))
    NEXT N
 17 PRINT A$''
 18 LET P=1
    DIM B(E,2)
    LET B(1,1)=1
    LET B(1,2)=E
 20
 80 REM SORTIERUNG
 90
152 LET L=B(P,1)
    LET R=B(P,2)
    LET P=P-1
154 LET I=L
    LET J=R
    LET B$=A$(INT ((L+R)*.5))
156 IF A$(I)<B$ THEN LET I=I+1
      GO TO 156
158 IF B$<A$(J) THEN LET J=J-1
      GO TO 158
160 IF I<=J THEN LET T$=A$(I)
      LET A$(I)=A$(J)
      LET A$(J)=T$
      LET J=J-1
      LET I=I+1
162 IF I<=J THEN GO TO 156

164 IF I<R THEN LET P=P+1
      LET B(P,1)=I
      LET B(P,2)=R
      PRINT A$
166 LET R=J
    IF L<R THEN GO TO 154
168 IF P THEN GO TO 152
170 PRINT 'A$
```



```
387963541209909886370263313197

387763541209103688637026331999
100101543269773886378263339999
000111543269773888637826333999
000111543263773336376288899999
000111543263263336377778889999
000111333232366346577778889999
000111232333366346577778889999
000111232333366346577778889999
000111223333356346677778889999
000111223333343656667778889999
000111223333343656667778889999
000111223333333456667778889999
000111223333333456667778889999

000111223333333456667778889999
```

Memo:
Bei Zahlensortierungen sind selbstverständlich die Stringfelder (A$, B$, T$) durch numerische Variablen zu ersetzen.

Abschließend ein einfache Faustegel für diejenigen, die nicht wissen, mit "was" sie nun sortieren sollen:

+ Bei weniger als 20 Elementen reicht BUBBLE aus;
+ universell und noch schnell bis etwa 100 Items ist SHELL;
+ bei größeren Mengen - soweit total ungeordnet (Vorcheck !) - ist QS lohnend

Natürlich lassen sich auch Sorts ganz vermeiden - wie ? Entweder durch Einsortieren schon bei Input oder durch Direktzugriff, Stichwort "Hash"-Verfahren für Leute, die mehr wissen wollen (Ein erster Einstieg dazu ist der Beitrag in "C't", Heft 9/85, Seite 74).

Suchen und Wiederfinden

Ideal - der Direktzugriff

Suchen ist ohne Probleme, wenn Suchargument gleich Arraysubskript ist. Dann kann direkt auf Daten zugegriffen werden. Wurde beispielsweise eine Schallplattensammlung von 1 bis N durchnummeriert und jede Einzelinformation (Titel, Komponist, Orchester, Interpret etc.) in gleicher Folge gespeichert, fällt blitzschneller Zugriff nicht schwer. Die 4 Felder seien mit 20 Stellen in einem Stringarray enthalten, der mit DIM S$(N,4,20) angelegt war. Der Zugriff zur Schallplatte X bedarf nur eines einfachen Statements

```
PRINT X;TAB 4; "TITEL : "; S$(X,1);TAB 4;"KOMPONIST :"
;S$(X,2)...
```

Nummerkreise sind aber selten geschlossen, von Textschlüsseln ganz zu schweigen. Wer wird schon für Umsatzzahlen der Jahre 1970 - 1985 fast 2000 Elements 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 Komplexheit nicht weiter eingegeangen 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 usf. 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. Anzeigetexe). 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-	<	>	>	>	>	=	

gleichs Offset (INT-Rundung)	-50	+25	+13	+7	+4	found

2er Potenzen spielen (indrekt) ein 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 Maximalstepzahl. Der Durchschnitt liegt unter der Hälfte davon, falls die Sortmenge nicht gerade gleich der nächsten 2er Potenz ist. Folgende Hilfstabelle gibt darüber Auskunft.

	0	1	2	3	4	5	6	7	8	9
2er Potenz	2 1	2 2	2 4	2 8	2 16	2 32	2 64	2 128	2 256	2 516
Elemente bis..	1	3	7	15	31	63	127	255	515	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 Namenssuche 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 A$, der von den Sorts her bekannt ist. Er muß vorhanden bzw. eingelesen sein. Die übrigen Variablen seien kurz erklärt, ehe das Programmangebot folgt:

Allgemein: A$ (E,S) = sortierter Datenarray
E = Elementzahl (hier 100)
B$ = Suchargument (Sstellig)

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

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

Linearsuche:M = Laufvariable


  10 REM SUCHVERFAHREN
  12
  15
  16 REM LADEN SORTSTRING A$
  17 LOAD *"M";2;"A$STRING" DATA
     A$()
  18 LET E=100
     INPUT "SUCHWORT(5 STELLEN)"
     ;B$
     IF B$="" THEN STOP
  19
  20 REM BINAERSUCHE
  21
  22 LET U=0
     LET O=E+1
     IF B$>A$(E) OR B$<A$(1) THE
     N GO TO 100

  24 LET M=INT ((O+U)*.5+.5)
  26 IF B$=A$(M) THEN GO TO 120
     IF B$<A$(M) THEN LET O=M
     GO TO 30
  28 LET U=M
  30 IF (O-U)<>1 THEN GO TO 24
  33 REM INDEXSEQ-SUCHE
  34
  35 REM EINRICHTEN INDEX
  36 DIM C$(26)
     FOR N=1 TO 26
     LET C$(N)=CHR$ 0
     NEXT N
  37 FOR M=1 TO E-1
  39   IF A$(M,1)<>A$(M+1,1) THEN
       LET C$(CODE A$(M,1)-64)=C
       HR$ M
  42 NEXT M
  43 LET C$(CODE A$(M,1)-64)=CHR
     $ M
  44
  45 REM SUCHE
  46 INPUT "SUCHBEGRIFF(5 ST.)";
     B$
     IF B$="" THEN STOP
  47 LET M=CODE B$(1)-64
     IF NOT CODE C$(M) THEN GO T
     O 100
  49 LET O=CODE C$(M)
     LET U=1
  50 IF M>1 THEN LET U=CODE C$(M
     -1)
       IF NOT U THEN LET M=M-1
       GO TO 50
  51 FOR M=U TO O
       IF A$(M)=B$ THEN GO TO 120
  52 NEXT M
```



```
  55 REM Z.VERGLEICH-LINEARSUCHE
  56 INPUT "SUCHBEGRIFF(5 ST.)";
     B$
     IF B$="" THEN STOP
  57 FOR M=1 TO E
       IF A$(M)=B$ THEN GO TO 120
  58 NEXT M
 100 PRINT B$;" NOT FOUND"
     STOP
 120 PRINT B$;" FOUND AT NO ";M
 122 GO TO 46
```

FORMEN DER SPEICHERUNG

Der universelle Datenzeiger

Speicherungsgrenzen

Gut 40 K RAM stehene 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 "Verpointerung" 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 Zahlenablage aus ? Zahlen sind Speicherfresser. Bei 5 Bytes pro Floating Point (ohne Variablennamen) kommt peu a 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 usf.

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 Ganzzahl, das zweite den 256er Modulus. Programmtechnich 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 benötige Platz belegt wird. Da wird mancher sofort an einen Riesenstring denken - aber Vorsicht ! Wenngleich das komfortable Stringslicing 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 Speicherungsgrenzen 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 Längenfelder 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 raffinierter als die Datenzeiger bislang (wie in HEIMBUDGET und REPDATA).

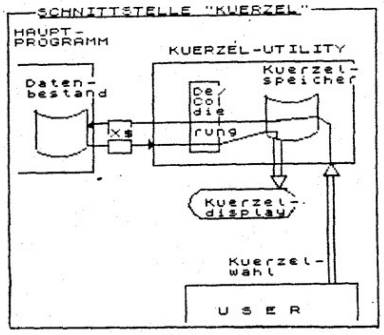
Platz sparen mit Pointern (Programm KUERZEL)

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 de/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 hüpft 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 Menuleiste am Screenfuß, Langtextinput entfällt. Das Programm realisiert daher zwei Forderungen: 1. Kürzelstring statt Bezeichnertabelle, 2. Anwahl 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

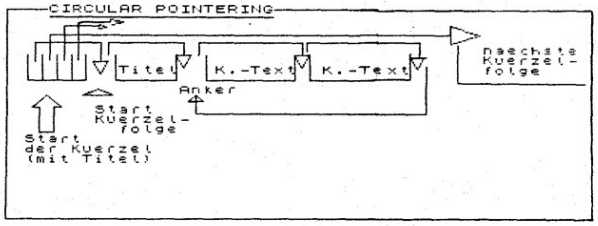
("Blättern"), Space quittiert die Wahl, eine andere Taste bringt die Titelsbene. Die gewählte Kürzelnummer wird in X$ als Code gespeichert und von Unterroutine K-IN dem Aufrufprogramm übergeben. Das wiederum kann via X$ ein Datenitem K-OUT zur Verfügung stellen, wo es decodiert wird. Die Logik zeigt 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 J$, K$ selber umfaßt alle Texte (Titel + Kürzel). Variable TEND erteilt Auskunft, wieviel Titel (=Kürzelgruppen) existieren. - Es gibt im Programm zwei Arten von Pointern :

a) einfache Vorwärtszeiger (J$ => jeweils die Startposition eines 'Titels', gespeichert als Code)
b) Ringpointer bei Kürzeln eines Titels (in K$ 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 X$)
	258-322	K-DEF = Definition der Kürzel (Titel + Texte)
		Unterroutinen
	264-270	Dimensionierung
	274-284	Titeleingabe
	288-296	Kürzeltexte
	298-308	Andruch + Anlage der Längenfelder
	312-318	Speichern Titelfeldlängen
	322	Fehlerausgang ("Überlauf")
	324-328	Belegungsplot
	330-346	Andruck Kürzelliste

Variablen

Strings

X$	Kürzelcode bzw. OUT-Text
K$	Kürzeldaten (Texte + Pointer)
J$	Start Textpositionen in K$
I$	User-Input
C$	Dimensionierungshilfstring

num.Variablen

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

```
328 PLOT 0,158
    DRAU 255,0
    DRAU 0,-3
    DRAU -255,0
    DRAU 0,2
    DRAU LEN,0
    DRAU 0,-1
    DRAU -LEN,0
    RETURN
330
332 REM K-PRINT
        OUT=X$
336 CLS
    PRINT TAB 3;"LISTE K U E R
    Z E L";TAB 3;"-------------
    ------";
340 FOR N=1 TO LEN J$
342  PRINT N;":";
344  LET X$=CHR$ N
     GO SUB 250
346 NEXT N
    RETURN
```

Sonderformen der Speicherung

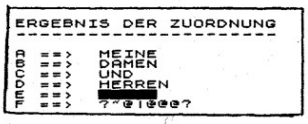
UDG's als Floskelspeicher

Der folgende Abschnitt befasst sich mit ungewöhnlichen Formen der Speicherung. Meist ist nicht bekannt, wo überall in SINCLAIRs Maschine noch etwas gespeichert werden kann. Wie kann man mehr Speicher gewinnen ? Wo gibt es "Nischen", in die man Daten bringen kann ?
Ohne zu tief auf Systemebene herabzusteigen und mit Maschinencodierern wetteifern zu wollen, ein Blick ins Handbuch (Stichwort "The memory", Kapitel 24) zeigt, wo etwas zu holen ist. Nicht belegt sind oft z.B. der Print Buffer,das Displayfile, der freie Raum jenseits vom RAMTOP usf. Zum Screen als Datenspeicher kommen wir noch, fangen wir mit etwas Vertrauterem an - den User defined grafics (UDG). Normalerweise wird man dort, vor NEW geschützt, eigene Grafikzeichen ablegen (neben MC's gewöhnlich). Das wäre doch ein nützlicher Platz, um Daten programmübergreifend festzuhalten. Statt Parameter auszulagern und mit LOAD DATA umständlich aufzurufen, bleiben die UDG-Informationen sicher stehen. Verfolgen wir die Idee weiter; wie bringt man normale Zeichen dorthin ?
Der Bereich nimmt Grafik "binär" definiert auf. Das kann nicht darüber hinwegtäuschen, daß die 8 Bits pro Pixelreihe dort in üblicher Codeform liegen. Die Werte 0-255 sind dort, wie bei jedem normalen, Poke/Peek vorhanden. Man kann andrerseits die UDG's in Zahlenform woanders ablegen (z.B. in einem DATA - Block) und mit READ zur Neubelegung von Grafik einlesen. Da jedes Grafikzeichen als 8*8 Matrix definiert ist und jede Reihe auf 8 Bitkombination (= ein Byte) beruht, fällt die Sache nicht schwer. Jedes UDG ist mit 8 Bytes definiert; legt man hier Kurztexte o.ä. ab, hat man insgesamt 21 mal 8 Zeichen als Speicher. Es handelt sich quasi um einen gesichterten Array, der mit dem Grafikzeichen qualifiziert wird. Insgesamt 168 Bytes lassen keine großen Sprünge zu, erlauben aber viele

Verwendungen. Man kann z.B. wiederkehrende Texte, ja sogar Stringzahlen (bis 8 Stellen - Vorsicht bei STR$ 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 B$, 8 Zeichen lang) wird an die (USR-)Adresse gepoket, u.z. in seiner Codeform. Das wars denn schon..
Das Programm kann gelöscht werden, die maximal 21 Floskeln bleiben erhalten. - Der Aufruf ist Sache einer Programmzeile (s. Zeile 25). Der in I$ eingegebene Buchstabe dient per PEEK USR (und Fortzählung), den Buchstabencode rückzugenerieren. Man soll und braucht wohlgemerkt nicht das UDG (mit "graphics"-Taste) selber aufzurufen; 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's als Text-Floskeln
12
13 DIM B$(8)
14 INPUT "UDG-BUCHST(A-U)";I$;
   " TEXT ";B$
   IF I$="" THEN GO TO 20
16 FOR N=0 TO 7
   POKE USR I$+N,CODE B$(N+1)
   NEXT N
17 GO TO 13
18
20 REM AUFRUF
22
23 PRINT "ERGEBNIS DER ZUORDNU
   NG":"----------------------
   ":
24 INPUT "UDG";I$
   IF I$="" THEN STOP
25 PRINT I$;" ==>  ";
26 FOR N=0 TO 7
   PRINT CHR$ (PEEK (USR I$+N
   ));
   NEXT N
27 PRINT
29 GO TO 24
30 REM WEITERER VORTEIL - INFO
   RMATION BLEIBT UEBERS PROGR
   AMM HINAUS ERHALTEN (NEUGES
   CHUETZT !)
31 PRINT:"ABCDEFGHIJKLMNOPQRST
   U"
```



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 16.384 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 Vorkehrungen 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. Wen 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 vdn 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 STR$ 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 B$, 8 Zeichen lang) wird an die (USR-)Adresse gepoket, u.z. in seiner Codeform. Das wars denn schon..
Das Programm kann gelöscht werden, die maximal 21 Floskeln bleiben erhalten. - Der Aufruf ist Sache einer Programmzeile (s. Zeile 26). Der in I$ eingegebene Buchstabe dient per PEEK USR (und Fortzählung), den Buchstabencode rückzugenerieren. Man soll und braucht wohlgemerkt nicht das UDG (mit "graphics"-Taste) selber aufzurufen; 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 [redacted]
12
13 DIM B$(8)
14 INPUT "UDG-BUCHST(A-U)";I$;
   " TEXT ";B$
   IF I$="" THEN GO TO 20
16 FOR N=0 TO 7
   POKE USR I$+N,CODE B$(N+1)
   NEXT N
17 GO TO 13
18
20 REM AUFRUF
22
23 PRINT "ERGEBNIS DER ZUORDNU
   NG";"----------------------
   ";
24 INPUT "UDG";I$
   IF I$="" THEN STOP
25 PRINT I$;" ==>  ";
26 FOR N=0 TO 7
   PRINT CHR$(PEEK (USR I$+N
   ));
   NEXT N
27 PRINT
28 GO TO 24
29
30 REM WEITERER VORTEIL - INFO
   RMATION BLEIBT UEBERS PROGR
   AMM HINAUS ERHALTEN (NEUGES
   CHUETZT !)
31 PRINT."ABCDEFGHIJKLMNOPQRST
   U"
```

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 16.384 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 Vorkehrungen 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. Wen 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 binäre Suchen von "MAIER" noch mal gezeigt ist, geht es an den Aufbau des Bildschirm - Datenspeichers. Dort sollen nämlich die Zusatzinformationen (etwa über Herrn MAIER) 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 immmer hat. Werden nun die 'Teiler' berücksichtigt, dann machen genau 10 Screen$ das im User-RAM angelegten Maximum von 1920 Sätzen aus. Durch die Integerumrechnung (vgl. Zeile 44) werden jedem Datenrecord genau 32 Stellen Zusatzinfomation zugeordnet. Der INT-Teil einer Satznummer adressiert geschickt den Zugriff per SCREEN$-LOAD: die Zahle N1 wandert 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 A$ zu kümmern; er kann diesen Programmteil "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 saven 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 gepoket.

Nachdem die Zeichen in I$ gesammelt sind, können sie angedruckt 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), ergiben 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 Initialiseren (Testdaten)
- Dimensionieren
- Testdaten belegen
- Längenausrichtung Suchstring (B$)

```
  22-36  Binärsuche
  37-49  Datenspeicherung auf Screen
         - Dimensioniern + Aufnahme Input (I$)
         - Laden Speicheradresse
         - Einpoken Zusatzinfos
  57-62  Retrieval Screen-Infos
         - Laden Screen
         - Herauspeeken Infos + Ablage in I$
         - Andruck

Variablen

A$(1920,20)
         Testdaten
     I$  Userinput/Ausgabe-Infos
     B$  Demo-Suchstring
     S$  Save/Load-Name

U,O,M    Aufteiler f. Binärsuche
         (M auch RecNr)
N1,N2    Adressverschlüssler
     NA  Startadresse Zusatzinfos (entspr. RecNr)
      E  Anzahl Records im Datenfile (Begrenzer)
      N  Laufvariable
```

```
 10 REM [redacted]
 12
 14 REM D E M O
 15
 16 REM TESTARRAY EINRICHTEN
 17 DIM A$(1920,20)
 18 REM BEISPIEL +NR FUER DATEN
+ NAME (=SUCHBEGRIFF)
 19 LET A$(1919)="MAIER"
    LET A$(1920)="ZZZZZZZ"
 20 LET E=1920
    LET B$="MAIER"
 21 IF LEN B$<20 THEN LET B$=B$
+" "( TO 20-LEN
 B$)
 22 REM BINAERSUCHE
 23 LET U=0
    LET O=E+1
    IF B$>A$(E) OR B$<A$(1) THE
N GO TO 33
 24 LET M=INT ((O+U)*.5+.5)
 25 IF B$=A$(M) THEN GO TO 35
 26 IF B$<A$(M) THEN LET O=M
    GO TO 30
 28 LET U=M
 30 IF (O-U)<>1 THEN GO TO 24
 31
 32
 33 PRINT "NOT FOUND"
    STOP
 34
 35 PRINT B$,"FOUND AT ";M
 36
```



```
 37 DIM I$(32)
 38 INPUT "DATA(32 STELLEN)";I$
 40 REM DATEN IN SCREENSPEICHER
BRINGEN
 42
 43 CLS
 44 LET N1=INT (M/192)
    LET N2=M-N1*192
 45 LET S$="3"+STR$ N1
 46 LET NA=16384+(N2-1)*32
 47 FOR N=0 TO 31
    POKE NA+N,CODE I$(N+1)
    NEXT N
 49
 50 REM WEITERE DATEN...
 51
 52 SAVE *"M";2;S$SCREEN$
 54
 55 REM ZURUECKHOLEN(VEREINFACH
   T, DA ALLE VARIABLEN BESETZT
    BLEIBEN)
 56
 57 LOAD *"M";2;S$SCREEN$
 58 DIM I$(32)
    LET M=1
 60 FOR N=NA TO NA+31
    LET I$(M)=CHR$ (PEEK N)
    LET M=M+1
    NEXT N
 62 PRINT B$,I$
```

Literaturverzeichnis

LOGAN, Ian Understanding your Spectrum, UK Lincoln 1982
ders. Das Microdrive Universum, München 1984

TOMS, 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