basic für Fortgeschrittene

Ein Kurs zur Erweiterung von Kenntnissen im BASIC-Programmieren

Von Dr. Joachim Baumann und Prof. Dr. Horst Völz

Vorbemerkungen

Die Mehrzahl der jungen Menschen kommt über die Programmiersprache BASIC erstmalig mit einem Rechner in Kontakt. Sie ist sowohl leicht erlernbar als auch auf fast allen Kleinstrechnern unmittelbar verfügbar. Dies dürfte sich auch in den nächsten Jahren kaum ändern. Da die ersten Versuche oft ohne Anleitung erfolgen, gewöhnt man sich leicht einen ungünstigen Programmierstil an, was durch einige Eigenschaften von BASIC selbst gefördert wird. So war es bei den BASIC-Lehrgängen des Rundfunks von Anbeginn ein Ziel, dem entgegenzuwirken.

Die Veröffentlichung des Grundlehrgangs „BASIC – Einmaleins des Programmierens" im Heft URANIA-Extra von 1987 ist bei hoher Auflage auf ein großes Interesse gestoßen. 1988 wurde vom Rundfunk die Sendefolge „BASIC für Fortgeschrittene" ausgestrahlt, die noch stärker auf methodische Aspekte ausgerichtet war. Auf ihrer Grundlage ist analog der folgende URANIA-Lehrgang erarbeitet worden.

„BASIC für Fortgeschrittene" setzt gewisse Grundkenntnisse des BASIC-Programmierens voraus, aber nicht unbedingt den Kurs „Einmaleins des Programmierens" in der URANIA-Extra-Ausgabe von 1987. Verweise im Text beziehen sich auf diesen Grundkurs und stellen zusätzliche Zusammenhänge her, können aber bei Nichtvorhandensein des Heftes auch einfach überlesen werden.

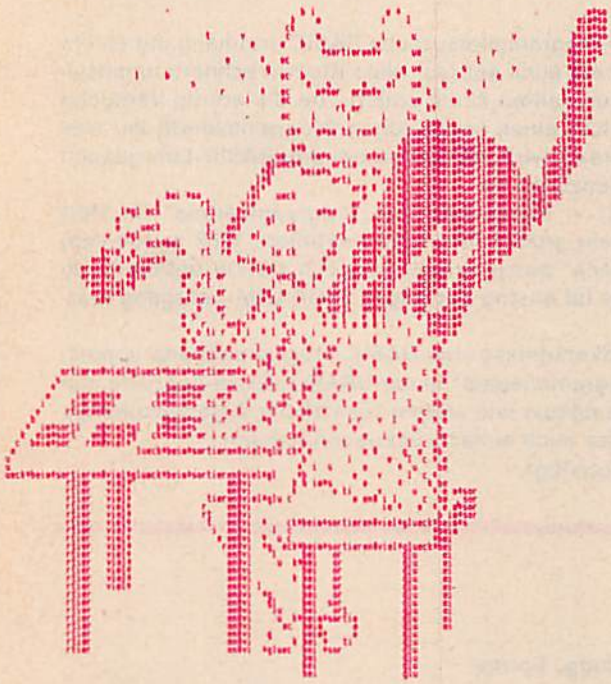
Dem Text ist ein umfangreicher Programmteil hinzugefügt.

Inhalt

1. Sortieren
2. Umgang mit Zahlen – Rundung, Formatierung, Kombinatorik
3. Chaotische Probleme, Fraktale
4. Einbinden von Maschinencode
5. Rekursive Programmierung
6. Analytische Methoden, Formelmanipulation
7. Lernfähige Programme, Künstliche Intelligenz
8. Menu- und Eingabetechniken
9. Maschinennahe Befehle
10. Anpassen von Daten

1. Sortieren

In der Textverarbeitung ist es oft wichtig, Verzeichnisse zu sortieren. So etwas kennen wir bezüglich Adressen, Telefonnummern, Büchern, Schallplatten usw. Wollen Sie z. B. die Telefonnummer eines Freundes wissen, so schauen Sie im alphabetisch geordneten Telefonbuch unter dem Namen nach und finden die entsprechende Nummer. Bei der Post wird aber oft der umgekehrte Vorgang benötigt, und zwar muß man dort auch von einer Nummer auf den Teilnehmer schließen können. Dazu muß die Teilnehmerkartei nach Nummern sortiert werden, damit das entsprechende Verzeichnis (rechentechnisch spricht man von Datei oder File) vorhanden ist.
Der Umgang mit Dateien erfordert weiter eine Vielzahl von Routinen: Die Daten müssen eingegeben und angezeigt werden, in der Datei müssen sich Veränderungen realisieren lassen, in den Daten muß ein Suchen möglich sein, und schließlich müssen Dateien numerisch oder alphabetisch geordnet (sortiert) werden.
Aus der Textverarbeitung ist bekannt, daß für das Sortieren von Dateien etwa 25 % aller Rechenzeit benötigt werden. Den Sortier-Programmen ist also eine große Bedeutung beizumessen, und im Laufe der Entwicklung der Rechentechnik entstand daher eine Vielzahl von Sortieralgorithmen. Sie unterscheiden sich im wesentlichen im Organisationsaufwand, in der Rechenzeit und im benötigten Speicherplatz.



viel glueck beim sortieren

Es bestehe die Aufgabe, ein gut gemischtes Skatblatt in der Reihenfolge Karo-7, -8, -9 usw., dann gleichermaßen für Herz, Pik bis Kreuz-Dame, -König, -As zu ordnen. Sie könnten dabei folgendermaßen vorgehen: Sie legen alle Karten, wie sie vom Stapel kommen, nebeneinander auf den Tisch. Dann suchen Sie die Karo-7 und legen sie an den Anfang einer neuen Reihe darunter. Danach suchen Sie die Karo-8 und legen sie neben die Karo-7 usw. bis zum Kreuz-As. Dieses Vorgehen erfordert viel Zeit und vor allem Platz und ist auf dem Rechner nicht ohne weiteres so zu realisieren. Im Rechner werden vielmehr elementarere Aufrufe, nämlich bevorzugt Vergleichs- und Austauschoperationen zwischen zwei Karten genutzt. Dabei unterscheiden sich die verschiedenen Algorithmen vor allem darin, wie die zwei Vergleichskarten aus dem Stapel ausgewählt werden.
Um die verschiedenen Sortieralgorithmen miteinander zu vergleichen, benötigen wir sowohl eine zufällig geordnete Datenmenge (File) von Zahlen oder Wörtern als auch die gleichartige Reproduktion genau dieser zufälligen Anordnung. Dazu dient das Programm FILE, welches aus zwei Teilen besteht: die Zeilen 790 bis 830 erzeugen entsprechende Zufallszahlen, die Zeilen 900 bis 960 Zufallstexte. Für den Vergleich verschiedener BASIC-Sortier-Routinen muß der entsprechende Teil von FILE nur an die BASIC-Routinen angefügt werden. Dann stehen die Datensätze in den Feldern A(X) bzw. A$(X) bereit.

a) Bubble-Sort

Am Beispiel des Skatblattes funktioniert dieser Algorithmus folgendermaßen:
Der Rechner nimmt die beiden ersten Karten und vergleicht sie. Sind sie falsch angeordnet, so vertauscht er beide. Anderenfalls bleiben beide so liegen. Dann wird die dritte mit der zweiten Karte verglichen und wenn notwendig vertauscht, dann die vierte mit der dritten usw.
Es sei die erste Karte das Kreuz-As. Dann wird sie mit allen anderen Karten im ersten Durchlauf vertauscht, d. h. das As wandert durch die gesamte Kartenreihe wie eine Blase (engl. bubble) ans Ende. Daher hat dieses Sortierverfahren seinen Namen. Doch mit einem Durchlauf allein ist das Skatblatt noch nicht sortiert, lediglich das Kreuz-As ist ans Ende gewandert. Die anderen Karten liegen noch in der alten Reihenfolge. Der Algorithmus muß also mehrfach wiederholt werden und zwar solange bis jede Karte an ihrem Platz liegt. Auf diese Weise können sowohl Zahlen (nach ihrer Größe) als auch Worte (nach dem Alphabet) sortiert werden. Der Algorithmus greift dabei auf die in den Feldern abgelegten Daten zu, und je zwei Feldelemente werden miteinander verglichen, also A(Y) mit A(Y+1). Bei falscher Anordnung werden beide miteinander vertauscht (Zeile 40 in den Programmen ZABUB und WOBUB). Der Austausch geht aber in BASIC nicht ohne weiteres. Hierfür wird eine Hilfsvariable benötigt (A bzw. A$ in Zeile 40). Außerdem wird beim Austausch ein Flag gesetzt (F in 30, 40 und 50); es kennzeichnet, ob bei einem kompletten Durchlauf ein Austausch erfolgt ist (F = 1) oder nicht (F = 0). Solange F = 1 gilt, muß der Algorithmus wiederholt werden. Wenn nach einem Durchlauf F = 0 wird, dann ist das Feld sortiert.
Der Vergleich von Sortierzeiten für Zahlen und Zeichenketten (Strings) zeigt, daß Strings auf diese Weise langsamer sortiert werden. Dabei steigt die Differenz in der Sortierzeit mit wachsender Datenmenge. Der Grund hierfür ist wieder die Garbage Collection (URANIA-Extra S. 39). Sie kann umgangen werden mit dem Algorithmus:

b) Pointer-Sort

Den Karten werden gemäß ihrer zufälligen Folge die Zahlen 1 bis 32 zugeordnet. Das kann man sich so vorstellen, daß die Zahlen auf extra Karten unmittelbar unter den Skatkarten liegen. Wenn nun nach Vergleich zweier benachbarter Skatkarten ein Austausch erfolgen muß, dann werden nicht mehr diese Karten sondern die ihnen zugeordneten Zahlenkarten ausgetauscht, d. h. der Austausch der Skatkarten erfolgt nun indirekt. Nach dem Sortieren steht dann unter Karo-7 die 1, unter Karo-8 die 2 ... unter Kreuz-As die 32.

Diese Pointer-Methode ist programmtechnisch und auch speichermäßig aufwendiger als der einfache Bubble-Sort, denn für die zusätzlichen Pointer wird ein zweites Feld A(Y) neben A$(Y) verwendet. Dieses Zusatzfeld muß zunächst mit A(Y) = Y belegt werden. Die Vorteile bestehen u. a. darin, daß erstens das ursprüngliche Feld nicht verändert wird. Es kann folglich auch nicht während des Sortierens – z. B. bei Stromausfall – zerstört werden. Zweitens wird bei dieser Methode die Garbage Collection vermieden. Die Pointer-Methode ist somit für Strings schneller als der direkte Bubble-Sort. Ein Beispiel ist das Programm BUB2. Eine weitere Modifikation von Bubble-Sort ist:

c) Shaker-Sort

Betrachten wir nochmals das Kreuz-As. Es liegt bei Bubble-Sort nach dem ersten Durchlauf am Ende der Reihe, egal wo es vorher lag. Das heißt, der zweite Durchlauf kann einen Schritt früher abgebrochen werden, der dritte Durchlauf zwei früher usw. Dadurch wird der Algorithmus schneller. Noch effektiver wird dieser Vorgang, wenn der Durchlauf von links nach rechts und dann von rechts nach links vollzogen wird. In diesem Fall bewegen sich also Blasen hin und her, so als ob geschüttelt wird (engl. to shake). Shaker-Sort verkürzt die Sortierzeit gegenüber Bubble-Sort bis zu 50 %. Ein Beispiel zeigt das Programm SHAKER.
Eines der schnellsten Sortierverfahren, das sich international durchgesetzt hat, ist:

d) Quick-Sort

Aus dem ungeordneten Skatblatt wird eine etwa in der Mitte des Stapels liegende Karte als Bezug für die folgenden Prozesse ausgewählt (beispielsweise Herz-König). Dann wird links beginnend eine Karte gesucht, die größer als der Herz-König ist. an der dritten Stelle könnte z. B. Pik-8 gefunden werden. Sie wird gekennzeichnet. Danach wird von rechts beginnend eine Karte gesucht, die kleiner als der Herz-König ist. An vierter Stelle wird nun z. B. Herz-Bube gefunden. Dann werden den Pik-8 und Herz-Bube vertauscht. Dieses Prinzip wird solange fortgesetzt, bis beide Seiten von Herz-König abgesucht sind. Dann befinden sich links von Herz-König nur niedrigere und rechts nur höhere Karten. Anschließend wird dieser Vorgang für die beiden Teilstapel wiederholt. Dabei werden unten z. B. Karo-Bube als Mitte gewählt. Nachdem auch hier der Austausch zwischen links und rechts erfolgt ist, wird die Teilung in den Teilstapeln analog fortgesetzt. Dies ist solange möglich, bis nur noch zwei Karten zu vergleichen sind. Wenn dieses Prinzip für alle Teilstapel durchgeführt ist, sind die Karten richtig angeordnet.
Dieser Algorithmus ist in BASIC deshalb relativ schwer zu programieren, weil er ein typisch rekursives Verhalten besitzt. Rekursives Programmieren ist in BASIC aber indirekt, z. B. über einen Stack, möglich (ausführlicher dazu in Thema 5). Auf dem Stack muß vermerkt werden, an welcher Stelle der Kartenstapel geteilt wurde und gerade richtig sortiert wird und welche Teilstapel noch zu bearbeiten sind. So etwas ist rechentechnisch immer mit relativ großem Organisationsaufwand verbunden. Doch es lohnt sich, denn Quick-Sort ist meist hundertmal und bei großen Dateien sogar tausendmal schneller als die anderen oben genannten Sortier-Verfahren. Durch Quick-Sort kann z. B. mit dem BASIC-Programm QUICK eine Datenmenge von 20 Kilobyte in einigen Minuten sortiert werden. Das vorgestellte Grundprogramm muß aber – wie auch die anderen Programme – für praktische Anwendungen durch spezifische Ein- und Ausgaberoutinen ergänzt werden.

Die verschiedenen Programme unterscheiden sich in der Laufzeit auch deutlich danach, ob das File bereits sortiert oder genau falsch sortiert bzw. zufällig vorliegt. Vielleicht schreiben Sie einmal ein Programm, mit dem Sie diese Einflüsse untersuchen können.

Zusammenfassung Sortieren

1. Dateien oder Files sind eine Zusammenfassung aus mehreren Teilen, dies sind Abschnitte, Sätze, Wörter oder Zahlen.
2. Sie lassen sich nach mehreren Gesichtspunkten sortieren. Zwei davon sind:
   - nach der Größe vorhandener Zahlen oder
   - nach dem Alphabet.
3. In BASIC ist eine alphabet-ähnliche Sortierung leicht zu realisieren. Dabei wird die ASCII-Nummer als Kennzeichen benutzt. Dadurch stehen alle Kleinbuchstaben gemeinsam hinter den Großbuchstaben.
4. Die Eigenschaften der Sortierverfahren betreffen vorrangig drei Punkte:
   - die Sortierzeit,
   - den Organisationsaufwand, d. h. die Größe des Programms,
   - den zusätzlich benötigten Speicherplatz, z. B. für die Anzahl der gebildeten Stapel.
5. Vor dem Sortieren kann die Datei unterschiedliche Zustände besitzen. Sie haben u. a. Einfluß auf die Sortierzeit. Die Datei kann:
   - bereits richtig sortiert sein,
   - genau entgegengesetzt sortiert sein,
   - zufällig angeordnet sein.
6. Das einfachste Sortierverfahren ist Bubble-Sort. Sein Algorithmus lautet:
   - A: Es wird mit dem ersten Wort, also n = 1, der Datei begonnen.
   - B: Es wird immer das n-te Wort mit dem (n+1)-ten verglichen.
   - C: Sind beide richtig angeordnet, so geht es bei E weiter.
   - D: Sind sie falsch angeordnet, so werden die Wörter vertauscht, und es wird ein Flag gesetzt.
   - E: Es wird n um 1 erhöht. Ist das letzte Wort erreicht, so wird bei F, sonst bei B fortgefahren.
   - F: Wenn das Flag gesetzt ist, beginnt man wieder bei A, sonst ist die Datei sortiert.
7. Es gibt zwei typische Operationen beim Sortieren:
   - Den Vergleich und
   - Den Austausch.

   Beide benötigen Zeit. Der Austausch kann bei Wörtern (im Gegensatz zu Zahlen) sehr nachteilig durch die Garbage-Collection beeinflußt werden. Dies kann vermieden werden, wenn der Austausch nur symbolisch erfolgt. Zu diesem Zweck nutzt man Zeiger, engl. Pointer, die auf die richtige Reihenfolge der Worte verweisen. Nur sie werden dann beim Sortieren verändert.
8. Beim Bubblesort gibt es mehrere Varianten, die den Algorithmus zwar komplizierter, dafür aber schneller machen. Hierzu gehören u. a.:
   - In der Datei wird nur bis zur erforderlichen Länge sortiert.
   - In der Datei wird abwechselnd vorwärts und rückwärts sortiert.

Im zweiten Fall wird von SHAKER-Sort gesprochen
9. Alle Sortierverfahren unterscheiden sich u. a. dadurch, über welchen Abstand hinweg Wörter verglichen und/oder ausgetauscht werden.
10. Ein sehr schnelles, aber kompliziertes Verfahren ist QUICK-Sort. Sein Algorithmus läuft wie folgt ab:
A: Das ganze File wird zunächst als ein Stapel betrachtet.
B: Aus den Wörtern eines Stapels wird in etwa ein mittleres als Bezug ausgewählt.
C: Es werden zwei Stapel gebildet, welche jeweils Wörter enthalten, die links bzw. rechts vom Bezug stehen.
D: Für jeden Stapel wird iterativ der Vorgang von B beginnend solange wiederholt, bis nur ein oder zwei Wörter existieren.
E: Nun werden die Wörter schrittweise von den Stapeln richtig hintereinander angeordnet.
Es sei bemerkt, daß im allgemeinen schrittweise immer nur ein Stapel bis auf ein oder zwei Wörter geteilt wird. Der Stand der anderen Stapel wird zwischendurch auf einem Stack gespeichert.

2. Umgang mit Zahlen – Rundung, Formatierung, Kombinatorik

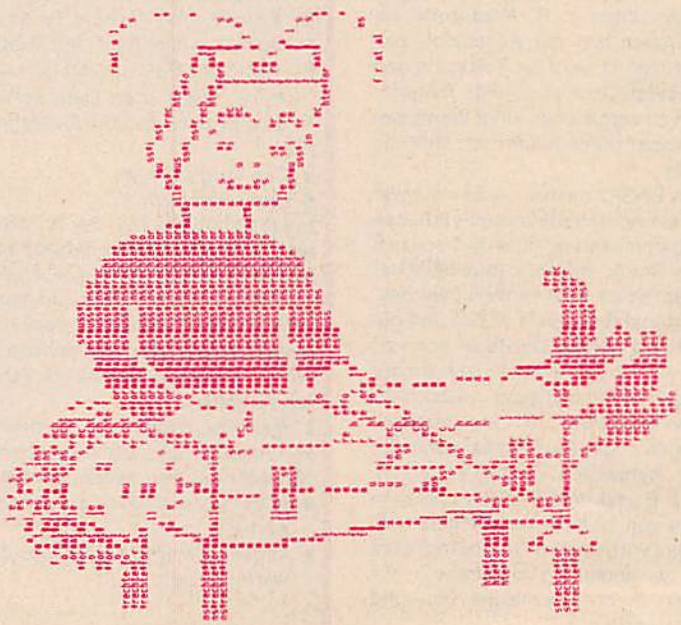
Der BASIC-Interpreter der Kleincomputer hat für die Darstellung von Zahlen sechs Ziffern zur Verfügung (z. B. ergibt PRINT PI die Zahl 3.14159). Diese begrenzte Darstellungsmöglichkeit von Zahlen wirkt sich bei vielen Anwendungen nachteilig aus. Bei finanzökonomischen Berechnungen ist mit größeren Geldbeträgen, z. B. mehreren tausend Mark, auf Pfennige genau umzugehen. Hierzu reichen die sechs gültigen Ziffern der BASIC-Interpreters bei weitem nicht aus. Auch die formatgerechte Anordnung von Zahlen auf dem Bildschirm oder beim Druck sowie der Umgang mit sehr großen oder sehr kleinen Zahlen bringen oft Probleme, die der BASIC-Interpreter nicht bewältigen kann. Durch eine geschickte Programmierung läßt sich aber oft Abhilfe schaffen. Einige ausgewählte Methoden werden im folgenden behandelt:

a) Runden von Zahlen

In den Anfängen der Rechentechnik gab es einen trivialen Fall von Computerkriminalität. Stellen Sie sich vor, Sie haben 757,31 Mark auf Ihrem Konto und sollen am Jahresende die Zinsen von 3 % gutgeschrieben bekommen. Gemäß unserem BASIC würden das 757,31 * Ø,Ø3 = 22,7196 Mark sein. Was soll nun mit den Hunderstel Pfennigen geschehen? Der kriminelle Programmierer schnitt alle Hundertstel und Tausendstel Pfennige ab und überwies sie auf sein Konto. Da er in einer großen Bank arbeitete, wurde er schnell sehr reich. So etwas darf natürlich nicht sein. Aus dem Grunde gibt es exakte Rundungsroutinen, bei denen im Mittel weder die Bank noch der Kontoinhaber übervorteilt wird. Alle Anteile, die kleiner als Ø,ØØ5 Mark sind, werden gleich Null gesetzt und alle, die größer sind, auf einen Pfennig aufgerundet. Ähnliche Rundungsroutinen werden in vielen anderen Fällen benötigt. Einen Algorithmus für das Runden von Zahlen enthält Punkt 1 der Zusammenfassung. Ein Beispiel für diesen Algorithmus ist im Programm RUND realisiert (Zeile 5Ø).

b) Formatieren von Zahlen

Für die Darstellung von Zahlen benutzt der BASIC-Interpreter automatisch drei Zahlenbereiche, den der ganzen Zahlen, den Gleitkommabereich und die Exponentialdarstellung (URANIA-Extra S. 9). Die Anordnung auf dem Bildschirm erfolgt ebenfalls gemäß feststehender Routinen des Rechners. Dies ist jedoch oft nachteilig für eine übersichtliche Anordnung, beispielsweise von Zahlenkolonnen in Tabellen. Formatieren heißt in diesem Sinne, die Zahlen nach genau definierten Regeln auf dem Bildschirm anzuordnen. Sehr große BASIC-Dialekte verfügen hierfür über die PRINT USING Anweisung. Bei den Kleinstcomputern fehlt sie, kann aber mittels eines BASIC-Programms nachgebildet werden. Die Grundidee geht davon aus, die Zahlen in Zeichenketten (mit STR$) umzuwandeln.



alles muss ins rechte format

Für ein rechtsbündiges Formatieren wird dann die Länge des Strings (d. h. der zugehörigen Zahl) bestimmt und danach entsprechend der Anzahl der gültigen Stellen die Position im Tabulator gesetzt. Das ist im Beispielsprogramm REFOR (Zeile 40) realisiert.
Eine zweite Formatierungsmöglichkeit für Zahlen besteht darin, sie so untereinander anzuordnen, daß die Dezimalpunkte an gleicher Stelle stehen (Programm PUFOR). Auch hierbei werden die Zahlen zuerst in Zeichenketten umgewandelt und ihre Längen bestimmt (Zeile 50). Danach wird ermittelt, an welcher Stelle des Strings der Dezimalpunkt steht (Zeile 50), und anschließend der benötigte Tabulator gesetzt (Zeile 60).

c) Kombinatorik

Die Wahrscheinlichkeit, im Lotto einen Fünfer zu erzielen, kann mathematisch berechnet werden. Es ist dabei zu berücksichtigen, daß aus der vorhandenen Anzahl von Zahlen unterschiedliche Fünfergruppen gebildet werden müssen, wobei in einer Gruppe dieselbe Zahl nicht zwei- oder gar mehrmals vorkommen darf. Da es eine Vielzahl solcher Gruppen gibt, ist die Wahrscheinlichkeit für einen Fünfer dementsprechend klein. Auch bei der Aufgabe, aus den 26 Buchstaben des Alphabets Wörter der Länge fünf (fünf Buchstaben) zu bilden, stehen wir vor dem Problem einer sehr großen Anzahl von Möglichkeiten. Wobei hier zusätzlich zu unterscheiden ist, ob in den zu bildenden Wörtern gleiche Buchstaben mehrfach auftreten dürfen oder nicht.
Das mathematische Gebiet, in dem die Anzahl solcher Abwandlungen systematisch untersucht wird, ist die Kombinatorik. Es wird dabei nur von einer Menge von Elementen ausgegangen. Je nachdem, welche Anzahl von Elementen in welcher Anordnung zugelassen werden, unterscheidet die Kombinatorik zwischen Permutationen, Variationen und Kombinationen. Definitionen hierfür sind in Punkt 3 der Zusammenfassung enthalten. Wie den Definitionen zu entnehmen ist, gehören zu den einzelnen Fällen Formeln, die zu großen Zahlen führen können:

- die Fakultät N! = 1*2*3*...*N
- die Potenzfunktion x^k
- der Binominialkoeffizient (n über k)

$$\binom{n}{k} = \frac{n*(n-1)*\ldots*(n-k+1)}{1*2*\ldots*k}$$

Für die Berechnung derartig großer Zahlen wäre es denkbar, eine Stringarithmetik wie bei der Formatierung von Zahlen zu verwenden. Doch das erweist sich als wenig effizient. Es ist in BASIC günstiger, alle diese Zahlen in Dreiergruppen zu zerlegen. Für die vollständige Zahl wird dann ein Feld verwendet, in dessen Elemente je eine Dreiergruppe abgelegt wird. Mit den einzelnen Gruppen kann dann wie bei der schriftlichen Rechnung operiert werden. Mit der so möglichen Addition, Subtraktion, Multiplikation und Division vielstelliger Zahlen und der Anwendung kombinatorischer Formeln erhält man die volle Stellenzahl. Diese Methode ist im Programm KOMBA realisiert. Es gestattet Rechnungen bis zu mehr als tausend Stellen.
Oftmals ist es nützlich, große und irrationale Zahlen mit vielen Stellen nur näherungsweise zu berechnen. Hierfür einige Programmbeispiele:
Das Programm FAKUL verwendet die Stirlingsche Formel. Mit ihr ist es in relativ kurzer Zeit möglich, die Fakultät auch extrem großer Zahlen näherungsweise zu bestimmen.
Von großem theoretischen Interesse sind die irrationalen Zahlen, denn sie haben unendlich viele Stellen ohne Periodizität. Zu ihnen gehören z. B. Pi und die Eulersche Zahl e = 2.71828... Um möglicherweise doch eine Gesetzmäßigkeit in der Ziffernfolge zu entdecken, wurde und wird immer wieder versucht, mehr Stellen zu berechnen. So wurde Pi bereits 1949 mit der ENIAC auf 2037 Stellen berechnet. Im Jahre 1967 waren etwa eine halbe Million und heute sind über 130 Millionen Stellen dieser Zahl bekannt. Eine Gesetzmäßigkeit ist bisher nicht aufgetreten. Solche Berechnungen dienen aber auch für verschiedene Computertests.
Das Programm PI nutzt die Beziehung von Gregory (1671): Pi = 4*ATN(1). Die Reihe für ATN(1) konvergiert jedoch sehr langsam. Deshalb wird hier die abgewandelte Formel von Machim (1706) genutzt (REM-Zeilen 240–300). In den Feldern A und B werden die Zwischenergebnisse abgelegt, und das Ergebnis steht im Feld C.
Das Programm EULER berechnet näherungsweise e. Dazu wird eine Arithmetik genutzt, die ähnlich wie im Programm KOMBA arbeitet. Auf die zugehörige Reihenzerlegung e ist in den REM-Zeilen 110–130 hingewiesen. Die Zwischenwerte werden schrittweise im Feld C und das Ergebnis im Feld B abgelegt.
Es sei bemerkt, daß bei den 3- bzw. 2-stelligen Zahlenblocks in den hier angeführten einfachen Programmbeispielen ein Schönheitsfehler existiert. Eine Blockgruppe 045 wird nur als 45 ausgegeben. Es werden also führende Nullen eines Blocks unterdrückt. Vielleicht versuchen Sie einmal selbst, mit einer kleinen Stringumwandlung diesen Effekt zu beheben.

Zusammenfassung Umgang mit Zahlen

1. Runden bedeutet immer, Zahlen auf eine bestimmte Genauigkeit festzulegen. Dazu diene ein Stellenwert X, z. B. 100, 10, 0.1 oder 1/100. Dann wird der folgende Algorithmus genutzt:
A: Der vorhandene Zahlenwert wird durch X dividiert.
B: Es wird 0,5 dazu addiert.
C: Es wird der Integerwert gebildet.
D: Der neue Zahlenwert wird mit X multipliziert.
2. Formatieren liegt immer dann vor, wenn Zahlen, Wörter oder Texte in ein Schema, eine Tabelle oder ein Formular eingepaßt werden müssen. Bei Zahlen bedeutet dies außerdem, daß sie auf genau definierte Positionen zu setzen sind.
3. Wenn verschiedene Elemente, z. B. A, B, C und D einzeln oder mehrfach existieren, so können sie in unterschiedlicher Weise zusammengefaßt werden. Es können aus ihnen also Wörter verschiedener Länge und Gestalt erzeugt werden. Entsprechend den dabei verwendeten Regeln sind zu unterscheiden: Permutationen, Variationen, Kombinationen.
In den drei Fällen sind außerdem zu unterscheiden, ob sich gleiche Elemente in einem Wort wiederholen dürfen oder nicht.
- Bei den PERMUTATIONEN müssen alle vorhandenen Elemente zum Aufbau eines Wortes benutzt werden. Wenn gleiche Elemente existieren, sind AABCD, ABACD, DABBC usw. gültige Wörter. Anderfalls nur ABCD, BCAD, CDAB usw. In diesem Fall beträgt die Anzahl der Permutationen N!, also hier
4! = 1*2*3*4 = 24.

- Bei den VARIATIONEN werden aus n Elementen Gruppen mit k Elementen bei k<n gebildet. Hier ist die Reihenfolge der Elemente wichtig. Sollen aus den 4 Elementen A, B, C, D drei ausgewählt werden, so sind im ersten Fall AAA, AAB, ACA unterschiedliche Wörter. Dann gibt es n^k mögliche Variationen. Andernfalls werden Elemente der Art AAB, ABA und BAA als gleich betrachtet. Sie werden dann als nur ein Element gezählt.
- Bei den KOMBINATIONEN werden im Gegensatz zu den Variationen nur unterschiedliche Elemente zugelassen. Ohne Unterscheidung der Reihenfolge sind also ABC, BCA, BAC verschiedene Wörter. Ansonsten gelten sie als gleiche Wörter. Dann existieren n über k Kombinationen.

4. Es können folgende Zahlen unterschieden werden:

- Ganze Zahlen, auch Integer genannt, wie 3, 127 oder −68.
- Brüche, rationale Zahlen, wie 1/3, 1/385 usw. Ihre Dezimaldarstellung bricht irgendwann ab oder führt zu Perioden, wie z. B. 1/3 = .3333333 ... oder 1/7 = Ø.142857 142857 142857 ...
- Irrationale Zahlen. Sie sind durch einen unendlich langen Dezimalbruch ohne Periode gekennzeichnet. Beispiele hierfür sind SQR(2), PI oder die Eulersche Zahl e.

3. Chaotische Probleme, Fraktale

Viele Erscheinungen in der Natur können durch chaotische Systeme beschrieben werden. Anwendungsgebiete hierfür sind beispielsweise Probleme der Evolution, des Magnetismus oder der Wetterprognose. Die Beschreibung der Systeme kann mit mathematischen Modellen (Gleichungen und Gleichungssysteme) erfolgen, deren Lösungen allerdings stark von den Anfangs- und Randbedingungen abhängen. Die Lösungen in unterschiedlichen Gebieten können stabil oder instabil, konvergent oder divergent sein. Bekanntlich ist für Mitteleuropa eine Wettervorhersage im April relativ schwierig, da sich Hoch- und Tiefdruckgebiete rasch abwechseln. Die Berechnung o. g. Modelle ist stets mit großem rechentechnischen Aufwand verbunden.

Hier soll ein Teilgebiet dieser Problematik betrachtet werden, das zu speziellen, oft ästhetischen Grafiken führt und wozu auch noch Kleinstrechner eingesetzt werden können. Gemeint sind die Fraktale, die erstmalig 1980 von B. B. Mandelbrot mittels Computer sichtbar gemacht wurden. Der Begriff ist von dem Wort Fraktion abgeleitet, wie er z. B. auch in der Politik, der Chemie oder der Medizin im Sinne von etwas Geteiltem oder Zerbrochenem gebräuchlich ist.

Für die Erzeugung von Fraktalen wird eine Pixelgrafik benötigt. Der KC 85/1 und KC 87 kommen daher bei diesem Kapitel etwas zu kurz. Es wird aber gezeigt, daß sie trotzdem nutzbringend eingesetzt werden können.

Das mathematische Grundproblem der Fraktale kann vereinfacht an der Gleichung A = X*A*(A-1) verdeutlicht werden. Mit dieser Gleichung wird für einen Eingabewert X (z. B. X = 1) und für einen Anfangswert von A (z. B. A = Ø die Variable A iterativ immer wieder neu berechnet. Nach der ersten Iteration ist A = Ø.25, nach der zweiten Ø.19 usw. Bei der Wiederholung konvergiert A gegen Null. Für andere Werte von X fürht die Iteration von A zu recht unterschiedlichen Ergebnissen:

- Es können mehrere periodische Zahlenwerte vorkommen,
- A kann divergieren, oder
- es tritt stochastisches Verhalten der Iterationswerte auf.

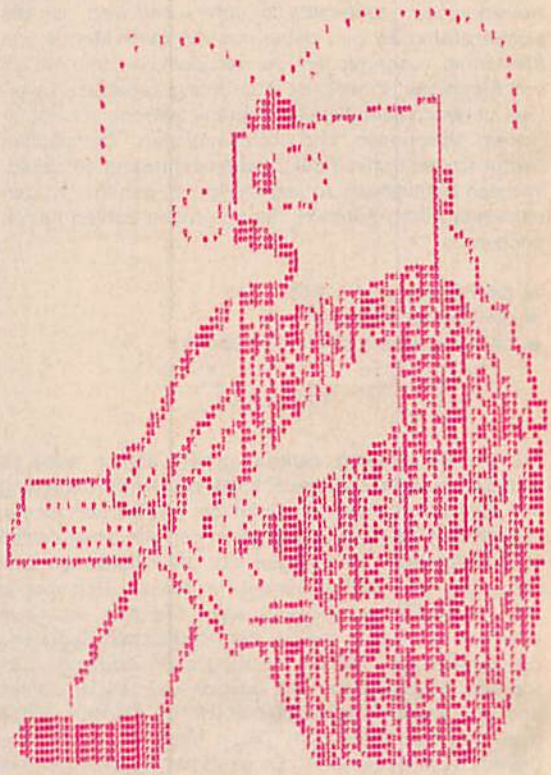
Einen Überblick gibt Punkt 1 der Zusammenfassung. Die Übertragung auf das Zweidimensionale führt mit einer speziellen grafischen Methode zu ästhetisch anmutenden Bildern, wobei sich zeigt, daß insbesondere die Gebiete mit stochastischem Verhalten von Bedeutung sind. Dazu brauchen wir jetzt aber zwei Variablen (z. B. A und B), zwei Eingabeparameter (z. B. X und Y) und zwei Gleichungen für die Iteration der Variablen. Das verbreitetste Beispiel ist das sog. Apfelmännchen

$$A = A*A + B*B - X$$
$$B = 2*A*B - Y$$

Für die grafische Umsetzung ist jetzt anders vorzugehen. Während im eindimensionalen Fall die Iterationszyklen direkt genutzt werden können, gibt es im Zweidimensionalen für die Anzahl der Iterationszyklen keine direkte Darstellungsmöglichkeit. Es existieren jedoch indirekte Methoden, die zu qualitativ unterschiedlichen Bildern für ein Fraktal führen:

a) ASCII-Methode

Zur Darstellung der Fraktale auf dem Bildschirm (24 × 40 Zeichen beim KC 87 oder 32 × 40 beim KC 85/2-4) können z. B. die ASCII-Zeichen genutzt werden. Die Anzahl der Iterationszyklen wird nun durch die Auswahl des jeweiligen ASCII-Zeichens verdeutlicht (A für wenige, Z für viele Zyklen). Für den Fall der Konvergenz an der berechneten Stelle wird ein Kleinbuchstabe, bei divergentem Verhalten ein Großbuchstabe gesetzt (unterschiedliche Klein- bzw. Großbuchstaben für verschiedene Konvergenz- bzw. Divergenzgeschwindigkeiten). Das verbleibende Gebiet (stochastisches Verhalten) wird durch ein deutlich unterscheidbares Sonderzeichen gekennzeichnet. So entsteht eine übersichtliche Darstellung des Funktionsverhaltens. Vereinfachend sei das Bild



fraktalprogramme zeigen probleme

auch als Fraktal bezeichnet. Der große Vorteil dieser Methode besteht einerseits darin, daß relativ schnell die Gebiete unterschiedlichen Verhaltens bildlich sichtbar werden und daß andererseits diese Methode auf allen KC 85 und dem KC 87 anwendbar ist. Ein Beispiel der ASCII-Methode ist im Programm TEFRA realisiert:
In Zeile 20 wird durch PEEK (−5) abgefragt, welcher Rechner vorliegt, und die richtige Zeilenzahl gesetzt. In Zeile 80 steht eine leicht abgewandelte Formel für das Apfelmännchen:

C = A*A−B*B−X
D = 2*A*B−Y

So kann die Konvergenzgeschwindigkeit gemessen werden, indem bestimmt wird, wie weit C von A bzw. D von B entfernt ist. Hierzu wird die Anzahl der Iterationsschritte bestimmt und der entsprechende Kleinbuchstabe gesetzt (Zeile 90). In ähnlicher Weise wird die Divergenzgeschwindigkeit bis zu einer vorgegebenen oberen Grenze bestimmt (Zeile 100).
Mit dem Programm TEFRA können in verschiedenenen X-Y-Intervallen (INPUT von Zeilen 30 und 40) recht unterschiedliche und interessante Bilder erzeugt werden. Aber auch andere Formeln in Zeile 80 führen zu neuartigen Grafiken, z. B.:

C = A+B−X : D = A*B−Y (Papagei)
C = A*A−B*B−Y : D = A*A+B*B+X (Fisch)

Lassen Sie Ihrer Phantasie freien Lauf und finden Sie mit anderen Formeln selbst „Ihr" Fraktal. Verschiedene Formeln finden Sie auch in der Zeitschrift SPECTRUM 8/1988.

b) 08-15-Methode

Eine sehr häufig gebrauchte Methode, Fraktale zu erzeugen, besteht darin, bei näherungsweise erreichter Konvergenz bzw. Divergenz die Iteration abzubrechen. Dann wird die Anzahl der dazu benötigten Zyklen bestimmt. War sie geradzahlig, so wird für diesen Ort X, Y ein Punkt gesetzt, anderenfalls bleibt er ungesetzt. Die Darstellung der Anzahl der Iterationszyklen erfolgt hier also „digital". Wegen der benötigten Pixelgrafik sind der KC 87 und der KC 85/1 dafür nicht geeignet.
Die Möglichkeit, hochauflösende Grafik dafür anzuwenden, ist im Programm FRAKT für das Apfelmännchen realisiert. Wie auch mit dem Programm TEFRA können durch Variation der Anfangsparameter in unterschiedlichen X-Y-Bereichen mit dem Programm FRAKT recht interessante und z. T. ästhetische Grafiken erzeugt werden. Da diese Methode sehr viel Zeit benötigt, kann zur Auswahl der interessanten Gebiete die ASCII-Methode sehr nützlich sein.

c) Gebirgs-Methode

Eine sehr anschauliche Methode besteht darin, für die Anzahl der Iterationen eine dritte Koordinate einzuführen. Da aber das so entstehende dreidimensionale Bild nicht direkt auf dem Monitor dargestellt werden kann, führen wir eine Projektion mit Drehung (z. B. 45°) aus. So kann das Fraktal perspektivisch wie ein Gebirge dargestellt werden, bei dem die divergenten Gebiete in „Tälern" liegen, die stochastischen und konvergenten dagegen in den „Bergen". Hierin begründet sich auch die Anschaulichkeit dieser Methode.
Ein Problem besteht jedoch darin, daß durch „hohe Berge" verdeckte Gebiete nicht mehr zu sehen sein dürfen. Vom Programm her muß dies durch eine besondere Technik realisiert werden. Man spricht in diesem Fall von der Darstellung mit verdeckten Linien (engl. hidden lines). Im Programm FRAGEB (abgeleitet von Fraktalgebirge) dient hierfür ein Höhenvektor, der im Feld S (320) in Zeile 10 abgelegt wird. Gleichzeitig ist im Programm FRAGEB die 08-15-Methode eingebaut, so daß mit dem Programm ein Fraktal auf zweierlei Arten sichtbar gemacht werden kann. Hierdurch kann ein tieferes Verständnis für das Fraktal gewonnen werden.

d) Farb-Methode

Besonders wirkungsvoll sind farbige Fraktale. Dabei wird die Anzahl der Iterationszyklen, ähnlich wie bei einem Atlas, durch Farben dargestellt. Im Gegensatz zur Landkarte, auf der Gebirge unterschiedlich braun bis rot, flache Gebiete verschieden grün und die Gewässer verschieden blau gekennzeichnet sind, ist für Fraktale die Farbe frei wählbar.
Wegen der begrenzten Farbbildauflösung des KC 85/3 ist die Erzeugung von farbigen Fraktalen auf direktem Weg jedoch nur sehr begrenzt möglich. Wir müsssen daher einen mittelbaren Weg wählen. Die Idee besteht darin, für verschiedene Iterationszyklen, die zur Berechnung der Fraktalpunkte benötigt werden, unterschiedliche Farben zu wählen. Die Gebiete gleicher Farbe können in einer Matrix abgelegt und danach einzeln ausgedruckt werden. Auf diese Art erhält man mehrere Farbauszüge des Fraktals, und es ist „nur noch" ein drucktechnisches Problem, hieraus ein Farbbild herzustellen. Diese Methode ist im Programm FRAKTAL realisiert. Im Programm wird die Anzahl der Zyklen durch fünf geteilt und der dabei bleibende Rest dient als Grundlage für die Farbwahl (Zeile 140). Die fünf Farbauszüge werden in der Matrix A (31, 319) abgelegt und können dann einzeln ausgedruckt werden.

e) Hilfsmethode für KC 85/1 und KC 87

Obwohl es wegen der fehlenden Pixelgrafik beim KC 85/1 und KC 87 nicht möglich ist, hochaufgelöste Fraktale auf dem Bildschirm darzustellen, können diese Rechner jedoch für die Berechnung der Bildpunkte dienen (Programm FRAK87). Die Bildpunkte werden in einer Matrix A(15,255) abgelegt. Diese Matrix wird dann in einen KC 85/2 oder KC 85/3 eingelesen (Programm FRAK3), und das Fraktal kann so auf dem Bildschirm sichtbar gemacht werden.
Der Vorteil des KC 85/1 bzw. KC 87 besteht hierbei in der etwas höheren Rechengeschwindigkeit (Taktfrequenz 2,5 gegenüber 1,75 MHz).

f) Hilfsprogramme für KC 85/2,3

Wegen der langen Rechenzeiten für ein Fraktal ist es nützlich, ein fertiges Bild auf Kassette zu speichern. Dazu dient ein Maschinenprogramm, das in den FRAKT und FRAGEB in der Zeile 0 abgelegt ist.
Nach der Berechnung muß in das Betriebssystem zurückgekehrt werden. Dort kann das Bild abgespeichert und hinterher auch wieder eingelesen werden. Mit COPY kann es im Betriebssystem jederzeit schnell wieder sichtbar gemacht werden.

Zusammenfassung
Chaotische Probleme, Fraktale

1. Die Anweisung A = X*A*(A−1) besitzt bei einem Startwert A = .5 in einer Iterationsschleife je nach X das folgende Verhalten:

- Für 0<X< = 1 konvergiert A gegen 0, wobei die Geschwindigkeit hierfür beachtlich von X abhängt.
- Für 1<X<1.7 zeigt A ein periodisches Verhalten. Nach einem Einschwingvorgang werden 2, 4, 8 oder auch mehr Zahlenwerte periodisch angenommen.

- Für $1.7 < X < 2$ verhält sich A stochastisch, d. h., es ist keine Regelmäßigkeit in der Abfolge der A zu erkennen.
- Für $X < 2$ wächst A über alle Grenzen. Es divergiert gegen unendlich. Beim Rechner tritt folglich Überlauf auf. Die Geschwindigkeit der Divergenz ist wieder von X abhängig.

2. Den Fraktalen liegt folgendes Prizip zugrunde:

A: Es wird ein X- und Y-Bereich für die Eingangsgrößen gewählt. Dieser wird punktuell auf die Pixel des Bildschirmes abgebildet.

B: Es werden zwei Formeln genutzt:
A = F(A,B,X,Y) und B = G(A,B,X,Y)

C: Es werden die zwei Iterationsvariablen A und B mit Startwerten belegt.

D: Bei der Iteration werden die Konvergenz- und/oder Divergenzgeschwindigkeit bestimmt.

E: Aus diesen Geschwindigkeiten wird die Färbung der Pixel festgelegt.

Für ein stabiles Verhalten ist es günstig, eine maximale Zykluszahl und eine größte Zahl für die Divergenz festzulegen.

3. Die Berechnungen bei den Fraktalen dauern für jeden Punkt und damit erst recht für die Fläche sehr lange. Weiter ist bei neuen Bereichen und Formeln vor der Rechnung nichts über das Aussehen des Fraktals bekannt. Deshalb sind insbesondere bei Kleinrechnern Testmethoden erforderlich. Hierfür wurde das Programm TEFRA entwickelt.

4. Anschaulich ist die Konvergenzgeschwindigkeit auch als Höhe eines Gebirges zu interpretieren. Je mehr Zyklen benötigt werden, desto höher ist hier der Berg. Eine Darstellung mit verdeckten Linien ermöglicht das Programm FRAGEB. Es enthält eine zweite Routine, die dasselbe Gebiet als Fraktal darstellt. Der Vergleich beider Bilder ermöglicht ein tieferes Verständnis der Problematik.

4. Einbinden von Maschinenprogrammen

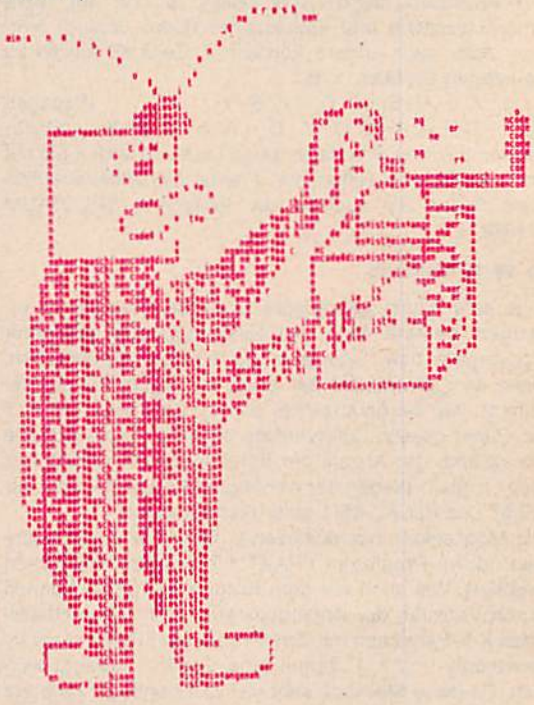
Das Einbinden von Maschinenprogrammen in BASIC-Programme bringt zahlreiche Vorteile mit sich, u. a.:

- höhere Rechengeschwindigkeit
- geringeren Speicherbedarf
- neue Qualität der Programmierung
- Lösung von Problemen, die in BASIC nicht realisierbar sind.

Maschinenprogramme bestehen aus einem Maschinencode (MC), der in der Regel aus einem in Textform geschriebenen Quellprogramm gewonnen wird. Dieser wichtige Teil der Maschinenprogrammierung (Assembler) wird hier nicht behandelt; der MC wird vielmehr als gegeben betrachtet. Er muß wesentlich stärker als ein BASIC-Programm auf die spezifischen Eigenschaften des Rechners Rücksicht nehmen. Da unsere Kleinstrechner alle mit der CPU U880 (Z-80) ausgestattet sind, ist es möglich, sowohl für den KC 85/2,3 als auch für den KC 85/1 bzw. KC 87 einheitliche Maschinen-Programme zu schreiben, die außerdem mit dem jeweiligen BASIC-Programm verbunden werden müssen. Beim Einbinden von MC in BASIC-Programme kommt es darauf an, das Maschinenprogramm an einer bestimmten Stelle fest im Speicher abzulegen und zwar so, daß beide Routinen (BASIC und MC) in einem File als Einheit vorliegen. Hierzu existiert eine Vielzahl von Methoden, die unterschiedliche Vor- und Nachteile besitzen. Die wichtigsten seien hier vorgestellt:

a) Ablegen von MC in DATA-Zeilen

Die wohl einfachste Methode zum Einbinden von MC in ein BASIC-Programm besteht darin, den MC in DATA-Zeilen abzulegen. Damit ist der MC selbst Teil des BASIC-Programms und kann mit READ gelesen werden. Dabei ist es üblich, die ursprünglichen Hex-Zahlen des MC als Dezimalzahlen in den DATA-Zeilen abzulegen. Diese Methode ist jedoch infolge der außerhalb des Rechners erfolgenden Umrechnung von hexadezimal in dezimal sehr fehleranfällig und nicht gut übersehbar. Vorteilhafter ist es, die Hex-Zahlen des MC unmittelbar in die DATA-Zeilen zu schreiben. Dabei erweist es sich als günstig, den Hex-Code halbbyteweise in zwei ASCII-Zeichen zu zerlegen. Danach kann mit einem BASIC-Programm selbst die Umrechnung von hexadezimal nach dezimal erfolgen. Die so errechneten Dezimalzahlen müssen dann mit dem BASIC-Programm an die richtigen Speicher-Stellen im Rechner gepoket werden (zu den maschinentechnischen Befehlen ausführlicher in Thema 9).



dies ist ein angenehmer maschinencode

Eine Anwendung dieser Methode zeigt das Programm RASTER für den KC 85/2,3. Durch die Beeinflussung des Farbbytes wird auf dem Bildschirm eine gut sichtbare Struktur erzeugt, die ohne sonstige Änderungen auf dem Bildschirm abwechselnd ein- und ausgeschaltet werden kann. Obwohl das Programm RASTER nur auf dem KC 85/2,3 lauffähig ist, kann der MC mit anderen Programmen auch beim KC 85/1 bzw. KC 87 in das BASIC-Programm eingebunden werden. Für jede Anwendung ist nur ein entsprechender MC in DATA-Zeilen abzulegen.

b) Ablegen von MC in einer REM-Zeile

Der BASIC-Interpreter übergeht alles, was nach REM steht. Dadurch ist es möglich, auch hier den MC abzulegen. Weil der MC meist auf einer festen Adresse abgelegt sein muß, sollte die entsprechende REM-Zeile die erste Zeile des BASIC-Programms sein (z. B. die Zeile 0, denn davor gibt es keine andere). Mit dieser Methode können ebenfalls kombinierte MC-BASIC-Programme

geschrieben werden, die auf allen KC-Typen lauffähig sind. Als praktisch günstig zum Schreiben von MC in einer REM-Zeile hat sich folgende Arbeitsweise bewährt:

1. Initialisieren von BASIC
2. Zeile Ø schreiben: Ø REM::::::: usw.
3. BYE
4. MC ab 4Ø6 H mit 3 Nullen festlegen
6. REBASIC
7. Schreiben des BASIC-Programmes

Für einen längeren MC ist es auch möglich, sich über zwei oder mehr Zeilen aus Doppelpunkten zunächst genügend Speicherplatz zu reservieren. Durch die Ablage von drei aufeinanderfolgenden Nullen wird immer automatisch mit REBASIC die richtige Zeilenlänge des MC realisiert. Falls noch nicht der exakte Maschinencode feststeht, ist es auch möglich, sich zunächst mit Doppelpunkten (3A) noch eine Reserve für eventuelle Änderungen aufrechtzuerhalten.
Für den MC selbst sind einige Besonderheiten zu berücksichtigen. Er darf keine Null enthalten. Sie würde der BASIC-Interpreter als Zeilenende verstehen und dadurch zumindest den MC zerstören. Oft werden dabei aber auch zusätzlich seltsam anmutende Zeilenzahlen entstehen, die das ganze Programm unbrauchbar machen. In vielen Fällen muß beim KC 85/2 und 3 der IMR (Bildwiederholspeicher) angeschaltet werden. Schließlich müssen alle verwendeten Register vor der Nutzung mit PUSH gerettet und mit POP am Ende zurückgegeben werden.
Die Methode des Ablegens von MC in einer REM-Zeile ist in den Programmen FUNKTION und FNSA realisiert. Beide Programme dienen zur effektiven Arbeit mit definierten Funktionen.
Will man in einem BASIC-Programm verschiedene Funktionen, die nicht vom BASIC-Interpreter bereitgestellt werden, verwenden, so werden sie normalerweise als definierte Funktionen in BASIC-Zeilen abgelegt. Werden andere Funktionen gebraucht, muß die entsprechende Zeile neu programmiert werden. Das ist sowohl umständlich als auch gefährlich. Hier bietet BASIC als interpretierende Programmiersprache einen großen Vorteil, in dem die neue Funktion in die entsprechende Zeile gepoket wird. Dazu bietet sich folgender Lösungsweg an: Die Zeile mit der definierten Funktion wird dazu wie folgt geschrieben:
1Ø DEF FNA(X)=X:!:::::: usw.
Die vielen Doppelpunkte dienen wieder als Platzhalter für die dorthin zu pokende eventuell lange Funktion. Innerhalb des Programms wird ein INPUT dazu verwendet, die zu definierende Funktion in einer Zeichenkette A$ zu übergeben. A$ befindet sich dann automatisch im BASIC-Puffer. Der BASIC-Interpreter besitzt eine Routine, welche den BASIC-Puffer nach vorhandenen reservierten Wörtern absucht und diese dann in Token umwandelt. Das Maschinenprogramm ruft diese Routine auf und überträgt den Text, also die neue definierte Funktion, in die Zeile 1Ø hinter das Gleichheitszeichen und fügt einen Doppelpunkt, gefolgt von einem REM, an. Alle Vorgänge werden von BASIC durch ein CALL im BASIC-Programm aufgerufen.
Im Programm FUNKTION können auf diese Weise immer wieder neue definierte Funktionen interaktiv eingegeben (Zeile 3Ø) und berechnet (Zeile 5Ø) werden. Eine höhere Qualität wird mit dem Programm FNSA erreicht. Es gestattet, bis zu 26 unterschiedliche Funktionen FNA bis FNZ interaktiv zu definieren und gleichzeitig anzusprechen. Es bietet daher für mathematische Berechnungen einen breiten Anwendungbereich. Im Beispiel-Programm ist nur das Grundgerüst mit fünf verschiedenen Funktionen in den Zeilen 1Ø bis 5Ø dargestellt. Es ist durch Sie zu einem nutzbaren Anwendungsprogramm auszubauen.

c) Verschiebung des Anfangs des BASIC-Programms

Normalerweise befindet sich der Anfang eines BASIC-Programmes auf der Adresse 4Ø1H. Dieser Wert ist im Pointer 35F/6ØH gespeichert. Durch Veränderung des Pointers kann der Beginn des BASIC-Programmes verlagert werden, und somit entsteht ab 4ØØH ein freier Speicherplatz, in den der MC abgelegt werden kann. Der Vorteil dieser Methode liegt vor allem darin, daß der MC nicht direkt im BASIC-Programm liegt und dadurch auch Nullen enthalten kann. Nachteilig ist es dagegen, daß für das Retten beider Teile kompliziertere Methoden notwendig sind. Am einfachsten ist es, den Bereich von 3ØØH bis zum Ende des BASIC-Programms als Ganzes im Betriebssystem zu retten. Für das Ende ist dazu der Pointer 3D7/8H abzufragen.

d) Ablegen am oberen Ende des Speichers

Mit der BASIC-Anweisung CLEAR a,b wird ein Stringraum der Größe a reserviert und die obere Adresse des Speichers b für Zeichenketten festgelegt. Oberhalb von b existiert daher ein freier Speicherbereich, der ebenfalls zur Ablage von MC verwendet werden kann.
Die Methoden c) und d) sind im Programm COPY realisiert. Sein Hauptzweck besteht darin, den Inhalt des Pixelspeichers in den oberen freien Speicherraum zu kopieren. Dadurch kann der dort befindliche Bild-Inhalt auf Kassette gerettet und auch wieder eingelesen werden. Normalerweise ist dies ja durch die Blockanzeige auf dem Bildschirm nicht möglich. Weiter enthält die Maschinenroutine von COPY auch einen Teil, mit dem der Pixelinhalt blitzschnell an den Bildschirm zurückgegeben wird. Durch Anwendung eines Parameters kann der Bildinhalt sogar modifiziert werden. COPY Ø erzeugt den alten Bildinhalt, COPY FF seine Negation. Das Programm selbst, der MC, liegt im Bereich von 4ØØH bis 455H.

Zusammenfassung Einbinden von Maschinenprogrammen

1. Maschinenprogramme können in DATA-Zeilen eines BASIC-Programmes abgelegt werden. Es ist vorteilhaft, wenn der übliche HEX-Code statt Dezimalzahlen verwendet wird. Dazu ist vor dem Poken an die freien Speicherstellen der HEX-Wert in eine Dezimalzahl zu wandeln. Günstig ist es, den HEX-Code halbbyteweise in zwei ASCII-Zeichen zu zerlegen. Über den ASCII-Code erfolgt die Berechnung der Dezimalzahl.
2. Eine fast ideale Methode beruht darauf, den Maschinencode in einer REM-Zeile abzulegen. Hierbei sind folgende Fakten zu beachten

- Um Adreßverschiebungen beim Programmieren in BASIC zu vermeiden, ist die erste Zeile, also Zeile Ø besonders geeignet.
- Das Maschinenprogramm darf keine Nullen enthalten.
- Statt des REM zu Beginn kann auch „!" oder GOTO n stehen.
- Extrem lange Maschinenprogramme können in einer überlangen Zeile abgelegt werden. Dann muß manuell der erste Zeilenpointer richtig gesetzt werden.
- In der Zeile mit dem Maschinencode lassen sich auch gut Copyright und andere Kommentare unterbringen.

Mit dieser Methode lassen sich Programme erzeugen, die gleichermaßen auf allen KC-Rechnern laufen.

3. Bei einem fertigen Programm kommen jedoch fast nur die Vorteile zum Tragen. Deshalb sollten solche Programme so geschrieben sein, daß der Nutzer nicht das Maschinenprogramm merkt.

4. Bei der Entwicklung eines kombinierten BASIC-Maschinen-Programms sollten öfter als sonst üblich, das Programm und Teilroutinen gerettet und bis zur Fertigstellung aufgehoben werden.

5. Für Maschinenprogramme nutzbarer Speicherraum kann auch an folgenden Stellen gewonnen werden:

- Der Anfang von BASIC-Programmen ist verschiebbar. Dies erfolgt durch Verändern eines Pointers.
- Am Speicherende kann Freiraum mittels Pointer oder der Anweisung CLEAR geschaffen werden.
- Hinter dem Speicherraum für die Felder ist meist ein Bereich bis zum BASIC-Stack nutzbar.

Hier eine Zusammenfassung der nutzbaren Speicherbereiche, wobei das BASIC-Programm normal von 400H ab existiert:

Prinzipiell freier Speicherraum

KC 85/2 und 3		KC 85/1 und KC 87
0 ... 140 H		28C ... 2BF sehr bed.
200 ... 2FFH		2C0 ... 2FF falls ohne Erw.
400 ... 3FFFH/7FFFH	1)	400...3FFFH/7FFFH/BFFFH1)
B800...BBFFH sehr bed.	2)	EBC0...EBFFH wenn Farbmod.
BC00 ... BFFFH	2)	

1) nutzbar über:
   a) REM-Zeilen.
   b) CLEAR XX, YYYY.
   c) Programmverschiebung 35F/60.
   d) bedingt hinter dem Array-Bereich.
2) mit beachtlich verlängerter Zugriffszeit.

Sehr bedingt bedeutet, daß hier eventuell aus dem System heraus Probleme auftreten können.

5. Rekursive Programmierung

Die Rekursion ist von fundamentaler Bedeutung für die Informatik. Sie besitzt nicht nur für die Lösung bestimmter mathematischer Probleme große Bedeutung, sondern ist auch für die Entwicklung von Programmiersprachen grundlegend (z. B. LISP oder Prolog). Vereinfacht kann sie als Selbstaufruf mathematischer Strukturen und/oder Funktionen (Algorithmen) verstanden werden. Bildlich gesprochen ist das wie bei Münchhausen, der sich am eigenen Schopf aus dem Sumpf zieht.

Eng mit der Rekursion ist die Iteration verbunden. Sie wird rechentechnisch durch eine FOR-NEXT-Schleife realisiert. Es zeigt sich, daß viele rekursive Probleme iterativ gelöst werden können. Im folgenden sollen einige mathematische Beispiele zur einfachen Erklärung der Rekursivität herangezogen werden.

Eine oft benötigte und sehr einfache Funktion ist die Fakultät. Bekanntlich gilt ja

N! = 1*2*...*N

In BASIC bietet sich hier eine iterative Lösung an:

```
10 INPUT N: X=1
20 FOR I=1 TO N
30 X=X*I
40 NEXT: PRINT X
```

Aus der Zeile 30 folgt aber auch ziemlich unmittelbar der Übergang zur rekursiven Darstellung gemäß:

0! = 1, 1! = 0!*1, 2! = 1!*2 usw.

oder echt rekursiv definiert:

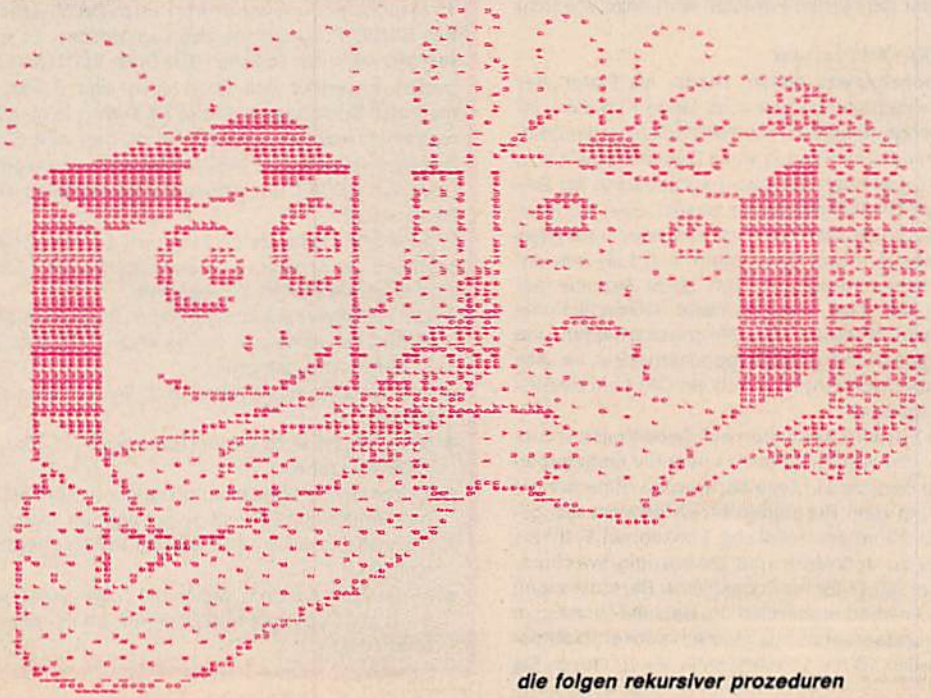
N! = (N−1)!*N

Diese Darstellungsform ist offensichtlich viel übersichtlicher als die obige iterative. Schwieriger zu erkennen ist aber, was sie für die Programmierung bedeutet. Dies wird mit folgendem Schema für 6! deutlicher:

6! = 6 * 5!
= 6 * 5 * 4!
= 6 * 5 * 4 * 3!
= 6 * 5 * 4 * 3 * 2!
= 6 * 5 * 4 * 3 * 2 * 1

Das legt neben der rekursiven Betrachtung wieder eine iterative Berechnung nahe.

Weiter sei noch eine andere rekursiv definierbare Funktion, nämlich der Binomialkoeffizient, betrachtet (s. auch



die folgen rekursiver prozeduren

Kapitel 2). Seine rekursive Definition lautet:

$$\binom{n}{m} = \binom{n}{m-1} \cdot \frac{n-m+1}{n}$$

Genau wie die Fakultät kann aber auch der Binomialkoeffizient (er tritt z. B. beim Pascalschen Dreieck auf) in seiner rekursiven Definition iterativ berechnet werden (Programm BINOM).

Ein drittes Beispiel sind die Fibonacci-Zahlen. Fibonacci hatte sich theoretisch mit der Vermehrung von Kaninchen befaßt. Er setzte voraus, daß ein Pärchen zwei Junge (ein männliches und ein weibliches) zur Welt bringen. Das junge Pärchen sei nach einer Generation und für die Dauer von zwei Jahren geschlechtsreif. Die Frage ist nun, wieviel Pärchen gibt es nach N Jahren?

Beginnen wir mit einem Pärchen, so ist die Fibonacci-Zahl von eins F(1) = 1, nach einem Jahr sind es zwei Pärchen, also ist F(2) = 2 und nach N Jahren sind zwei Generationen vorhanden, also gilt:

$$F(N) = F(N-1)+F(N-2)$$

Wir haben also eine allgemeinere rekursive Definition erhalten. Programmtechnisch ist daher F(N) etwas komplizierter zu realisieren als bei den ersten Beispielen. Eine relativ übersichtliche iterative Lösung stellt das Programm FIBON dar.

Den Programmen BINOM und FIBON ist gemeinsam, daß bei der iterativen Berechnung der rekursiven Funktionen die Anzahl der benötigten Schritte in der FOR-NEXT-Schleife vorher bekannt ist. Das ist nicht bei allen rekursiven Problemen der Fall. Ein Beispiel hierfür ist der größte gemeinsame Teiler (ggt).

Ein Algorithmus hierfür stammt von Euklid. Wir wollen beispielsweise den ggT der beiden Zahlen 315 und 585 ermitteln. Dazu dividieren wir 585/315 und bestimmen den Rest R1 (R1 = 270). Nun wird die kleinere der beiden Zahlen (315) durch R1 geteilt und wieder die kleinere der beiden Zahlen (315) durch R1 geteilt und wiederum der Rest bestimmt (R2 = 45). Schließlich ergibt 270/45 den Rest R3 = 0. Dann ist R2 = 45 der ggT der Ausgangszahlen 585 und 315.

Dieser Algorithmus kann also rekursiv definiert werden, indem man immer wieder zwei Zahlen dividiert und den Rest bestimmt. Wird kein Rest erhalten, ist der ggT gefunden.

Da wir nicht wissen, wie oft der Algorithmus wiederholt werden muß, verwendet man bei der iterativen Berechnung solcher rekursivern Probleme am günstigsten eine FOR-NEXT-Schleife mit der Schrittweite Null (zur unendlichen FOR-NEXT-Schleife siehe auch URANIA-Extra S. 28). Ist für den Fall des ggT der Rest gleich Null, kann beispielsweise mit einem GOTO-Befehl aus der Schleife herausgesprungen werden (Programm GGT).

Eine qualitativ neue Problematik, die in der Mathematik eine große Rolle spielt, bietet die Rekursionstheorie. Sie steht in engem Zusammenhang mit der allgemeinen Algorithmentheorie. Sie macht Aussagen darüber, was berechenbar ist. Hierbei werden nicht nur einzelne Funktionen, sondern ganze Funktionsklassen definiert, um alles erfassen zu können. Es bietet sich dabei ein Vergleich zur Rekursivität der menschlichen Sprache an. Wir gehen bei ihr von bestimmten Grundelementen und Regeln der Sprache aus, und mit diesen beiden Gegebenheiten ist die Sprache unendlich ausdrucksfähig. Ähnlich geht die Algorithmentheorie vor. Sie definiert wenige Regeln und einen Bestand von Grundfunktionen, aus denen dann alle berechenbaren Funktionen abgeleitet werden.

In den 40er Jahren unseres Jahrhunderts erreichte diese Problematik einen gewissen Höhepunkt, der durch die Aufstellung der nach dem Mathematiker Ackermann benannten Funktion bewirkt wurde. Die Ackermann-Funktion hat nämlich eine Rekursivität von größerer Allgemeinheit als alle zuvor bekannten Funktionen. Für sie gilt:

$$A(0,Y) = Y+1$$
$$A(X,0) = A(X-1,1)$$
$$A(X,Y) = A(X-1,A(X,Y-1)).$$

Das Besondere bei der Ackermann-Funktion ist, daß als Argument wiederum eine Ackermann-Funktion auftritt. Das Verfahren wird Ihnen verständlicher, wenn Sie beispielsweise einmal per Hand den Wert für A(2,2) berechnen.

$$A(2,2) = A(1,A(2,1)) = A(1,A(1,A(2,0))) = A(1,A(1,A(1,1))) = \ldots$$

Zur Berechnung von ausgewählten Werten der Ackermann-Funktion sollen hier zwei qualitativ unterschiedliche Methoden betrachtet werden:

a) Ablage der Argumente auf einen Stack

Die verschachtelten Zahlenwerte werden bei einem Rechenprogramm günstig auf einem Stack abgelegt. Hierauf beruht das erste der zwei unterschiedlichen BASIC-Programme.

Im Verlauf der Rechnung ruft die Ackermann-Funktion sich wiederholt und mehrfach auf. Die Argumente, die die einzelnen Ackermann-Funktionen in sich aufnehmen, müssen wir zu diesem Zweck speichern. Dies kann am einfachsten durch die Ablage der Werte auf einem Stack erfolgen. Diese Methode verwendet das Progamm ACK2. Um den Überblick über den Ablauf bei den Berechnungen zu behalten, wird in ACK2 nach jeder neuen Rekursion der vollständige Stack angezeigt. Dieser ziemlich komplizierte Rechenprozeß ähnelt dem Sortieralgorithmus für Quick-Sort (siehe Thema 1).

b) Ablage der Argumente in einzelnen Feldelementen

Die äußere Form der Ackermann-Funktion ähnelt in BASIC einem zweidimensionalen Feld, und in den einzelnen Feldelementen können die Werte der Ackermann-Funktion abgelegt werden. Dazu ist auf Grund der rekursiven Definition der Ackermann-Funktion lediglich ein Umspeichern zwischen einzelnen Feldelementen nötig. Die Akkermann-Funktion kann somit iterativ berechnet und der Weg über den Stack umgangen werden (Programm ACK1). Hierbei tritt jedoch ein neues Problem auf, dem besondere Aufmerksamkeit geschenkt werden muß. Wir müssen nämlich wissen, wie weit wir auf das Feld zurückgreifen können, d. h., wie groß das Feld zu definieren ist. Da das von vornherein nicht bestimmt werden kann, ist es sinnvoll, wie folgt vorzugehen:

Wir definieren ein sehr großes Feld und achten darauf, daß nicht auf Feldelemente zurückgegriffen wird, die nicht definiert sind (Zeile 80). Wenn dies realisiert ist, so kann die Ackermann-Funktion gemäß ihrer Definition im Programm direkt abgearbeitet werden (Zeile 100). Das ist nur ein Vorteil dieser Berechnungsmethode. Ein weiterer besteht darin, daß mit dem Programm ACK1 die Ackermann-Funktion für viele Argumente (abhängig vom Rechnertyp) zusammenhängend berechnet wird. Zusätzlich wird der Organisationsaufwand für die Berechnungen im Gegensatz zur rekursiven Methode über den Stack wesentlich verringert. Dies schlägt sich natürlich in einer viel geringeren Rechenzeit des Programms ACK1 nieder (siehe Tabelle).

Hier zeigt sich ein Problem vieler rekursiver Sprachen. Sie benötigen für die Stackarbeit sehr viel Zeit. Daher sollten rekursive Probleme möglichst iterativ gelöst werden. Doch leider existieren solche Methoden nicht für alle rekursiven Probleme. Andererseits ist aber die rekursive Darstellung im Gegensatz zur iterativen übersichtlicher.

Zusammenfassung Rekursive Programmierung

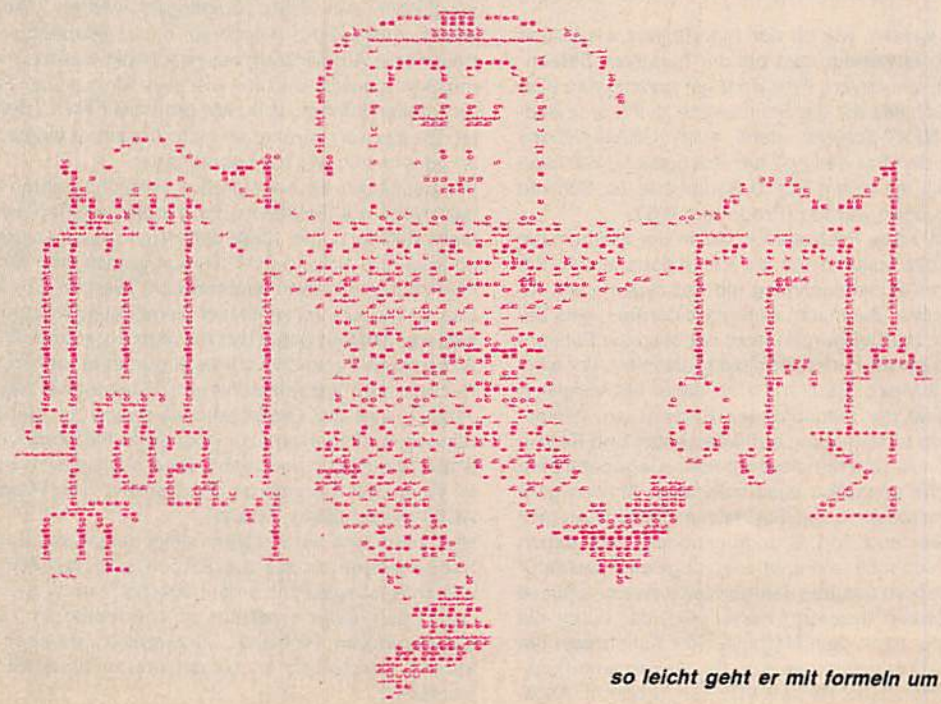
1. Für viele Funktionen ist es vorteilhaft, sie rekursiv zu definieren. Dabei steht sowohl rechts als auch links vom Gleichheitszeichen die zu definierende Funktion. Sie zitiert sich also selbst. Auf diese Weise kann Wesentliches über die Funktion besonders deutlich hervorgehoben werden. Beispiele sind die Fakultät, der Binomialkoeffizient und die Fibonacci-Zahlen.
2. Die Programmierung von Funktionen, die rekursiv definiert sind, ist meist mittels FOR-NEXT-Schleifen, also iterativ möglich.
3. Bei der iterativen Berechnung von rekursiv definierten Funktionen können zwei Fälle unterschieden werden:
   - Die Anzahl der Iterationszyklen ist zu Beginn bekannt. Dann genügt die einfache FOR-NEXT-Schleife.
   - Die Anzahl der Iterationszyklen ist – wie beim euklidischen Algorithmus für den größten gemeinsamen Teiler – zu Beginn nicht bekannt. Dann existiert aber eine Ende-Bedingung. Dies entspricht der WHILE-WEND-Konstruktion anderer Sprachen. Sie wird in BASIC durch die FOR-NEXT-Schleife mit der Schrittweite Null realisiert.
4. Mittels Rekursion und ergänzender Aussagen lassen sich nicht nur einzelne Funktionen, sondern ganze Klassen von Funktionen definieren. Dadurch ist es möglich, alles was berechenbar ist, zu erfassen.
5. Eine Funktion, die in diesem Kontext besondere Bedeutung hat, ist die Ackermann-Funktion:
   $A(0, y) = y + 1$
   $A(x, 0) = A(x - 1, 1)$
   $A(x, y) = A(x - 1, A(x, y - 1))$
   Sie gehört zur Klasse der allgemein rekursiven Funktionen. Zu ihrer Berechnung ist daher eigentlich eine rekursive Programmiersprache notwendig. In BASIC ist dies mittels eines nachgebildeten Stacks möglich. Dies leistet das Programm ACK2.
6. Durch einen Trick ist ein Weg mittels Arrays zu realisieren. Er läßt sogar die Struktur der Ackermann-Funktion deutlich hervortreten. Darüber hinaus ist das Programm ACK1 sogar extrem viel schneller als das eigentlich rekursive Programm. Es verlangt erheblich weniger organisatorischen Aufwand.
7. Rekursives Programmieren nutzt die Vorteile der rekursiven Definitionen, erfordert aber meist lange Abarbeitungszeiten, die infolge der umfangreichen Stackarbeit entstehen. Deshalb sind iterative Verfahren oft vorteilhaft. Sie sind jedoch nicht für alle rekursiven Probleme nutzbar. Dies gilt insbesondere für rekursive Prozeduren und Strukturen.

6. Analytische Methoden, Formelmanipulation

Die Verbindung von Rechentechnik und mathematischen Methoden ist sehr eng und heute nicht mehr wegzudenken. Nicht ganz so einfach ist allerdings die Frage zu beantworten, inwiefern mittels der Rechentechnik mathematische Beweise erbracht werden können. Das ist insbesondere dann interessant, wenn die klassische Mathematik versagt. Dafür gibt es aber nur sehr wenige Beispiele. Eines davon ist das Vierfarbenproblem.

Hierbei geht es darum, auf einer politischen Landkarte alle Länder so mit Farben zu kennzeichnen, daß sich nicht zwei Länder gleicher Farbe berühren. Die spezifische Frage lautet: Genügen immer vier Farben? Für das Beispiel Luxemburg mit seinen drei Nachbarn BRD, Belgien und Frankreich ist dies sicherlich erfüllt. Hat ein Land dagegen vier Nachbarn, so kommt man zur Kennzeichnung der fünf Länder erstaunlicherweise sogar mit nur drei Farben aus.

Es ist nun eine Vermutung, daß man generell für politische Landkarten mit nur vier Farben auskommt. Für diese Hypothese existiert aber kein Beweis im Sinne der



so leicht geht er mit formeln um

klassischen Mathematik. Ein rechentechnischer Beweis konnte jedoch im Jahre 1976 von den Mathematikern Wolfgang Haken und Kenneth Apple (beide USA) erbracht werden. Es gelang ihnen, daß allgemeine Vierfarbenproblem mit den Methoden der klassischen Mathematik auf ungefähr 2000 Fälle zu reduzieren und diese Fälle mittels Computer zu durchmustern. Dazu brauchte der Rechner rund 1000 Stunden.

Betrachten wir ein anderes – das HYDRA-Problem (URANIA-Extra S. 23 und S. 30). Es wurde etwa 1960 von McCarthy aufgestellt. Ausgangspunkt ist eine beliebige natürliche Zahl. Ist sie gerade, so wird die Zahl durch Zwei dividiert. Ist sie dagegen ungerade, dann wird mit Drei multipliziert und Eins addiert. Dieser Algorithmus wird solange wiederholt, bis als Ergebnis die Zahl Eins entsteht. Es gibt die Vermutung, daß das HYDRA-Problem für alle Startwerte zur Eins führt. Ein mathematischer Beweis dafür existiert jedoch bis heute nicht. Computerrechnungen bestätigen diese Vermutung, wenn auch die Anzahl der auf Rechnern dargestellten Zahlen unvergleichbar klein in bezug auf die Menge der natürlichen Zahlen ist.

Es zeigt sich also, daß die Rechentechnik ein hilfreiches Mittel für die klassische Mathematik sein kann. Mit ihr können mathematische Beweise erbracht oder Hypothesen bis zu einem gewissen Grade unterstützt werden.

Im folgenden soll eine interessante Aufgabe rechentechnisch realisiert werden, die allerdings in der Mathematik schon bewiesen ist. Sie sagt aus, daß jede natürliche Zahl durch die Summe von maximal vier Quadratzahlen darstellbar ist. Beispielsweise ist $18 = 4*4 + 1*1 + 1*1$ oder $18 = 3*3 + 3*3$ und schließlich ist $18 = 3*3 + 2*2 + 2*2 + 1*1$. Mit einem Programm sollen nun die verschiedenen Kombinationen von Quadratsummen gefunden werden. Das leistet das Programm VIQUA. Aus der Analyse des Programms kann man Verständnis dafür gewinnen, wie kombinatorische Probleme programmiert werden und wie man auch mit BASIC mathematische Beweise rechentechnisch unterstützen kann.

In Naturwissenschaft und Technik wird oft die Mathematik, speziell die analytische Mathematik angewendet. Hierbei muß meist mit recht komplizierten Formeln operiert werden. Man hat eine Problemstellung, übersetzt sie in Formeln und kommt zu neuen komplizierten Ausdrücken. Diese sind z. T. so komplex, daß sie vereinfacht werden müssen. Diese Vereinfachung war bislang dem Menschen vorbehalten. Seit einigen Jahren gibt es aber Computerprogramme zur Formelmanipulation. Sie liefen zunächst nur auf Großrechnern, dann auch auf Personalcomputern. Neuerdings stehen erste Programme auch für Kleinstrechner zur Verfügung.

Im folgenden sollen drei derartige kleine Programme beschrieben werden:

a) Programm POT

In der Kleinstrechnertechnik ist es nicht vorteilhaft, die Potenzfunktion zu verwenden. Hierbei treten relativ lange Rechenzeiten und Rundungsfehler auf, da die Potenzfunktion im Rechner über Logarithmus und Exponentialfunktion bestimmt wird. Dies kann umgangen werden, wenn z. B. in einem Programm aus $Y = X^3$ $Y = X*X*X$ erzeugt wird. Genau solche Umformungen sind mit dem Programm POT möglich. (Der rechentechnische Nutzen für derart triviale Fälle soll hier nicht zur Diskussion stehen, es geht vielmehr um das Prinzip.) Zusätzlich läßt das Programm noch gewisse Klammerungen und Faktoren in den umzuformenden Ausdrücken zu. Zur Realisierung derartiger „Formelmanipulationen" werden im Programm POT (wie auch in den weiteren Programmen) Stringoperationen angewendet.

Eine höhere Stufe dieses Prinzips bietet das:

b) Programm DIFER

Oft werden bei mathematischen Berechnungen nicht nur bestimmte Formeln sondern auch ihre Ableitungen benötigt. Hierzu kann das Programm DIFER genutzt werden. In ihm wird interaktiv eine Funktion eingegeben (Zeile 550). Dies Funktion kann sich in recht komplexer Weise aus den vom BASIC-Interpreter bereitgestellten Grundfunktionen zusammensetzen. Die Berechnung der Ableitung erfolgt unter Berücksichtigung von Ketten-, Produkt- und Quotientenregeln. Es wurde versucht, das Programm relativ übersichtlich und nachvollziehbar zu schreiben. Der Preis dafür ist aber, daß in den abgeleiteten Funktionen viele Klammerebenen und Faktoren stehen, die man bei einer Ableitung „per Hand" nicht verwenden würde. Dadurch bleibt das Programm gut lesbar. Die relativ umständliche Form der Ableitung einer Funktion wird vermieden bei dem dafür aber auch wesentlich komplexeren:

c) Programm ABLEI

Das Programm liefert ebenfalls die Ableitung einer Funktion, die interaktiv eingegeben werden kann. Im Vergleich zum Programm DIFER ist jedoch die Anwendung komfortabler, denn unter Ausnutzung einer eingebauten Maschinenroutine (in Zeile 0) wird die eingegebene Funktion in einer definierten Funktion (Zeile 2) abgelegt. Danach wird die Funktion abgeleitet, vereinfacht und die Ableitung wiederum in einer definierten Funktion (Zeile 30) abgelegt. Somit stehen sowohl die Ausgangs- als auch die abgeleitete Funktion für weitere Anwendungen zur freien Verfügung. Es können Tabellen erstellt oder Nullstellen bestimmt werden u. a. m.

Natürlich kann jeder Anwender das Programm nach seinen Bedürfnissen verbessern oder anpassen. So lassen sich relativ leicht mehrfache Ableitungen realisieren. Vielleicht ist es auch Grundlage dafür, ähnliche Programme, z. B. für die Integration, zu entwickeln. Es wäre schon nützlich, wenigstens die Grundintegrale zu erzeugen.

Zusammenfassung Analytische Methoden, Formelmanipulation

1. Das Vierfarbenproblem kann beispielhaft an einer politischen Landkarte betrachtet werden. Die Frage besteht dann darin, ob es bei beliebiger Anordnung der Länder immer möglich ist, mit 4 Farben auszukommen, ohne daß sich gleiche Farben an einer Grenze berühren.
2. Hierbei konnte gezeigt werden, wie die Rechentechnik auch bei mathematischen Beweisen wirksam werden kann. Sie ermöglicht nämlich eine weitaus größere Anzahl von Fällen zu durchmustern. Die Anzahl dieser Fälle muß aber endlich sein.
3. Wird die Anzahl der Fälle extrem groß, so übersteigt die Rechenzeit alle Grenzen und das Problem ist damit nicht mehr mittels der Rechentechnik lösbar.
4. Werden die Zahlenwerte sehr groß, oder müssen gar unendlich große Zahlen, wie beim Hydraproblem zugelassen werden, so kann ebenfalls nur ein kleiner Teil der Fälle durchmustert werden. Dennoch hilft in diesen Fällen die Rechentechnik etwas in dem Sinne, daß sie die Vermutung weiter unterstützt. Der eigentliche Beweis muß jedoch mit den bekannten mathematischen Mitteln erbracht werden.

5. Seit einigen Jahren sind Programmpakete bekannt, mit denen man formal wie in der Analytischen Mathematik arbeiten kann. Es können also Formeln vereinfacht, abgeleitet oder integriert werden und vieles andere mehr. Es stehen schon jetzt für Kleinstrechner einfache Methoden bereit. Damit gelangt die Rechentechnik auf ein neues Niveau, nämlich von der Rechnung mit Zahlenwerten zum Umgang mit Formeln.

7. Lernfähige Programme, Künstliche Intelligenz

Die Arbeiten zur „Künstlichen Intelligenz" (KI) verfolgen gegenwärtig zwei Hauptziele:

- Es geht darum, neue Rechnerstrukturen zu finden, die eine höhere Leistung als die z. Z. existierenden erreichen. Beispiel hierfür ist die Entwicklung von Parallelrechnern.
- Mittels der Rechentechnik wird versucht, menschliches Verhalten zu simulieren. Dabei muß das gewonnene Ergebnis aber durchaus nicht mit „menschlichen" Methoden oder daraus abgeleiteten Algorithmen übereinstimmen.

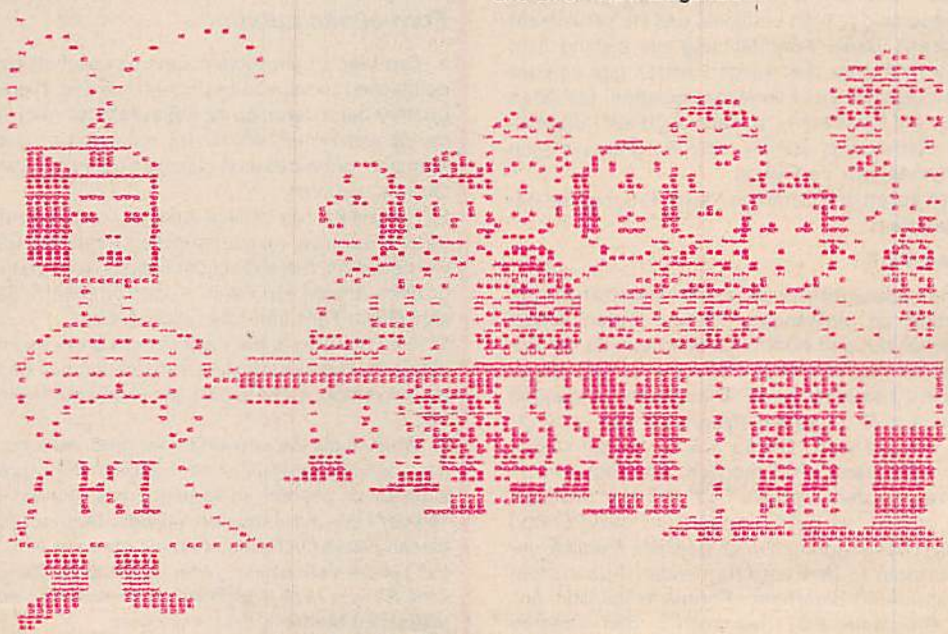
Für die zweite Aufgabe der KI sind u. a. „intelligente" Spielprogramme ein recht bedeutsames Arbeitsgebiet, kommt es doch darauf an, durch leistungsfähige Programme den Rechner zu einem starken Spielgegner zu machen. Besonderes Interesse wird bei vielen Untersuchungen natürlich dem Schach entgegengebracht, und es ist erstaunlich, welch hohen Stand die internationalen Computer-Schach-Wettbewerbe ausweisen. Auch käufliche Schachcomputer oder Programme wie CHESSMASTER demonstrieren einen beachtlichen Stand. Da Schachprogramme sehr umfangreich sind, kann an dieser Stelle nicht weiter auf sie eingegangen werden.
Anliegen dieses Kapitels ist es, an relativ einfachen Beispielen einige Grundmethoden lernfähiger Programme vorzustellen, welche für viele Aufgaben der KI, natürlich in weit komplizierterer Form, ebenfalls von Bedeutung sind.

a) Das Wassserglasproblem

Es besteht die Aufgabe, mit drei Gläsern unterschiedlichen Volumens (z. B. 50, 70 und 120 ccm) eine bestimmte Menge Wasser (z. B. 90 ccm) abzufüllen. Wie geht der Mensch dabei vor und wie kann diese Aufgabe mit einem Programm gelöst werden?
Der Mensch überlegt folgendermaßen:
Wir füllen das Glas mit 70 ccm und gießen das Wasser in das größte Glas. Die noch benötigten 20 ccm sind aber gerade die Differenz der beiden kleineren Gläser. Also füllen wir erneut das Glas mit 70 ccm und gießen 50 ccm in das kleine Glas. Der Rest (20 ccm) kommt in das große Glas. Die Aufgabe ist gelöst.
Der Algorithmus für ein entsprechendes Programm wird meist anders sein, denn selbst in derart übersichtlichen Fällen probiert der Rechner alle möglichen Kombinationen durch. Dieser Weg ist im Programm WAGLA realisiert. Dabei können eine beliebige Zielmenge (Zeile 30) und drei verschiedene Werte für die Glasvolumina (Zeilen 40 bis 60) eingegeben werden. In den Zeilen 70 bis 130 wird dann getestet, welche Kombinationen der drei Gläser die gewünschte Zielmenge ergeben. (Man beachte auch hierbei die große Bedeutung der IF-THEN-Anweisung!). Man sieht also, daß das Wasserglasproblem vom Menschen und vom Rechner unterschiedlich gelöst wird. Genau hieran zeigt sich die Problematik der KI, Programme zu schaffen, die vor allem im Ergebnis dem menschlichen Denken äquivalent sind.
Schon etwas komplizierter wird die Aufgabe bei dem Spiel „Türme von Hanoi", das im Zusammenhang mit der KI oft erwähnt wird. Dieses Problem wurde im Programm HANOI (URANIA-Extra S. 57) mit einer Modulo-Arithmetik gelöst und soll hier nicht nochmals betrachtet werden.

b) Das Damenproblem

Auf einem Dame-(Schach-)Brett sollen acht Damen so aufgestellt werden, daß sie sich nicht gegenseitig bedrohen. Dazu stellt man beispielsweise die erste Dame auf ein Eckfeld. Damit sind die entsprechende waagerechte und senkrechte Linie sowie eine Hauptdiagonale für weitere Damen gesperrt. Die zweite Dame wird auf ein erlaubtes Feld gesetzt und blockiert wiederum zwei Linien und zwei Nebendiagonalen.



kampf gegen die kuenstliche intelligenz

Fortschreitend bleiben immer weniger erlaubter Felder für die Damen. Das macht das Problem so kompliziert, und in der Regel ist das Ziel auf diesem Weg nicht zu erreichen. Man muß gewisse Züge zurücknehmen und neu setzen können. Diesen Vorgang nennt man in der KI Backtracking. Das Backtracking muß also programmiert werden.
Das Damenproblem ist im Programm DAMEN realisiert, jedoch nicht nur für ein Schachbrett (8 × 8), sondern für ein beliebiges quadratisches Feld n*n und die entsprechende Anzahl n (4 bis 19) Damen (Zeile 20).
Außerdem werden alle verschiedenen möglichen Damenanordnungen bestimmt.
Da beim Backtracking im voraus die Anzahl notwendiger Operationen nicht bekannt ist, bietet sich die unendliche FOR-NEXT-Schleife mit der Schrittweite Null an (Zeile 40). Um falsche Züge auch wieder zurücknehmen zu können, werden alle Züge auf einem Stack gespeichert (Zeilen 90, 100).
Das Backtracking kann anschaulich am Beispiel eines Labyrinths dargestellt werden. Es gibt die Erzählung vom Ariadne-Faden: Man nehme ein Wollknäuel, befestige den Faden am Eingang und wandere in das Labyrinth hinein. Wenn es nicht weitergeht, befestigt man den Faden dort und wandert mit dem Wollknäuel zurück, bis ein neuer Abzweig kommt. So findet man den Weg durch das Labyrinth. Er verläuft schließlich dort, wo nur ein Faden liegt. Ein doppelter Ariadne-Faden kennzeichnet einen Irrweg.
Das Backtracking im Rechner ist ähnlich. Im Programm müssen alle Wege (Züge beim Damenproblem) berechnet und doppelte Züge gekennzeichnet werden. Genau dies geschieht durch Ablage auf dem Stack. Wenn ein Zug zurückgenommen werden muß, wird der Wert wieder vom Stack heruntergeholt.

c) Solitude (Solohalma)

Das Brett für dieses beliebte Spiel ist im Bild (Programmteil) dargestellt. Alle Felder außer dem mittleren sind am Anfang mit Spielsteinen besetzt (man kann sich das Spiel mit recht einfachen Mitteln auch selbst bauen). Ähnlich wie bei Dame müssen durch Überspringen Spielsteine entfernt werden. Ziel ist es, daß am Ende nur noch ein Stein (möglichst im mittleren Feld) übrigbleibt. Bei einem Spiel wird das wegen der hohen Schwierigkeit aber kaum gelingen. Zwei oder drei Reststeine kennzeichnen schon einen Meister! Wegen der großen Zahl möglicher Zugkombinationen wird auch ein Rechnerprogramm für das Spiel recht kompliziert sein.
Solohalma kann mit dem Programm SOLET gespielt werden. Im Programm können die Steine gesetzt werden, wobei gleichzeitig geprüft wird, ob das Setzen möglich ist. Außerdem wird jeder Zug, wie beim Backtracking, auf einem Stack gespeichert. Die Möglichkeit, Züge zurücknehmen und wiederholen zu können, vergrößert die Übersichtlichkeit während des Spielablaufs wesentlich.
Das Programm SOLET ist gleichermaßen auf den KC 85/2-4 und dem KC 85/1 bzw. KC 87 lauffähig. Wegen der unterschiedlichen Grafik beider Typen mußten allerdings programmtechnische Kompromisse eingegangen werden, die die Übersichtlichkeit des Programms verschlechtern.

d) Lernfähiges Nimm-Spiel

Als Kind hat wohl jeder schon Nimm gespielt. Man hat 35 Hölzer (oder Spielsteine), und zwei Spieler müssen abwechselnd ein bis fünf Hölzer wegnehmen. Wer das letzte nehmen muß, hat verloren.
Dieses Spiel wurde schon vielfach programmiert. Hier wird eine spezielle, nämlich lernfähige Version vorgestellt (Programm LNIMM). Anfänglich ist das Programm „dumm". Sie spielen gegen den Rechner, der nur per Zufall (Zeile 170) seine Anzahl von Hölzern berechnet. Es ist daher leicht, in dieser Phase des Programms gegen den Rechner zu gewinnen. Aber: Nach jedem Spiel speichert der Rechner, wer gewonnen hat und wieviele Hölzer jeweils genommen wurden. Dies geschieht in einer Matrix (35,5), also vorhandene mal genommene Hölzchenzahl.
Zunächst werden die einzelnen Feldelemente von A mit dem Wert 128 belegt (Zeile 40). An jedem Spielende werden die betreffenden Werte erhöht (wenn der Rechner gewonnen hat) oder erniedrigt (wenn der Rechner verloren hat). Dadurch bildet das Programm in der Matrix ab, wieviel Hölzer in jeder Situation genommen werden müssen. Mit jedem neuen Spiel wird diese Matrix verbessert, und es wird Ihnen nach etwa einhundert Spielen, wenn Sie auch nur einen Fehler machen, kaum gelingen, gegen den Rechner zu gewinnen.
Um aber nicht erst einhundert Spiele gegen den Rechner machen zu müssen, um in ihm einen starken Partner zu haben, ist im Programm eine Routine eingebaut, die den Rechner gegen sich selbst spielen läßt und so automatisch die Lernmatrix aufbaut.
LNIMM ist ein Beispiel dafür, wie der Rechner beim Spiel aus Erfahrung lernt und dies für weitere Handlungen nutzt. Und genau das ist eine der Aufgaben, die bei vielen Problemen der KI gestellt wird. Zugegeben, LNIMM ist ein relativ einfaches Spielprogramm.
Weit komplizierter ist Schach. Schach ist im Prinzip ein deterministisches Spiel, d. h., man könnte alle möglichen Züge in gegebener Spielstellung berechnen. Nur ist dann das Programm so umfangreich, daß es selbst auf den größten Rechnern nicht lauffähig wäre (Speicher- und Rechenzeitprobleme). Andererseits gestattet die Vielzahl möglicher Kombinationen nicht, ein Schachprogramm lernfähig zu gestalten. Deshalb werden bei derartigen Spielprogrammen (Schach, Mühle usw.) Positionen und Spielstärken für die Figuren eingeführt und hieraus numerische Bewertungen für die Züge abgeleitet.
Die Entwicklung von Schachprogrammen ist mittlerweile so weit, daß ein normaler Spieler kaum gegen gute Schachcomputer gewinnen kann, ein Großmeister jedoch echte Chancen hat. Anders ist es dagegen bei dem komplizierten Spiel Go. Für Go gibt es z. Z. kein Programm, das über die Leistung eines mittleren Spielers hinauskommt.
Spiele und „Künstliche Intelligenz" bringen sich also gegenseitig voran. Wir hoffen, daß wir Ihnen mit diesem Kapitel einige Grundprobleme der KI deutlich machen konnten. Die Analyse der Programme wird Ihnen dabei nützlich sein.

Zusammenfassung
Lernfähige Programme

1. Die Künstliche Intelligenz läßt sich als ein wissenschaftliches Arbeitsgebiet beschreiben, das heute vor allem zwei Ziele verfolgt:
   - Die Entwicklung neuer hochleistungsfähiger Rechner
   - Simulation von intelligentem menschlichen Verhalten mittels der Methoden der Rechentechnik.

   Im zweiten Fall ist dabei nicht entscheidend oder gefordert, daß gleiche oder ähnliche Ergebnisse auch gleiche Algorithmen oder Denkprinzipien bedeuten.
2. Einige Probleme lassen sich über die Durchmusterung aller möglichen Kombinationen lösen. Hierzu gehört z. B. das Wasserglasproblem. Doch so etwas ist die Ausnahme.

3. Eine wichtige Methode der Künstlichen Intelligenz ist das Backtracking. Hierbei werden alle vorangegangenen Züge auf einen Stack gespeichert. Falls man sich geirrt hat, so ist der gegangene Weg gespeichert, und es können die Züge teilweise zurückgenommen und durch andere ersetzt werden. Eine solche Methode ist beim Labyinth am Beispiel des Ariadne-Fadens bekannt. Das Damenproblem gehört ebenfalls zu dieser Problemklasse.
4. In einigen Fällen ist es möglich, Spiele lernfähig zu gestalten. In einer Matrix führen dabei die Erfolgszüge zu höherer und die Verlustzüge zu niedrigerer Bewertung. Die Zugwahl erfolgt dann neben dem Zufall durch die Größe dieser Koeffizienten. So kann mit einem an sich leistungsschwachen Programm begonnen werden, welches aber über die Änderung dieser Koeffizienten ständig seine Leistungsfähigkeit steigert.
5. Für komplizierte Probleme kann man weder eine solche Matrix hinreichend groß machen, noch kann man durch Kombinatorik die besten Züge ausrechnen. Dann sind komplizierte, heuristische Algorithmen erforderlich. Sie werden u. a. bei Schachprogrammen angewendet. Unter anderem werden dabei je nach Spielsituation und Figurenstärke die Entscheidungen bezüglich der Zugwahl verändert.

8. Menu- und Eingabetechniken

Zwischen dem Menschen und dem Rechner ist Kommunikation notwendig. Wir führen Eingaben durch und erhalten Ausgaben und Antworten am Bildschirm. Dieses Wechselspiel (Mensch-Maschine-Dialog) wird durch Programme gestaltet. Die entsprechende Programmierung muß in der Regel so durchgeführt werden, daß der Dialog mit dem Rechner auch für Nichtfachleute sinnvoll und effektiv verläuft. Hierbei hat sich in den letzten Jahren eine beachtliche Entwicklung vollzogen.
Der ursprüngliche, sehr spartanische Dialog wird heute überwiegend übersichtlich und leicht verständlich abgewickelt. Zu der standardgemäßen Eingabe über die Tastatur kommen immer neue Möglichkeiten hinzu, so z. B. die Eingabe mit Maus, Digitalisierungstablett, Joystick, Lichtstift und in Ansätzen auch schon die Spracheingabe.
Die Ausgabe über den Bildschirm steht beim Dialog aber nach wie vor an erster Stelle. Andere Techniken haben hier kaum Bedeutung erlangt. Was auf dem Bildschirm erscheint, wird oft als Bedieneroberfläche bezeichnet. Sie gilt es effektiv zu gestalten, d. h., sinnvoll zu programmieren. Einige Regeln, die dabei beachtet werden sollten, enthalten die Punkte 1 bis 3 der Zusammenfassung.
Im folgenden werden zwei verschiedene Eingabetechniken für unsere Kleincomputer vorgestellt:

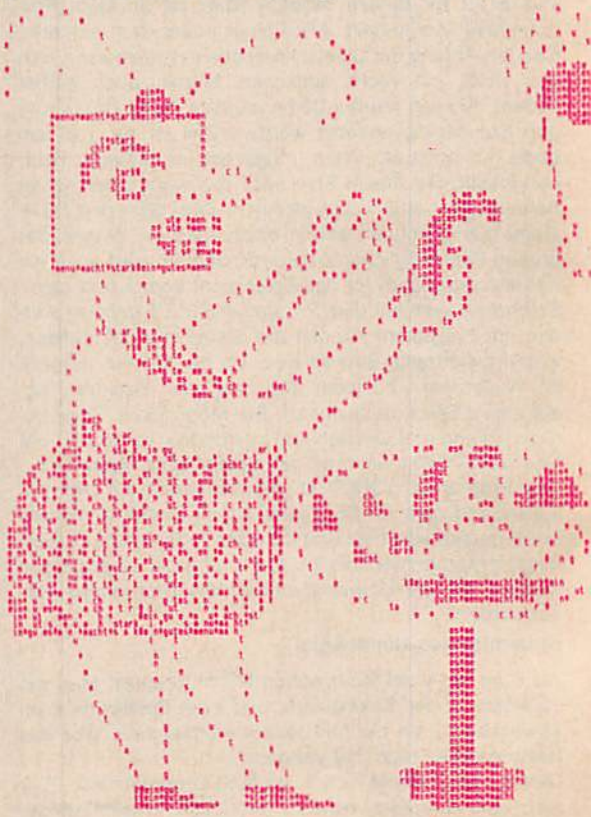
a) Eingabe von Zeichen

Angenehme Bedienarbeit erfordert u. a. eine übersichtliche Menugestaltung. Ihre Entwicklung geschah schrittweise. Lange erfolgte einfache Tasteneingabe. Dann geschah die Menu-Auswahl über die Eingabe von Zahlen. Ein solches Menu könnte z. B. folgendes Aussehen haben:
INPUT„1 = Eingabe, 2 = Rechnen, 3 = Ausgabe ...", A
Hier soll eine ähnliche Technik vorgestellt werden, die allerdings mnemotechnische Begriffe verwendet. Das trivialste wäre, das obige Beispiel in der Form:
INPUT „E = Eingabe, R = Rechnen, A = Ausgabe ...";
AS
zu schreiben. Mnemotechnisch günstiger ist es, eine bestimmte Anzahl von Zeichen für die Menuauswahl zuzulassen. Dies ist im Programm EINGABE realisiert. Mit dem Programm können drei Operationen durchgeführt werden:

- Eingabe von ASCII-Zeichen, Umwandlung in ihre entsprechenden Hex- und Dezimalwerte.
- Eingabe von Dezimalwerten, Umwandlung in ASCII-Zeichen, Ausgabe der ASCII-Zeichen
- Eingabe von HEX-Werten, Umwandlung in ASCII-Zeichen, Ausgabe der ASCII-Zeichen

Auf dem Bildschirm erscheint dazu das Menu in der Form: ASCII-Zeichen
Dez-Werte
Hex-Werte

Am Anfang des Programms wird abgefragt, wieviel gültige Zeichen für die Menuwahl existieren sollen (Zeile 30). Legen Sie z. B. im Programm EINGABE zwei gültige Zeichen fest und wollen HEX-Werte in ASCII-Zeichen umwandeln (Operation 3), so genügt bei der Menuwahl die Eingabe der beiden Buchstaben HE; es sind aber auch HEX, HEX-W usw. möglich. Bei einer anderen Anzahl von gültigen Zeichen bestehen auch weitere Eingabemöglichkeiten. Sie sollten mit dem Programm verschiedene Beispiele durchprobieren.
Das Programm EINGABE hat jedoch einen kritischen Punkt, und zwar ist es (bewußt) nicht absolut absturzsicher. Die Wahrscheinlichkeit des Programmabsturzes ist ja erfahrungsgemäß bei der Eingabe relativ groß. Es gibt nämlich so viele Möglichkeiten der Dateneingabe, daß sie oft vom Programmierer nicht alle überprüft werden können. Deshalb sollte hier eine hohe Sicherheit angestrebt werden. Testen Sie vielleicht das Programm EINGABE auf seine Absturzmöglichkeit, d. h., suchen



ein gutes menu macht stark

Sie Schleifen, aus denen es nicht herauskommt. Sie werden so für eigene Programme Erfahrungen sammeln können!
Die Gefahr des Systemabsturzes kann wesentlich verringert werden mit der:

b) Eingabe über den Cursor

Immer mehr setzt sich eine Eingabetechnik durch, welche nur die Position des Cursors auswertet. Der Cursor wird dazu mit den Sondertasten, einer Maus, dem Joystick oder Lichtgriffel gesteuert. Das kann beispielsweise auch innerhalb eines Menus erfolgen. Wenn dadurch der Cursor das gewünschte Menuwort erreicht hat, erfolgt die Auslösung des entsprechenden Programms nur noch durch Quittieren über die ENTER-Taste, oder das Drücken der Maustaste.
Auf unseren Kleinstrechnern ist eine solche Eingabe nicht ohne weiteres zu realisieren, da sich die verschiedenen Typen u. a. in ihren Betriebssystemen unterscheiden. Das Programm MENU mußte, damit es auf den Rechnern KC 85/1 und KC 87 bzw. KC 85/2,3 lauffähig ist, etwas komplexer gestaltet werden. Es simuliert mittels einer speziellen Fenstertechnik eine Maus. Dabei existieren drei Fenster. Angewendet wird diese Technik auf die Umwandlung Dez → Hex, jedoch mit einer langen Arithmetik, wie sie öfter beim Programmieren gebraucht wird.
Das Menu wird im ersten Fenster auf dem Bildschirm angezeigt. Im zweiten Fenster können die Zahlenwerte eingegeben werden, und dort erscheint auch das Ergebnis. Entscheidend ist das dritte Fenster, in dem man auswählt, wie der Cursor im ersten Fenster bewegt wird. Dazu existieren drei Eingabemöglichkeiten im dritten Fenster. Mit H und T kann der Cursor im ersten Fenster hoch und runter bewegt werden. Mit B wird diese Menuwahl bestätigt. Natürlich wäre es naheliegend, für die Menuauswahl im ersten Fenster die Cursortasten direkt zu verwenden. Dies hätte aber für die beiden Rechnertypen zu sehr verschiedenen Programmen geführt.
Das Programm läuft folgendermaßen ab: Auf dem Bildschirm sehen wir das Menufenster mit dem Cursor. Im dritten Fenster sehen wir einen zweiten Cursor, mit dem wir den Cursor im ersten Fenster steuern können, nachdem eine Bestätigung erfolgte. Dann kann im zweiten Fenster die Eingabe realisiert werden, die das Rechenprogramm betrifft. Es zeigt sich, daß es mit so einer Eingabetechnik nicht zum Programmabsturz kommt, da die Eingabe über die Fensterwahl so beschränkt ist, daß alle Falscheingaben vom Programmierer leicht abgefangen werden können. Außerdem werden sie weitgehend automatisch abgeblockt, da die Position des Cursors eindeutig auf dem Bildschirm gegeben ist.
Zu einem übersichtlichen Menu gehören auch Sonderzeichen, die im normalen Zeichensatz nicht vorhanden sind. Der KC 85/1 und KC 87 bietet mit seiner fest implementierten Pseudografik unmittelbar einige Möglichkeiten. Für den KC 85/2,3 müssen solche Zeichen erst mit einem Programm erzeugt und generiert werden. In der Regel geschieht das über Maschinencode oder Zahlen, die an die entsprechenden Stellen gepoket werden. Das ist relativ schwierig.
Eine einfache (wenn auch nicht universelle) Möglichkeit, Sonderzeichen zu erzeugen, bietet das Programm ZEICHEN. Dazu werden in acht DATA-Zeilen zu je acht Zeichen die gewünschten Sonderzeichen vergrößert abgelegt (Zeilen 100 bis 170). Danach muß das Programm viele Umrechnungen realisieren, bis es das Zeichen an die richtige Stelle poken kann. Das Programm ZEICHEN soll für Sie eine Anregung sein. Durch Änderung der DATA-Zeilen können Sie sich leicht eigene Sonderzeichen erzeugen. Hier, wie auch bei vielen anderen Programmen, wird der Nutzen vor allem von der Phantasie des Programmierers bestimmt. Wir wünschen Ihnen dabei viel Spaß!
Ein Rechenprogramm zeichnet sich dadurch aus, daß wir es immer wieder von neuem ablaufen lassen können. Natürlich ist es uninteressant, immer zum gleichen Ergebnis zu kommen. Es müssen also die Eingabewerte (z. B. Parameter, Koordinaten, Meßdaten ...) verändert werden. Das erfordert aber in der Regel, im neuen Ablauf alle Werte neu eingeben zu müssen, was natürlich sehr aufwendig sein kann. Besser ist es, von den vorangegangenen Eingabewerten auszugehen, diese auf dem Bildschirm anzuzeigen und nur dann, wenn ein Wert verändert werden soll, diesen neu einzugeben. Soll dagegen der alte Wert beibehalten werden, braucht nur die ENTER-Taste betätigt zu werden. Diese Methode ist äußerst effektiv, wenn von vielen Daten immer nur einige geändert werden müssen.
Genau in diesem Sinne ist das Programm INKOR geschrieben. Es ist jedoch nur auf dem KC 85/2,3 lauffähig, da viele Steuerzeichen und Abfragen vom Bildschirm benötigt werden. Obwohl es dadurch nicht leicht zu durchschauen ist, sollten Sie trotzdem versuchen, das Programm INKOR zu analysieren. Es wird Ihnen bei der Entwicklung eigener Programme hilfreich sein.

Zusammenfassung
Menu- und Eingabetechniken

1. Die Menus sollten kurz und eindeutig sein. Die Praxis zeigt, daß zu lange Texte meist nicht gelesen werden.
2. Die Menus sollten auch gewissen ästhetischen Ansprüchen genügen. Wenn man die Möglichkeit hat, Sonderzeichen zu generieren, wie z. B. beim KC 85/2 und 3 sollte davon Gebrauch gemacht werden.
3. Für die Antworten sollten nicht zu viele Wahlmöglichkeiten angeboten werden. Ein Optimum liegt zwischen 3 und 8. Sind mehr notwendig, so ist es besser, Untermenus zu bilden.
4. Bei der Eingabe sollte jedes Programm so gestaltet sein, daß keine Eingabe zu einer fehlerhaften Reaktion des Rechners oder gar zum Absturz führt. Diese Forderung ist sehr schwer zu realisieren, da in der Regel die volle Kombinatorik aller Eingaben schwer zu überblicken ist. Dennoch sollte der Programmierer hierbei sehr große Sorgfalt walten lassen.
5. Die Cursor-bezogene Eingabe ist in BASIC schwer zu realisieren. Sie hat aber drei wichtige Vorteile:
   - Es werden Fehleingaben nahezu vollständig unterbunden
   - Die Menu-Gestaltung läßt sich in besonders übersichtlicher Form realisieren.
   - Sie führt zu modernen Eingabetechniken, die z. B. durch eine Maus oder Joystick gesteuert werden.

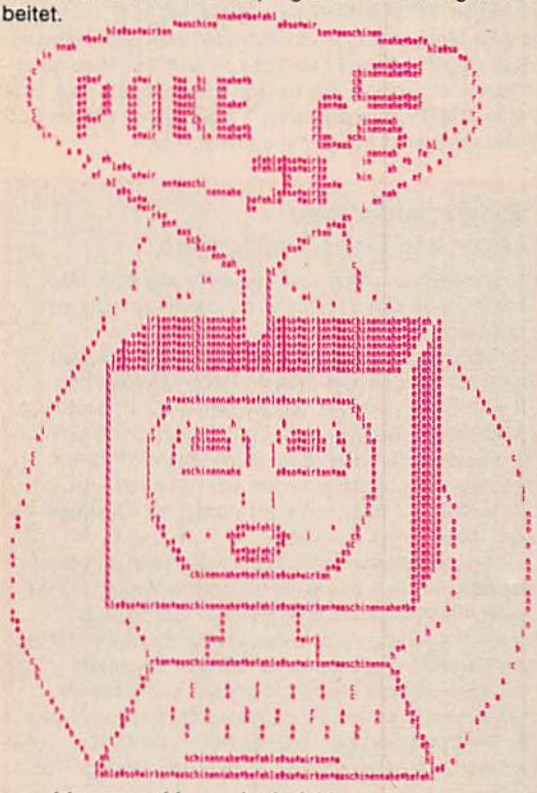
9. Maschinennahe Befehle

Die Bezeichnung maschinennah soll hier bedeuten, daß mit diesen Befehlen unmittelbar auf Ressourcen des Rechners zugegriffen werden kann. Dies bringt oft Vorteile bezüglich:
- Geschwindigkeit,
- Übersichtlichkeit
- Entstehen neuer Möglichkeiten.

Natürlich entstehen dadurch auch Nachteile. Sie betreffen u. a.:

- Probleme bei der Anwendung in anderen Rechnern
- erforderliche Kenntnisse des Maschinencodes
- Gefahr des Systemabsturzes
- komplizierte Erzeugung des Programms
- zerteiltes Programm
- zusätzliche Dokumentation
- Einschalten des IMR
- gemeinsames Retten von BASIC-Maschinen-Code
- erforderliche Wahl des günstigen Speicherbereiches.

Bei einem fertigen Programm kommen fast nur die Vorteile zum Tragen. Eventuell ist aber die Dokumentation komplizierter. Die Notwendigkeit der Übertragung des Programms auf andere Rechner tritt nur in Sonderfällen ein. Sie wird fast unmöglich, wenn die Rechner unterschiedliche Mikroprozessoren haben. Auch bei stark abweichenden Betriebssystemen kann es schwierig sein. Auf alle Fälle ist es zweckmäßig, immer den Quellcode des Maschinenprogramms mitzuteilen. Das endgültige Programm sollte so gestaltet sein, daß der Nutzer nicht merkt, daß ein Maschinenprogramm im Hintergrund arbeitet.



so wirken maschinennahe befehle

Bei der Entwicklung eines kombinierten BASIC-Maschinen-Programms sind jedoch die o. g. Probleme recht bedeutsam. Besonders die Gefahr des Systemabsturzes, der ja in BASIC ohne maschinennahe Befehle praktisch nicht vorkommt, sollte genau beachtet werden. Programm und Teilroutinen müssen häufiger gerettet und bis zur Fertigstellung aufgehoben werden.
Die maschinennahen Befehle bestehen aus fünf unterschiedlichen Gruppen:

- mittelbare Befehle
  CLEAR n, m, LINES n, NULL n, RESTORE n, WIDTH n, OPEN, CLOSE, PRINT #n usw.
- direkter Zugriff zum Arbeitsspeicher
  PEEK, DEEK, POKE, DOKE, VPEEK, VPOKE
- direkter Zugriff auf die Peripherie
  INP, OUT, WAIT
- Ausnutzen von Maschinenroutinen
  CALL, CALL*
- Maschinenroutinen mit Parameter
  USR

CALL wurde im Kapitel 4 behandelt, kann hier also entfallen.
Um die Verhältnisse am oberen Ende des Speicherbereichs zu überblicken, erweist sich das Programm CLEAR als nützlich. Mit ihm lassen sich experimentell die beiden Parameter in ihrer Wirkung erproben.

a) Speicherzugriffe

Beim Speicherzugriff mit maschinennahen Befehlen sind die folgenden unterschiedlichen Bereiche bedeutsam:

- unbenutzte Gebiete
- innerhalb des Programms, der Variablen usw.

Als Anwendungsfall sollen möglichst viele Zahlenwerte zwischen 0 und 254 gespeichert werden. Jeder Wert ist dabei günstig in einem Byte unterzubringen. Hierfür eignet sich jener Speicherbereich, der zwischen Array- und Stack-Ende liegt. Der nutzbare Adreßbereich ist folglich über die Pointer 987 und 854 zu gewinnen. So entsteht das Grundprogramm DATEN, welches allerdings nur eine einfache Eingabe- und Anzeige-Routine enthält. Es ist für die praktikable Anwendung durch weitere Routinen zu ergänzen. Das Programm funktioniert folgendermaßen:
In B wird die Startadresse für die Ablage gespeichert, und A enthält die maximale Anzahl Byte, die in diesem Programm verwendet werden können. Sie wird ohne Erweiterungs-RAM bereits mit 14 655 angezeigt. Dies ist eine recht große Anzahl, die mit anderen Methoden nicht zu erreichen sein dürfte. Denn selbst bei Verwendung eines Array müßte es zusätzlich gelingen, die vier Byte jedes Elements z. B. mittels komplizierter Codierung einzeln zu nutzen. Bei jeder anderen Variante geht noch mehr Speicherplatz für die Namen der Variablen verloren. Bei dem Programm sind folgende Besonderheiten zu beachten:

- Der Speicherraum sollte aus Gründen der Sicherheit immer erst dann bestimmt werden, wenn bereits alle benutzten Variablen und Felder initialisiert sind. Deshalb ist hier die Zeile 20 vor 30 notwendig. Die Variablen A, B, C wurden bewußt abweichend von den Gepflogenheiten über einen logischen Vergleich initialisiert.
- Der Stack sollte mit etwa 100 vom möglichen Speicherraum abgezogen werden.
- Mit den Zeilen 70 bis 100 erfolgt die Eingabe. Dabei werden die Zahlenwerte automatisch auf 1 Byte beschränkt. Nichtganzzahlige Werte werden auf ihren Integer-Bereich reduziert.
- Die Anzeige endet beim Erreichen eines Wertes von 255.

Eine (oft verpönte) Methode besteht darin, Programme sich selbst modifizieren zu lassen. Für BASIC lassen sich dabei relativ gutartige Varianten realisieren. Das Beispielprogramm SPRICH fleddert Sprichwörter. Es ist so gestaltet, daß leicht weitere Sprichwörter ergänzt werden können. Sie müssen aus jeweils zwei Zeilen bestehen. Per Zufall wird abwechselnd zwischen einer ersten und einer zweiten Zeile unterschiedlicher Sprichwörter ausgewählt. Natürlich ließe sich so etwas auch über ein hinreichend großes String-Array realisieren. Hier wird jedoch mit DATA-Zeilen und RESTORE n gearbeitet, wobei das n über Zufall gepoket wird. So wird auch die Garbage Collection unterbunden.
In der Zeile 40 wird der erste Pointer auf A übertragen. In den Zeilen 50 bis 70 werden die Pointer verfolgt bis eine Null das Programmende anzeigt. ELSE führt dann aus der unendlichen Schleife heraus. Dabei enthält C den letzten echten Pointer. Zwei Adressen weiter befindet

sich die letzte Zeilennummer. Sie muß deshalb so kompliziert bestimmt werden, weil ja Sprichwörter nach Belieben ergänzt werden sollen. Aus ihr wird dann die Anzahl der Sprichwörter bestimmt. Die Sprichwörter beginnen in Zeile 200. Daher ist C-200 die Zeilenzahl für alle Sprichwörter. Wird dieser Wert durch 20 geteilt, so ergibt sich die Anzahl der vorhandenen Sprichwörter. Dieser Wert wird in C abgelegt.
Ab Zeile 90 erfolgt die Suche nach dem Token RESTORE mit dem Wert 139. Mit beachtlicher Reserve dürfte es im Adreßbereich von 1 125 bis 5000 liegen. Die Token-Adresse wird auf A übertragen. Da diese Suche Zeit dauert, ist die Meldung von Zeile 20 nützlich.
Die Kombinationen der generierten Sprichwörter entstehen aus je einer geraden und ungeraden Zeilenzahl. Hierzu dient die Zeile 140 mit STEP 10 in J. Die Zufallszahl für die Zeilennummer wird dann in L abgelegt. Sie ist in ASCII-Zeichen zu wandeln. Die Zergliederung auf einzelne ASCII-Zeichen erfolgt in Zeile 160. Dabei ist das erste Zeichen ein Vorzeichen. Es wird mit der Zählung ab 2 unterdrückt. STR$(L) wandelt die Zeilennummer in einen String um, der durch MID$ zerlegt wird und über ASC als Zahl gepokt werden kann.
Die Zahl 220 nach RESTORE kann je nach Abbruch des Programms auch anders lauten. Das Leerzeichen nach RESTORE ist notwendig für das Poken. Falls das Programm geändert oder vielleicht einmal RENUMBER verwendet wird, ist darauf zu achten. Hier drei Ausgaben des Programms:

Wenn der Hahn kraeht auf dem Mist
dreist wenn er die Wahrheit spricht

Wenn der Abt zum Glase greift
muss man zweimal messen

Ehe man einmal abschneidet
aendert sich das Wetter oder es bleibt wies ist

In den DATA-Zeilen darf kein Komma vorkommen. Dann wird nämlich der folgende Teil abgetrennt. Deshalb wurde in Zeile 210 statt dessen ein Bindestrich verwendet.

b) Port-Zugriffe

Beim KC 85/2 bzw. 3 sind Portbefehle direkt nur auf die PIO und CTC anwendbar. Hierbei werden vom Interpreter die Maschinenbefehle IN A, (C) und OUT (C), A genutzt.
Für die im Bit-Betrieb initialisierte PIO gilt die Belegung:

bit	Dez-Wert	Kanal A 88H = 136D	Kanal B 89H = 137D
0	1	ROM	Lautstärke
1	2	RAM	Lautstärke
2	4	IMR	Lautstärke
3	8	RAM-Schreibschutz	Lautstärke
4	16	frei	Lautstärke
5	32	Tape-LED	frei
6	64	Motorsteuerung	frei
7	128	BASIC-ROM	blinken

Für einfache Anwendungen kommen also nur die Tape-LED, die Motorsteuerung und das Blinken auf dem Bildschirm in Betracht. Eine Beeinflussung der LED ermöglicht z. B. über einen Fototransistor eine galvanisch entkoppelte Steuerung von Geräten. Die Ansteuerung der LED ermöglicht das Programm LED.

c) Die USR-Funktion

Die USR-Funktion ist der anspruchsvollste BASIC-Befehl. Genau genommen ist sie jedoch nur eine CALL-Variante mit zusätzlicher Übergabe von Werten in das Maschinenprogramm und zurück. Es gilt die Syntax

A = USR(X).

Der X-Wert wird mit einer Routine des BASIC-Interpreters EIN an das Registerpaar DE übergeben. Sie liegt auf der Adresse 0C96FH und muß vom Maschinenprogramm aufgerufen werden.
Die Rückgabe von Rechenergebnissen kann auf die Variable A erfolgen. Die zugehörigen Werte müssen zuvor in den Registern A und B durch das Maschinenprogramm abgelegt sein. Die Übergabe erfolgt durch das Programm AUS auf der Adresse 0D0B1H.
An der Stelle von A und X können in der USR-Funktion auch andere Variablen stehen. Sie übernehmen dann die Funktion dieser Variablen in gleicher Weise.
Genau wie beim CALL-Aufruf muß vor dem USR-Aufruf die Startadresse der zugehörigen Maschinenroutine auf 772 mit DOKE übertragen werden. Zusätzlich müssen – ebenfalls genau wie beim CALL – im Maschinenprogramm alle Register zwischen Eintritt und Austritt unverändert bleiben. Weiter ist darauf zu achten, daß gegebenenfalls der Bildwiederholspeicher angeschaltet wird.
Im Grunde genommen kann die USR-Funktion immer vermieden werden. Dann sind vom BASIC-Programm die Werte zunächst mit POKE oder DOKE auf freie Speicherplätze abzulegen. Das Maschinenprogramm holt sie sich von dort und gibt seine Ergebnisse auf freie Speicherplätze zurück. Von dort holt sie sich das BASIC-Programm mit PEEK oder DEEK. Dies ist jedoch umständlicher und wohl auch nicht ganz so übersichtlich. Da die USR-Funktion generell nur 2 Byte empfangen und abliefern kann, gibt es durchaus Fälle, wo diese Methode zusätzlich angewendet wird.
Das Programmbeispiel BASGO betrifft die Erweiterung der BASIC-Befehle GOSUB und GOTO. Sie lassen als Argumente nur Zeilennummern, aber keine mathematischen Ausdrücke zu. Dies ist für einige Anwendungen von Nachteil. Insbesondere komplizierte Verteiler mit ON-GOTO oder ON-GOSUB werden dadurch manchmal recht unübersichtlich. Bei einem Menu mit Eingaben von 2 bis 20 kann man diese Werte auf die Variable A übergeben und danach mit GOTO 100*A auf die Zeilennummern 100, 200, 300... 1900 oder 2000 springen lassen. Das Maschinenprogramm muß dann aus USR(100*A) die Zeilennummer auf DE übergeben, um danach mit der im Interpreter vorhandenen GOTO-Routine fortzufahren. Wenn von Stackbereinigungen abgesehen wird, ist dieses Programm geradezu trivial. Es besteht aus den 7 Zeilen nach der Marke JUMP im Quellcode. Die RUN-Mode-Routine wertet das DE-Register zu diesem Zweck aus. Sie muß bei Aufruf der GOTO-Routine ab Adresse 0CA0AH aber auf dem Stack liegen.
Die GOSUB-Lösung ist etwas komplizierter. Zunächst wird der Bildwiederholspeicher eingeschaltet (IMR an). Außerdem erfolgt der Aufruf der Testroutine TEMO für die Fehleranzeige. Schließlich muß bei Rückkehr aus der Subroutine die aktuelle Zeilennummer gerettet werden. Sie ist in AKZEI = 358H abgelegt. Dieser Wert ist auf dem Stackpointer zu übertragen. Der Stack muß das Token für GOSUB 8CH enthalten. Es wird dem Register A übergeben und mit AF gepusht. Mit INC SP wird das Flagregister indirekt wieder vom Stapel entfernt. So ist alles vorbereitet, und der Aufruf der GOTO-Routine kann erfolgen. Sie ist durch diese Zusatzmaßnahmen zur GOSUB-Routine geworden.
Das BASIC-Programm BASGO enthält die Maschinenroutinen wieder in Zeile 0. Die zugehörigen Startadressen sind:

JUMP : 406H = 1030D
GOSUB : 413H = 1043D

Es trägt betont Demonstrationscharakter. In Zeile 3Ø können wir uns entscheiden, ob das erweiterte GOTO oder GOSUB genutzt werden soll. Bei GOTO wird in Zeile 5Ø die gewünschte Zeilennummer an A übergeben. Dann wird die entsprechende Adresse mit DOKE 772,1Ø3Ø eingetragen. Über die Eingabe 16Ø bis 18Ø kann eine dieser Zeilen mit dem erweiterten GOTO angesprochen werden. Dementsprechend erfolgt über die dort stehenden PRINT-Anweisungen sichtbar die Betätigung dieser Routine, und mit dem in Zeile 18Ø stehenden GOTO 5Ø geht es zurück zur erneuten Eingabe einer Zeile. Wird jetzt eine unzulässige Zeile, z. B. 2ØØ, eingegeben, so folgt die bekannte Ausschrift:

?UL-ERROR IN 7Ø

Bei Eingabe von z. B. 15Ø folgt analog der Fehlerhinweis, daß ein RETURN ohne GOSUB vorliegt.

Wird der erweiterte GOSUB-Befehl genutzt, so gelangt man zur Zeile 9Ø und kann korrekt die Zeilen 12Ø bis 15Ø durch die Eingabe aufrufen. Nach der Rückkehr aus der USR-Funktion in Zeile 11Ø erfolgt dann mittels GOTO 9Ø der Rücksprung zur erneuten Eingabezeile. Ein fehlerhafter Aufruf wird auch hier wieder entsprechend angezeigt. Etwas unerwartet verhält sich die Routine jedoch, wenn die Zeilen 16Ø bis 18Ø angewählt werden. Der PRINT-Ausdruck erfolgt korrekt; jedoch wird danach die Anfrage „GOTO-Zeile" sichtbar. Es wird die Anweisung GOTO 5Ø in Zeile 18Ø ausgeführt, und danach geht es korrekt gemäß diesem Befehl weiter. Nun ist aber der BASIC-Stapelspeicher nicht mehr in Ordnung. Das macht sich bei diesem einfachen Programm zwar noch nicht bemerkbar. Bei wirklichen Programmen sollte es vermieden werden.

Zusammenfassung
Maschinennahe Befehle

1. Maschinennahe Befehle ermöglichen den unmittelbaren Zugriff auf die Ressourcen des Rechners. Hierbei können folgende Befehlsgruppen unterschieden werden:
   - unmittelbare, wie CLEAR, LINES, RESTORE, WIDTH usw.
   - mit direktem Zugriff auf den Arbeitsspeicher, wie PEEK, DEEK, POKE, DOKE, VPEEK und VPOKE
   - mit direktem Zugriff auf die Peripherie, wie INP, OUT und WAIT
   - das Ausnutzen von Maschinenroutinen über CALL und
   - der leistungsfähige Befehl USR
2. Diese Befehle besitzen Vor- und Nachteile. Die Vorteile bestehen vor allem in der höheren Geschwindigkeit. Die Nachteile betreffen u. a.:
   - die Übertragbarkeit auf andere Rechner
   - notwendige Kenntnisse des Maschinencodes
   - Gefahr des Systemabsturzes
   - Wahl eines günstigen Speicherbereiches
3. Der Speicherzugriff mit maschinennahen Befehlen auf Bereiche innerhalb des Programms, der Variablen usw. muß mit größter Umsicht erfolgen.
4. In den unbenutzten Bereichen sind Daten mit hoher Dichte unter anderem durch POKE und DOKE abzulegen und mittels PEEK und DEEK zurückzuholen.
5. Der Speicherraum zwischen den Feldern und dem Stack sollte immer erst dann bestimmt werden, wenn bereits alle benutzten Variablen und Felder initialisiert sind.
6. Auch in BASIC existiert die Methode, ein Programm sich selbst modifizieren zu lassen. In Sonderfällen bringt diese verpönte Methode Vorteile.
7. Der Zugriff auf Ports erfolgt mit den Befehlen INP und OUT. Dies setzt genaue Kenntnisse bezüglich der Peripherie des Rechners voraus.
8. Die USR-Funktion ist eine CALL-Variante mit zusätzlicher Übergabe von Werten. Bei A=USR(X) wird der Wert von X an das DE-Register übergeben. Dies realisiert eine Routine des BASIC-Interpreters auf der Adresse C96F. Die Rückgabe von Ergebnissen kann auf die Variable A aus den Registern A und B erfolgen. Die zugehörige Routine liegt auf der Adresse DØB1. Die Startadresse der Maschinenroutine muß mit DOKE 772,Adr. eingetragen werden.

10. Anpassen von Daten

Für viele praktische Anwendungen ist es erforderlich, Meßwerte zu nutzen und auf ihre Richtigkeit oder Genauigkeit hin zu überprüfen. Besonders vorteilhaft sind dabei statistische Methoden. Der Begriff Meßwert soll hier vereinfacht nur als Zahlenwert verstanden werden, so wie er bei einer Messung anfällt.

Nehmen wir z. B. an, daß die Messung des Durchmessers eines Rundstahls die Werte 3,1 mm, 3,Ø4 mm, 3,13 mm usw. ergibt. Für diese wenigen Meßdaten sind die Mittelwertbildung oder andere Berechnungen relativ einfach. Fallen dagegen sehr viele Daten an, wie z. B. bei der statistischen Auswertung der Bevölkerungsstruktur eines Landes (Demographie), so sind Rechnerprogramme unbedingt notwendig.

Für derartige Betrachtungen ist die Fehlerrechnung entwickelt worden. In ihr kommen neben dem Mittelwert noch andere statistische Größen wie z. B. die Standardabweichung (Bereich um den Mittelwert), die Streuung u. a. vor. Sie geben Hinweise auf die Eigenschaften von Meßdaten und Fehlern. Mit einem Programm, das solche Größen berechnet, können aus den Meßdaten viele neuartige Aussagen erhalten werden. So gibt es ein Beispiel aus der Informationstheorie, bei dem Musikwerke analysiert wurden, und es zeigte sich, daß mit relativ einfachen Methoden ein komplizierteres statistisches Maß, die Kurtosis, in Übereinstimmung mit dem Zeitpunkt der Entstehung der Komposition steht.

Mit dem Programm STAMZA können verschiedene statistische Größen berechnet werden. Dazu müssen die entsprechenden Daten eingegeben werden (Zeile 8Ø). Danach erfolgt die Berechnung wichtiger statistischer Größen wie z. B. Mittelwert, Streuung, Standardabweichung, Schiefe, Steilheit und Kurtosis. Das Programm STAMZA kann leicht in andere Programme (z. B. als GOSUB-Routine) eingebaut werden und steht damit jederzeit für die praktische Anwendung zur Verfügung.

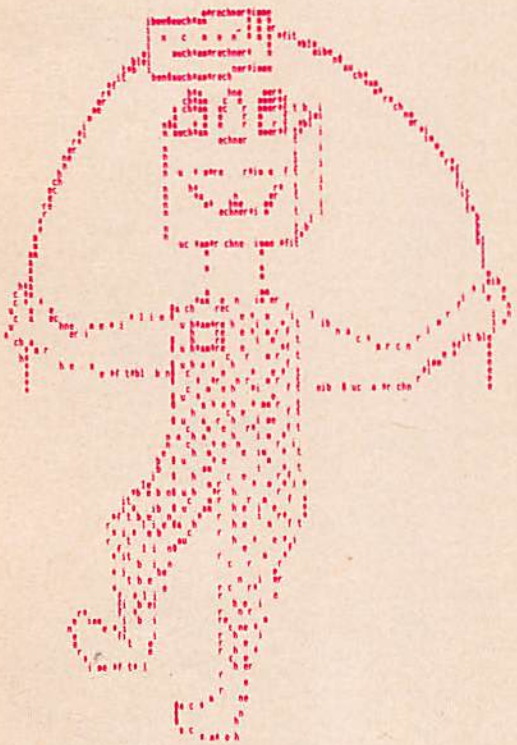
a) Einparametrige Anpassung (Fit)

Nehmen wir an, es existieren einhundert Bücher unterschiedlicher Seitenzahl. Es ist nun leicht, beispielsweise

die mittlere Seitenzahl zu bestimmen (z. B. mit dem Programm STAMZA).
Wollen wir dagegen untersuchen, ob zwischen der Seitenzahl eines Buches und dessen Preis ein Zusammenhang besteht, so ist eine andere Methode anzuwenden. Dafür benötigt man Funktionen, die beide Werte miteinander verkoppeln. In einem Programm müssen die Ausgangswerte eingegeben werden (z. B. Seitenzahl und Preis der Bücher), und das Programm muß den bestmöglichen Zusammenhang bezüglich der ausgewählten Funktion herstellen. Diesen Vorgang nennt man Anpassung von Daten oder, aus dem englischen abgeleitet, Fit.
Eine solche Problematik wird im Programm KORRE realisiert. Es sind dazu Datenpaare (X,Y) einzugeben, und das Programm ermittelt für vier wichtige Funktionen

- proportional : $Y = a * X$
- reziprok : $Y = a / X$
- exponentiell : $Y = a^x$
- potentiell : $Y = X^a$

die Parameter a und die dazugehörigen Korrelationskoeffizienten K. Der Korrelationskoeffizient sagt aus, wie gut die jeweilige Funktion dem Zusammenhang bezüglch der Datenpaare entspricht. K kann Werte zwischen Null (kein Zusammenhang zwischen den Ausgangswerten) und Eins (idealer Zusammenhang) annehmen. Für die weitere Auswertung ist dann meist die günstigste Funktion zu verwenden. Die Anpassung von Daten ist also eine Methode, mit der aus einer Vielzahl von Ausgangswerten verdichtete Aussagen getroffen werden können.



auch am rechner immer fit bleiben

b) Zweiparametrige Anpassung (Fit, Regression)

Für viele Anwendungen reicht eine Anpassung von Daten über einparametrische Funktionen nicht aus. Es existieren zahlreiche Methoden, bei denen mit zwei, drei und mehr Parametern operiert wird, um die Anpassung zu verbessern. Besonders wichtig ist die zweiparametrige Anpassung, auf die wir uns hier beschränken wollen. Eine Beispielfunktion hierzu ist:

$$Y = a * X + b$$

Andere wichtige Funktionen besitzen einen reziproken, exponentiellen, logarithmischen o. a. Zusammenhang.
In einem Programm müssen dann die beiden Parameter a und b bestimmt werden. Dies ist zwar rechentechnisch aufwendiger, dafür sind die erzielten Ergebnisse, d. h. die Anpassung, besser.
Die zweiparametrige Anpassung bietet zusätzlich die Möglichkeit, in die „Zukunft oder die Vergangenheit zu sehen". Das heißt, es können Aussagen getroffen werden, die außerhalb des vorhandenen Datenbereiches liegen. In diesem Fall spricht man auch häufig von Regression. Ein Beispiel: Als Eingabedaten liegt die jährliche Produktionsmenge von Fernsehgeräten eines Betriebes für den Zeitraum von 1960 bis 1987 vor. Für die o. g. Funktion wäre Y die jährliche Produktionsmenge und X das Jahr. Wenn mit einem Programm die Parameter a und b bestimmt wurden, so kann die Produktion von Fernsehgeräten in dem Betrieb z. B. für das Jahr 2000 vorausbestimmt werden (natürlich nicht mit absoluter Sicherheit).
Auch hier kommt man zu optimalen Ergebnissen, wenn man aus mehreren zweiparametrige Funktionen durch Vergleich der Korrelationskoeffizienten die günstigste Variante auswählt. Dieses einfache, aber keineswegs absurde Beispiel zeigt, wie vorteilhaft die zweiparametrige Anpassung gerade für prognostische Zwecke ist. Ein Programm hierfür ist das Programm FIT. Zur optimalen Anpassung der Eingabewerte werden fünf verschiedene Funktionen

$$Y = a + b * X$$
$$Y = a + b / X$$
$$Y = a + b * LN(x)$$
$$Y = a * b^x$$
$$Y = a * X^b$$

herangezogen und die dazugehörigen Parameter sowie die entsprechenden Korrelationskoeffizienten berechnet. Diese Funktionen erkennen Sie im Aufruf von Zeile 2ØØ. Für das Programm werden aber nicht nur sie benötigt, sondern auch die Linearisierung dieser Funktionen und ihre Umkehrfunktionen. Die Definition all dieser Funktionen erfolgt in den Zeilen 2Ø bis 16Ø.
Auch dieses Programm können Sie für Ihre speziellen Anwendungen durch geringfügige Änderungen nutzbringend einsetzen.

Zusammenfassung für dieses Kapitel auf S. 22

Zusammenfassung Anpassen von Daten

1. Meßwerte sind stets mit Fehlern behaftet. Durch wiederholte Messung bewirken die zufälligen Fehler eine Streuung um den richtigen Mittelwert. So ist es möglich, mit mehreren Messungen dem wirklichen Wert näher zu kommen oder ihn einzugrenzen.
2. Mit statistischen Methoden ist es möglich, Aussagen über die vermutliche Größe des Fehlers zu machen. Ein Maß hierfür ist die Streuung. Sie kann aus mehreren Meßwerten berechnet werden.
3. Es gibt verschiedene statistische Maße, welche Hinweise auf Eigenschaften von Meßwerten und Fehlern geben.
4. Bei mehreren Anwendungen ändert sich eine Ausgangsgröße unter dem Einfluß der Eingangsgröße. Wichtig ist es hierbei, den Zusammenhang zu bestimmen. In vielen Fällen besteht direkte Proportionalität, die aber durch Störungen verdeckt wird. Ein Maß für den Grad der Gültigkeit einer solchen Annahme ist die Korrelation. Sowohl der Proportionalitätsfaktor als auch der Korrelationskoeffizient sind gut zu berechnen.
5. Neben dem linearen Zusammenhang existieren noch viele andere. Besonders wichtig sind der reziproke, exponentielle und jener mit einer Potenz.
6. Für den Zusammenhang von Eingangs- und Ausgangsgrößen sind zweiparametrische Funktionen von großer Bedeutung. Der meist verwendete Ansatz geht von der Beziehung $Y=A+B*X$ aus. Nach der Methode der kleinsten quadratischen Abweichung sind die Parameter A und B sowie der Korrelationskoeffizient zu berechnen. Dieses Prinzip heißt im englischen FIT. Im deutschen ist Anpassung gebräuchlich. Nicht selten wird auch von Regression gesprochen. Dieses beinhaltet aber bereits eine zusätzliche Anwendung, nämlich mit den berechneten Parametern auf Werte zu schließen, die außerhalb der Daten liegen.
7. Neben dem linearen FIT gibt es auch logarithmische, exponentielle usw. Sie sind jedoch nicht unmittelbar zu realisieren. Es muß zusätzlich die Linearisierung der Funktion durchgeführt werden.

Programme

Zum Kapitel 1

FILE

```
780 REM###### FILE ######
790 REM----- ZAHLEN -----
800 INPUT"Startwert >0 ";A: A=RND(-A)
810 INPUT"Zahlenbereich";A,B: B=B-A
820 INPUT"   Anzahl    ";N: DIM AZ(N),A(N)
830 FOR X=0 TO N: AZ(X)=A+B*RND(1):A(X)=X: NEXT: RETURN
890 REM----- TEXTE -------
900 INPUT"Startwert >0 ";A: A=RND(-A)
910 CLEAR 10000: A=ASC("A"): B=ASC("Z")-A+1
920 INPUT"   Anzahl    ";N: DIM A$(N),A(N)
930 FOR X=0 TO N: A$="": A(X)=X
940 : FOR Y=0 TO 20*RND(1)
950 :  A$=A$+CHR$(A+B*RND(1))
960 NEXT: A$(X)=A$: NEXT: RETURN
```

ZABUB

```
10 PRINT TAB(10);"## ZABUB ##": PRINT
20 GOSUB 800: INPUT"Start";A
30 FOR X=0 TO N: F=0: FOR Y=0 TO N-1
40 IF AZ(Y)>AZ(Y+1) THEN A=AZ(Y): AZ(Y)=AZ(Y+1): AZ(Y+1)=A: F=1
50 NEXT: PRINT X;: IF F=0 THEN X=N
60 NEXT
70 FOR X=0 TO N: PRINT AZ(X),: NEXT: END
790 REM----- ZAHLEN ----
800 INPUT"Startwert >0 ";A: A=RND(-A)
810 INPUT"Zahlenbereich";A,B: B=B-A
820 INPUT"   Anzahl    ";N: DIM AZ(N)
830 FOR X=0 TO N: AZ(X)=A+B*RND(1): NEXT: RETURN
```

WOBUB

```
10 PRINT TAB(10);"## WOBUB ##": PRINT
20 CLEAR 10000: GOSUB 900: INPUT"Start";A
30 FOR X=0 TO N: F=0: FOR Y=0 TO N-1
40 IF A$(Y)>A$(Y+1) THEN A$=A$(Y): A$(Y)=A$(Y+1):A$(Y+1)=A$: F=1
50 NEXT:PRINT X;: IF F=0 THEN X=N
60 NEXT
70 FOR X=0 TO N: PRINT A$(X): NEXT: END
890 REM----- TEXTE -------
900 INPUT"Startwert >0 ";A: A=RND(-A)
910 A=ASC("A"): B=ASC("Z")-A+1
920 INPUT"   Anzahl    ";N: DIM A$(N), A(N)
930 FOR X=0 TO N: A$=""
940 : FOR Y=0 TO 20*RND(1)
950 :  A$=A$+CHR$(A+B*RND(1))
960 NEXT: A$(X)=A$: NEXT: RETURN
```

BUB 2

```
10 PRINT TAB(10);"## BUB2 ##": PRINT
20 CLEAR 10000: GOSUB 900: INPUT"Start";A
30 FOR X=0 TO N: F=0: FOR Y=0 TO N-1-X
40 IF A$(A(Y))>A$(A(Y+1)) THEN A=A(Y): A(Y)=A(Y+1): A(Y+1)=A: F=1
50 NEXT: PRINT X;: IF F=0 THEN X=N
60 NEXT
70 FOR X=0 TO N: PRINT A$(A(X)): NEXT: END
890 REM----- TEXTE -------
900 INPUT"Startwert >0 ";A: A=RND(-A)
910 A=ASC("A"): B=ASC("Z")-A+1
920 INPUT"   Anzahl    ";N: DIM A$(N),A(N)
930 FOR X=0 TO N: A$=""
940 : FOR Y=0 TO 20*RND(1)
950 :  A$=A$+CHR$(A+B*RND(1))
960 NEXT: A$(X)=A$:A(X)=X: NEXT: RETURN
```

SHAKER

```
10 PRINT TAB(10);"## SHAKER ##": PRINT
20 CLEAR 10000: GOSUB 900: INPUT"Start";A
30 FOR X=0 TO N STEP 2: F=0: X2=INT(X/2): FOR Y=X2 TO N-1-X2
40 IF A$(A(Y))>A$(A(Y+1)) THEN A=A(Y): A(Y)=A(Y+1): A(Y+1)=A: F=1
50 NEXT: PRINT X;: IF F=0 THEN X=N: GOTO 90
60 : F=0: FOR Y=N-2-X2 TO X2 STEP -1
70 IF A$(A(Y))>A$(A(Y+1)) THEN A=A(Y): A(Y)=A(Y+1): A(Y+1)=A: F=1
80 NEXT: PRINT X+1;: IF F=0 THEN X=N
90 NEXT
100 FOR X=0 TO N: PRINT A$(A(X)): NEXT: END
890 REM----- TEXTE -------
900 INPUT"Startwert>0 ";A: A=RND(-A)
910 A=ASC("A"): B=ASC("Z")-A+1
920 INPUT"   Anzahl    ";N: DIM A$(N), A(N)
930 FOR X=0 TO N: A$=""
940 : FOR Y=0 TO 20*RND(1)
950 :  A$=A$+CHR$(A+B*RND(1))
960 NEXT: A$(X)=A$:A(X)=X: NEXT: RETURN
```

QUICK

```
10 PRINT TAB(10);"## QUICK ##": PRINT
20 CLEAR 10000: DIM S(18,1): GOSUB 900: INPUT"Start";A
30 X=0: S(0,0)=0: S(0,1)=N
40 L=S(X,0): R=S(X,1): X=X-1
50 Y=L: J=R: H=A(INT((R+L)/2))
60 IF A$(A(Y))>=A$(H) GOTO 90
70 IF Y>=R GOTO 90
80 Y=Y+1: GOTO 60
90 IF A$(A(J))<=A$(H) THEN 120
100 IF J<=L GOTO 120
110 J=J-1: GOTO 90
120 IF Y<=J THEN M=A(Y): A(Y)=A(J): A(J)=M: Y=Y+1: J=J-1
130 IF Y<=J GOTO 60
140 IF (R-Y)<=(J-L)GOTO 170
150 IF L<J THEN X=X+1: S(X,0)=L: S(X,1)=J
160 L=Y: GOTO 190
170 IF Y<R THEN X=X+1: S(X,0)=Y: S(X,1)=R
180 R=J
190 IF R>L GOTO 50
200 IF X>=0 THEN PRINT X;: GOTO 40
210 FOR X=0 TO N: PRINT A$(A(X)): NEXT: END
890 REM----- TEXTE -------
900 INPUT"Startwert> 0 ";A: A=RND(-A)
910 A=ASC("A"): B=ASC("Z")-A+1
920 INPUT"   Anzahl    ";N: DIM A$(N), A(N)
930 FOR X=0 TO N: A$=""
940 : FOR Y=0 TO 20*RND(1)
950 :  A$=A$+CHR$(A+B*RND(1))
960 NEXT: A$(X)=A$:A(X)=X: NEXT: RETURN
```

Zum Kapitel 2

RUND

```
10 CLS: PRINT"  ##  RUND  ##"": PRINT
20 X=1234567: B=-1/7
30 FOR I=-4 TO 4: C=10^I: PRINT"auf";C: Y=X
40   FOR J=1 TO 12
50    PRINT INT(Y/C+.5)*C,: Y=Y*B
60   NEXT: PRINT: INPUT"weiter";Z
70 NEXT
```

```
  ##  RUND  ##
auf 1E-04
 1.23457E+06 -176367       25195.2
-3599.32      514.189     -73.4555
 10.4936     -1.4991       .2142
-.0306        4.4E-03     -6E-04

auf 1E-03
 1.23457E+06 -176367       25195.2
-3599.32      514.189     -73.456
 10.494      -1.499        .214
-.031         4E-03       -1E-03

auf .01
 1.23457E+06 -176367       25195.3
-3599.32      514.19      -73.46
 10.49       -1.5          .21
-.03          0            0
   usw.
```

REFOR

```
10 CLS: PRINT"     ##  REFOR  ##""
20 X=2.1E8: B=-1/7
30 FOR I=1 TO 15
40 PRINT TAB(20-LEN(STR$(X)));X: X=X*B
50 NEXT
```

```
     ##  REFOR  ##
          2.1E+08
           -3E+07
       4.28572E+06
           -612245
           87463.6
          -12494.8
           1784.97
          -254.996
            36.428
            -5.204
           .743428
          -.106204
           .015172
      -2.16743E-03
       3.09633E-04
```

PUFOR

```
10 CLS: PRINT"    ##  PUFOR  ##""
20 X=1.23E8: B=-1/8
30 FOR I=1 TO 15: X$=STR$(X): L=LEN(X$)
40   FOR J=1 TO L
50    IF MID$(X$,J,1)="." THEN L=J-1: J=15
60   NEXT: PRINT TAB(10-L);X: X=X*B
70 NEXT
```

```
     ##  PUFOR  ##
           1.23E+08
          -1.5375E+07
           1.92188E+06
        -240234
          30029.3
          -3753.66
            469.208
            -58.651
              7.33137
             -.916422
              .114553
             -.0143191
             1.78989E-03
            -2.23736E-04
             2.7967E-05
```

KOMBA

```
10 CLS: PRINT TAB(10)"### KOMBA ###": GOTO 680
20 REM======= Multiplikation ==========
30 Y=0: FOR I=0 TO N
40   X=A(I)*A+Y: Y=INT(X/S): A(I)=X-S*Y
50 NEXT: IF Y>0 THEN N=N+1: A(N)=Y
60 IF Y>999 THEN A(N)=Y-S: N=N+1: A(N)=1
70 RETURN
80 REM========== Division =============
90 Y=0: FOR I=N TO 0 STEP -1
100   X=A(I)+S*Y: A(I)=INT(X/A): Y=X-A*A(I)
110 NEXT: IF A(N)=0 THEN N=N-1: IF N<0 THEN N=0
120 IF Y>0 THEN PRINT: PRINT"####  Rest="Y"#####"
130 RETURN
140 REM========== Eingabe =============
150 INPUT X: IF X<2 THEN PRINT"zu klein": GOTO 150
160 IF X>999 THEN PRINT"zu gross": GOTO 150
170 IF X-INT(X) THEN PRINT"nicht integer": GOTO 150
180 RETURN
190 REM========== Anzeige =============
200 FOR I=N TO 0 STEP -12: FOR J=0 TO 11
210   C=A(I-J): IF C>99 THEN PRINT C;: GOTO 230
220   C$="000"+MID$(STR$(C),2): PRINT" "MID$(C$,LEN(C$)-2)+" ";
230   IF I-J=0 THEN J=15
240 NEXT: PRINT: NEXT
250 PRINT"Die Zahl hat"; 3*N+LEN(STR$(A(N)))-1; "Stellen"
260 PRINT: RETURN
270 REM======= Lange Eingabe ==========
280 PRINT: PRINT"Eingabe nur in 3er-Gruppen"
290 PRINT"Jede Eingabe wird so interpretiert"
300 PRINT"Ende der Eingabe mit negativer Zahl": PRINT
310 FOR I=M TO 0 STEP -1: INPUT A(I)
320   IF A(I)-INT(A(I)) THEN PRINT"nicht integer": GOTO 320
330   IF A(I)>999 THEN PRINT"zu gross": GOTO 320
340   IF A(I)<0 THEN N=M-I-1: I=0
350 NEXT: FOR I=0 TO N: A(I)=A(M-N+I): NEXT: RETURN
360 REM========== Fakultaet ============
370 PRINT"Eingabe n=";: GOSUB 150: N=0: A(0)=1: L=X
380 FOR A=2 TO L: GOSUB 30: NEXT
390 PRINT: PRINT"Die Fakultaet von"; L; "betraegt:": RETURN
400 REM========== Potenz =============
410 PRINT"Fuer x^y Eingabe x=";: GOSUB 150: N=0: A(0)=X: A=X
420 PRINT"y=";: GOSUB 150: L=X
430 FOR J=2 TO L: GOSUB 30: NEXT: PRINT
440 PRINT A; "^"; L; "betraegt:": RETURN
450 REM========== n ueber m ============
460 PRINT"n ueber m; n=";: GOSUB 150: L=X: N=0: A(0)=X
470 PRINT"m=";: GOSUB 150: D=X
480 IF L<X+X THEN D=L-X: IF D<2 GOTO 460
490 FOR J=2 TO D: A=L+1-J: GOSUB 30: A=J: GOSUB 90: NEXT: PRINT
500 PRINT L; "ueber"; D; "bzw. ueber"; L-D; "betraegt:": RETURN
510 REM======== Lange Zahl ============
```

```
520 PRINT"zuvor bezueglich langer Zahl"
530 INPUT"0:nichts 1:eingeben 2:anzeigen";A
540 IF A=0 THEN RETURN
550 PRINT: ON A GOSUB 280, 200: RETURN
560 REM====== Multiplikation==========
570 GOSUB 520: PRINT"Multiplikator =";: GOSUB 150: A=X: GOSUB 30
580 PRINT"Multiplikation mit"; A; "ergiebt:": RETURN
590 REM========= Division =============
600 GOSUB 520: PRINT" Divisor =";: GOSUB 150: A=X: GOSUB 90
610 PRINT: PRINT"Division mit"; A; "ergiebt:": RETURN
620 REM======== n!/(n-m)! =============
630 PRINT"Fuer n!/(n-m)!; n=";: GOSUB 150: L=X
640 PRINT"m=";: GOSUB 150: D=L-X: N=0: A(0)=D+1
650 FOR A=D+2 TO L: GOSUB 30: NEXT: PRINT
660 PRINT L; "! /"; D; "! betraegt:": RETURN
670 REM========== Start ==============
680 PRINT"H.Voelz";TAB(32)"28.6.87": PRINT: S=1000
690 INPUT"Maximale Stellenzahl";M: M=INT((M+2)/3): DIM A(M)
700 PRINT"Funktionen :"
710 P=B: PRINT: PRINT"1:Fakultaet   2:Potenzen  3:n ueber m"
720 INPUT"4:Multiplikt. 5:Division  6:n!/(n-m)!";B: PRINT
730 ON B GOSUB 370, 410, 460, 570, 600, 630: GOSUB 200: GOTO 700
```

EULER

```
10 CLS: PRINT"Eulersche Zahl": PRINT
20 INPUT"Stellenzahl";S
30 PRINT: PRINT"####  bitte warten  ####": PRINT
40 V=3: Z=10^V: A=S/V+2.9: DIM B(A), C(A): N=1: B(A)=1: C(A)=1
50 U=0: FOR J=1 TO A: Q=B(J)+C(J)+U: U=INT(Q/Z)
60 B(J)=INT((Q/Z-U)*Z+.5): NEXT: N=N+1
70 R=0: FOR J=A TO 1 STEP-1
80 Q=R*Z+C(J): C(J)=INT(Q/N): R=INT(Q-N*C(J)): NEXT
90 IF C(1)>0 OR N=2 GOTO 50
100 FOR J=A TO 2 STEP -1: PRINT B(J);: NEXT
110 !          1    1    1    1
120 ! e = 1 + -- + -- + -- + -- + ...
130 !         1!   2!   3!   4!
```

```
Eulersche Zahl

Stellenzahl 30

####  bitte warten  ####

 2  718  281  828  459  45  235  360  287  471  352  2  718  281
```

PI

```
10 CLS: PRINT TAB(12);"### PI ###": PRINT: GOTO 130
20 !----- Subroutinen ------
30 R=0: FOR J=A TO 1 STEP -1: Q=R*Z+A(J)
40 A(J)=INT(Q/N): R=INT(Q-N*A(J)): NEXT: RETURN
50 R=0: FOR J=A TO 1 STEP -1: Q=Z*R+B(J)
60 B(J)=INT(Q/N): R=INT(Q-N*B(J)): NEXT: RETURN
70 N=25: GOSUB 30: N=2*I+1: GOSUB 30: GOSUB 50: N=57121: GOSUB 50
80 F=2*I-1: U=0: FOR J=1 TO A: Q=A(J)*F
90 W=INT(Q/Z): A(J)=INT((Q/Z-W)*Z+.5)+U: U=W: NEXT
100 FOR J=1 TO A: Q=B(J)*F: W=INT(Q/Z)
110 B(J)=INT((Q/Z-W)*Z+.5)+U: U=W: NEXT: RETURN
120 !------- MAIN ----------
130 INPUT"Stellenzahl";S
140 PRINT: PRINT"       ### bitte warten ###": PRINT
150 V=2: Z=10^V: A=S/V+3.9: DIMA(A), B(A), C(A)
160 A(A)=16: B(A)=4: I=1: N=5: GOSUB 30: N=239: GOSUB 50
170 U=0: FOR J=1 TO A: Q=C(J)+A(J)-B(J)+U: U=INT(Q/Z)
180 C(J)=INT((Q/Z-U)*Z+.5): NEXT: GOSUB 70: I=I+1
190 U=0: FOR J=1 TO A: Q=C(J)-A(J)+B(J)+U: U=INT(Q/Z)
200 C(J)=INT((Q/Z-U)*Z+.5): NEXT
210 GOSUB 70: I=I+1: IF A(1)>0 OR N=3 GOTO 170
220 FOR J=A TO 3 STEP -1: PRINT C(J);: NEXT
230 ! Formel von Gregory 1671:
240 !                  1   1   1
250 !PI/4=ATN(1)= 1 - - + - - - +-...
260 !                  3   5   7
270 ! Formel von J.Machim 1706                    ### PI ###
280 !                  1         1
290 ! ATN(1) = 4*ATN(-) - ATN(---)          Stellenahl 20
300 !                  5        239
                                                ### bitte warten ###

                                              3  14  15  92  65  35  89  79  32  38  46
```

FAKUL

```
10 CLS: PRINT"### FAKUL ###": PRINT
20 INPUT"N =";N
30 X=LN(8*N*ATN(1))/2+N*LN(N)+N*LN(EXP(-1))
40 X=X/LN(10): Y=INT(X): X=X-Y: X=10^X
50 PRINT X; "E+"; Y
60 GOTO 20

### FAKUL ###

N = 127
 3.01081 E+ 213
N = 4576
 1.91095 E+ 14765
N = 1E+29
 1 E+ 2.85657E+30
```

Zum Kapitel 3

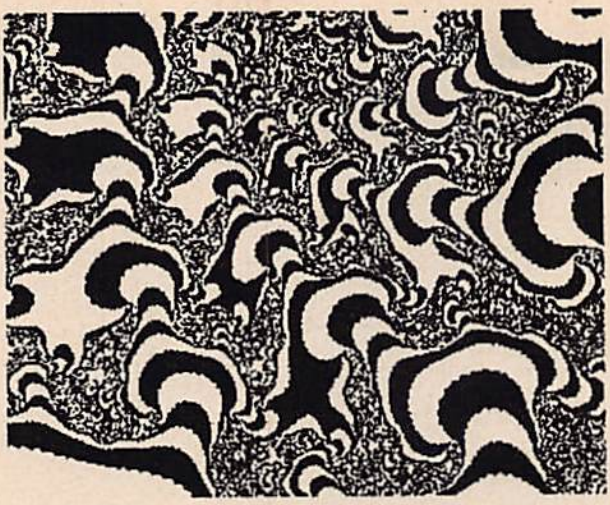
TEFRA

```
10 CLS: PRINT TAB(12);"## TEFRA ##": PRINT
20 H=199: A=0: B=0: M=0: X=0: Y=0: Q=23: IF PEEK(-5)<130 THEN Q=31: H=155
30 INPUT"X1, X2 =";X1,X2: XD=(X2-X1)/40: Z=27: G=50: E=.01
40 INPUT"Y1, Y2 =";Y1,Y2: YD=(Y2-Y1)/Q: Y=Y2+YD: WINDOW 0,Q,0,39: CLS
50 FOR I=0 TO Q-1: X=X1-XD: Y=Y-YD
60  FOR J=0 TO 39: X=X+XD: A=.5: B=0: C=A: D=B: F=H
70   FOR N=1 TO Z
80    C=A*A-B*B-X: D=A*B: D=D+D-Y
90    IF ABS(C-A)<E AND ABS(D-B)<E THEN F=N+95: N=Z
100   A=C: B=D: IF ABS(A) + ABS(B) >G THEN F=N+63 : N=Z
110 NEXT: PRINT CHR$(F);: NEXT: NEXT
120 INPUT"";A: IF A=0 THEN END
```

TEFRA

FRAKT

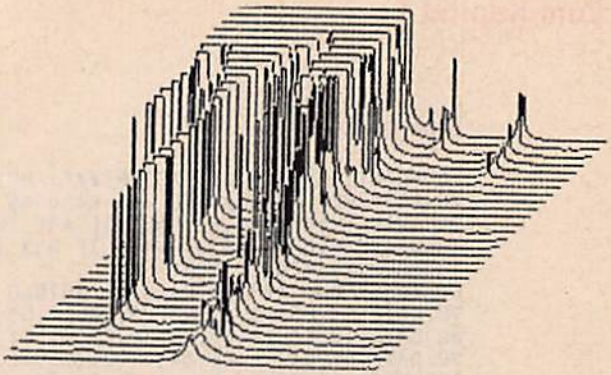
```
0 ---- Maschinencode --- ## FRAKT ## ----
10 CLEAR 30,6100: WINDOW 0,31,0,39: CLS: PRINT TAB(13);"## FRAKT ##": PRINT
20 A=0: B=A: C=A: D=A: Y=A: N=A: G=100: E=.01: H=A
30 INPUT"Zyklen =";Z
40 INPUT"X1, X2 =";X1, X2: XD=(X2-X1)/320: X=X1-XD
50 INPUT"Y1, Y2 =";Y1, Y2: YD=(Y2-Y1)/256: CLS
60 FOR I=0 TO 319: Y=Y1-YD: X=X+XD
70   FOR J=0 TO 255: Y=Y+YD: A=.5: B=0: H=.5
80     FOR N=1 TO Z
90      C=A*A-B*B-X: D=A*B: D=D+D-Y
100     IF ABS(C-A)<E AND ABS(D-B)<E THEN N=Z
110     A=C: B=D: IF ABS(A)+ABS(B) > G THEN H=N/2: N=Z
120    NEXT: IF H>INT(H) THEN PSET I,J,7
130 NEXT: NEXT: CALL*406
140 INPUT"";A: IF A<>0 THEN 140
```



.527 .537 .899 .187 Q35

FRAGEB

```
0 --- Maschinencode ---- ## FRAGEB ## ----
10 CLEAR 30,6100: WINDOW 0,31,0,39: DIM S(320): A=0: B=A: C=A: D=A: GOTO 90
20 '----------Iteration-----------
30 FOR N=1 TO Z
40   C=A*A-B*B-X: D=A*B: D=D+D-Y
50   IF ABS(C-A)<E AND ABS(D-B)<E THEN N=Z
60   A=C: B=D: IF ABS(A)+ABS(B) > G THEN H=N/2: N=Z
70 NEXT: RETURN
80 !-----------Main---------------
90 E=.01: Y=A: G=100: H=A: N=A: CLS: PRINT TAB(15)"## FRAGEB ##": PRINT
100 INPUT"Zklen =";Z
110 INPUT"X1, X2 ";X1, X2: XD=(X2-X1)/200
120 INPUT"Y1, Y2 ";Y1, Y2: YD=(Y2-Y1)/240: YG=YD*8
130 INPUT"1: Gebirge  2: Fraktal";A: ON A GOSUB 210, 150: GOTO 130
140 !--------Fraktal-------------
150 Y=Y1-YD: CLS
160 FOR I=0 TO 239: Y=Y+YD: X=X1-XD
170   FOR J=0 TO 199: X=X+XD: A=.5: B=0: H=1
180   GOSUB 30: IF H=INT(H) THEN PSET J,I,7
190 NEXT: NEXT: CALL*406: INPUT"";A: IF A=0 THEN RETURN
200 !--------Gebirge--------------
210 Y=Y1-YG: FOR I=0 TO 320: S(I)=0: NEXT: CLS
220 FOR I=0 TO 120 STEP 4: Y=Y+YG: X=X1-XD
230   FOR J=0 TO 199: X=X+XD: A=.5: B=0: H=Z: Q=S(I+J)
240     IF Q<S(I+J+1) THEN Q=S(I+J+1)
250     Q=Q+1: GOSUB30: IF I+H>Q THEN S(I+J)=I+H: PSET I+J,I+H,7
260     IF J=0 THEN F=H: NEXT: NEXT: GOTO 310
270     IF ABS(F-H)<2 THEN F=H: NEXT: NEXT: GOTO 310
280     A=F: B=H: IF H<F THEN A=H: B=F
290       FOR M=A+1 TO B-1: IF I+M>Q THEN PSET I+J,I+M,7
300 NEXT: F=H: NEXT: NEXT
310 CALL*406: INPUT"";A: IF A=0 THEN RETURN
```



FRAK 87

```
10 CLS: PRINT"## FRAK87 ##": PRINT:A=0: B=A: C=A: D=A: G=50: E=.01
20 DIM A(15,255): X=A: Y=A: INPUT"Zyklen =";Z
30 INPUT"X1, X2 =";X1,X2:XD=(X2-X1)/320
40 INPUT"Y1, Y2 =";Y1,Y2: YD=(Y2-Y1)/256: Y=Y1-YD
50 FOR I=0 TO 255: X=X1-XD: Y=Y+YD: R=-1
60  FOR J=0 TO 319 STEP 20: Q=0: R=R+1
70   FOR K=0 TO 19: X=X+XD: A=X: B=Y: Q=Q+Q
80    FOR N=1 TO Z
90      C=A*A-B*B-X: D=A*B: D=D+D-Y
100     IF ABS(C-A)>E OR ABS(D-B)>E THEN A=C: B=D: NEXT: NEXT: GOTO 140
110     A=C: B=D: IF ABS(A)+ABS(B)<G THEN NEXT: NEXT: GOTO 140
120 IF 2*INT(N/2)<N THEN Q=Q+1
130 NEXT K
140 A(R,I)=Q: NEXT: PRINT I;: NEXT
150 PRINTX1;X2;Y1;Y2;Z;G;E
160 INPUT"Name";A$: CSAVE*""+A$;A
```

FRAK 3

```
0 --- Maschinencode --- ## FRAK3 ## ----
10 WINDOW 0,31,0,39: DIM A(15,255): CLS: PRINT TAB(15);"## FRAK3 ##": PRINT
20 INPUT"Name";A$: CLOAD*""+A$;A: CLS: F=524288
30 FOR X=0 TO 255: D=0
40   FOR Y=0 TO 15: C=A(Y,X): E=F
50     FOR I=0 TO 19: IF C>=E THEN PSET D,X,7: C=-E
60 E=E/2: D=D+1: NEXT:NEXT: NEXT: CALL*406
70 INPUT"";A: IF A<>0 THEN 70
```

FRAKTAL

```
10 WINDOW 0,31,0,39: CLS: PRINT TAB(12);"## FRAKTAL ##": PRINT
20 DIM A(31,319), B(7): A=0: B=A: C=A: D=A: X=A: Y=A: N=A: G=100: E=.01
30 INPUT"Zyklen =";Z
40 INPUT"X1, X2 =";X1,X2: XD=(X2-X1)/320: X=X1-XD
50 INPUT"Y1, Y2 =";Y1,Y2: YD=(Y2-Y1)/256: CLS
60 !--------Bilderzeugung-------
70 FOR I=0 TO 319: Y=Y1-YD: X=X+XD
80  FOR J=0 TO 255 STEP 8
90   FOR K=0 TO 7: Y=Y+YD: A=.5: B=0
100   FOR N=1 TO Z: B(K)=4
110    C=A*A-B*B-X: D=A*B: D=D+D-Y
120    IF ABS(A-C)<E AND ABS(B-D)<E THEN NEXT K: GOTO 160
130    A=C: B=D
140    IF ABS(A) + ABS(B) >G THEN B(K)=N-5*INT(N/5): NEXT K: GOTO 160
150  NEXT: NEXT
160   A=0: FOR K=0 TO 7: A=A*5+B(K): IF B(K)=4 THEN PSET I,J+K,7
170        NEXT: A(J/8,I)=A:!Matrixelemet
180 NEXT: NEXT
190 !---------Eckpunkte---------
200 PSET 0,0,7: PSET 319,0,7: PSET 0,255,7: PSET 319,255,7
210 INPUT"";A:IF A<>0 THEN 210
220 !-------Bildauswertung------
230 B=78125: INPUT"Wert 0-4";X: CLS
240 FOR I=0 TO 319
250  FOR J=0 TO 31: L=8*J: C=B: A=A(J,I)
260   FOR K=0 TO 7
270    D=INT(A/C): A=A-D*C: C=C/5
280    IF D=X THEN PSET I,L+K,7
290 NEXT: NEXT: NEXT: GOTO 200
```

RASTER

```
10 PRINT TAB(15)"## RASTER ##": PRINT
20 RESTORE: FOR X=0 TO 70: READ A$
30 A=ASC(LEFT$(A$,1))-48: IF A>9 THEN A=A-7
40 B=ASC(RIGHT$(A$,1))-48: IF B>9 THEN B=B-7
50 POKE X,A*16+B: NEXT
60 INPUT"Raster";A: CALL*0: GOTO60
70 DATA F5,E5,D5,C5,DB,88,CB,D7,D3,88,21,00,A8,06,10,7E,EE,04,77,23,23
80 DATA 10,F8,11,40,00,19,E5,B7,11,00,B0,ED,52,E1,38,E8,06,40,2B,10,FD
90 DATA 06,04,7E,EE,04,77,23,23,10,F8,11,10,00,19,E5,B7,11,00,B2,ED,52
100 DATA E1,38,E8,C1,D1,E1,F1,C9
```

FUNKTION

```
0 REM: Maschinencode
10 DEFFNA(X)=LN(X)/LN(2):REM::::::::::::::::::::::::::::::::::::::::::::
20 WINDOW 0,31,0,39: CLS: PRINT TAB(10)"### FUNKTION ###": PRINT
30 INPUT"F(X) =";A$: CALL*40A: PRINT
40 PRINT TAB(6);"Tabelle": PRINT: PRINT"X=",A$: PRINT
50 FOR X=1 TO 10: PRINT X, FN A(X): NEXT
```

```
%DISPLAY 400 4DF
0400 00 3D 04 00 00 9C 3A 3A
0408 3A 02 E5 D5 C5 F5 21 62
0410 03 E5 CD DA C4 21 40 04
0418 3E A7 BE 23 20 FC 23 23
0420 23 23 23 EB E1 AF BE 28
0428 06 7E 12 23 13 18 F6 3E
0430 3A 12 13 3E 8E 12 F1 C1
0438 D1 E1 C9 0C 00 88 04 0A
0440 00 94 A7 41 28 58 29 B4
0448 BF 28 58 29 AF BF 28 32
0450 29 3A 8E 3A 3A 3A 3A 3A
0458 3A 3A 3A 3A 3A 3A 3A 3A
0460 3A 3A 3A 3A 3A 3A 3A 3A
0468 3A 3A 3A 3A 3A 3A 3A 3A
0470 3A 3A 3A 3A 3A 3A 3A 3A
0478 3A 3A 3A 3A 3A 3A 3A 3A
0480 3A 3A 3A 3A 3A 3A 3A 00
0488 B8 04 14 00 DD 20 30 2C
0490 33 31 2C 30 2C 33 39 3A
0498 20 99 3A 20 9E 20 A5 31
04A0 30 29 22 23 23 23 20 46
04A8 55 4E 4B 54 49 4F 4E 20
04B0 23 23 23 22 3A 20 9E 00
04B8 D3 04 1E 00 84 22 46 28
04C0 58 29 20 3D 22 3B 41 24
04C8 3A 20 9D AE 34 30 41 3A
04D0 20 9E 00 F7 04 28 00 9E
04D8 20 A5 36 29 3B 22 54 61
```

FNSA

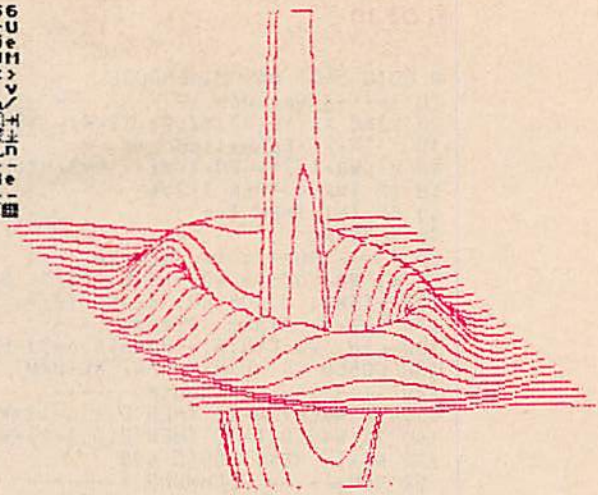
```
0 --- Maschinencode ---- ## FNSA ## ---
10 DEF FN A(X)=X*X:REM::::::::::::::::::::::::::::::::::::::::::::::::
20 DEF FN B(X)=X^2:REM::::::::::::::::::::::::::::::::::::::::::::::::
30 DEF FN C(X)=COS(X):REM::::::::::::::::::::::::::::::::::::::::::::::
40 DEF FN D(X)=TAN(X):REM::::::::::::::::::::::::::::::::::::::::::::::
50 DEF FN E(X)=ATN(X):REM::::::::::::::::::::::::::::::::::::::::::::::
60 B=1032: FOR A=65 TO 69: A$=""
70 PRINT"FN  "CHR$(A)"(X)=";:INPUT"";A$: IF A$>"" THEN POKE B,A: CALL*40A
80 NEXT
```

```
%DISPLAY 400 4DF
0400 00 62 04 00 00 8E 3A 3A
0408 42 02 E5 D5 C5 F5 21 62
0410 03 E5 CD DA C4 21 47 04
0418 3E A7 BE 23 20 FC 23 3A
0420 08 04 BE 23 20 F2 23 23
0428 23 23 EB E1 AF BE 28 06
0430 7E 12 23 13 18 F6 3E 3A
0438 12 13 3E 8E 12 F1 C1 D1
0440 E1 C9 0C 0A 43 6F 70 79
0448 72 69 67 68 74 20 48 2E
0450 56 6F 65 6C 7A 20 44 65
0458 7A 2E 31 39 38 37 0D 0A
0460 0A 00 AB 04 0A 00 94 20
0468 A7 20 41 28 58 29 D4 58
0470 AE 58 3A 8E 8E 3A 3A 3A
0478 3A 3A 3A 3A 3A 3A 3A 3A
            *** FNSA ***
```

PLOT 3D

```
0 GOTO 560: MASCHINENCODE
10 !----- Geraden ----------
20 LINE XS,YS,XZ,YZ,0: XS=XZ: YS=YZ: RETURN
30 !------ Schnittpunkte -----
40 H=(W0-T)/(V+W0-T-W1): S=S+H*(U-S): T=T+H*(V-T)
50 IF T>254 THEN T=254
60 IF T<1 THEN T=1
70 RETURN
80 !---- Ober,unter-Grenze ---
90 W1=W0: GOSUB 40: XS=S: YS=T: RETURN
100 W1=W0: GOSUB 40: XZ=S: YZ=T: GOTO 20
110 !----- algem. Grenzen ---
120 K=H: W1=G(B,A): W0=G(B,A-1)+H*(W1-G(B,A-1))
130 GOSUB 40: H=H+K-H*K: RETURN
140 !------- Horizonte -----
150 IF W>G(1,A-1) THEN G(1,A-1)=W+.5
160 IF W<G(0,A-1) THEN G(0,A-1)=W-.5
170 NEXT: NEXT: GOTO 690
180 REM------ RECHNUNG --------
190 COLOR1,7:::CLS:C=L:REM88AA
200 A1=-.5:REM::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
210 FOR Y=YU TO YO STEP (YO-YU)/L
220 YV=(Y-YU)*YF:C=C-1:A=C:REM18
230 Q=Y*Y:REM+Y*Y):REM::::::::::::::::::::::::::::::::::::::::::::::::::::::::
240 FORX=XUTOXOSTEPXW:A=A+1:REM8
250 Z=SQR(Q+X*X): Z=COS(Z)*EXP(A1*Z):REMREM::::::::::::::::::::::::::::
260 S=U: U=UF+(X-XU)*XF-YV: T=V: W=V: V=(Z-ZU)*ZF+YV
270 !----- Entscheidungen ----
280 G=F: F=2: IF V>G(1,A) AND G(1,A)>G(0,A) THEN F=4
290 IF V>254 THEN F=5
300 IF V<G(1,A) AND V>G(0,A) THEN F=3
310 IF V<1 THEN F=1
320 IF X=XU AND F=2 THEN XS=U: YS=V: NEXT: NEXT: GOTO 690
330 IF X=XU THEN: NEXT: NEXT: GOTO 690
340 H=0: ON G GOTO 350, 450, 500, 520, 400
350 IF F>1 THEN W0=1: GOSUB 90:IF F=2 THEN XZ=U: YZ=V: GOSUB 20
360 IF F>2 THEN B=0: GOSUB 120: XZ=S: YZ=T: GOSUB 20
370 IF F>3 THEN B=1: GOSUB 120: XS=S: YS=T: IF F=4 THEN XZ=U: YZ=V:
380 IF F=5 THEN W0=254: GOSUB 100                              GOSUB 20
390 GOTO 150
400 IF F<5 THEN W0=254: GOSUB 90: IF F=4 THEN XZ=U: YZ=V: GOSUB 20
410 IF F<4 THEN B=1: GOSUB 120: XZ=S: YZ=T: GOSUB 20
420 IF F<3 THEN B=0: GOSUB 120: XS=S: YS=T: IF F=2 THEN XZ=U: YZ=V:
430 IF F<2 THEN W0=1: GOSUB 100                                GOSUB 20
440 GOTO 150
450 IF X=XU+XW AND F<5 AND F<1 THEN XZ=U: YZ=V: GOSUB 20:GOTO 150
460 IF F=2 THEN XZ=U: YZ=V: GOSUB 20: GOTO 150
470 IF X=XU+XW AND F 3 GOTO 520
480 IF F>2 GOTO 360
490 GOTO 430
500 IF F>2 GOTO 370
510 IF F<3 GOTO 420
520 IF F=4 THEN XZ=U: YZ=V: GOSUB 20: GOTO 150
530 IF F=5 GOTO 380
540 GOTO 410
550 !------ Start --------
560 WINDOW 0,31,0,39: COLOR7,0: CLS
570 PRINT TAB(10)"### PLOT3D ###"
580 PRINT"H.VOELZ"TAB(30)"85/87": PRINT: DIM G(1,75)
590 !------- Eingaben --------
600 A$="": INPUT"Def.Konst.A1-AZ";A$: IF A$<>"" THEN CALL*40A
610 A$="": INPUT"Hilfsfkt. Q(Y)=";A$: IF A$<>"" THEN CALL*40F
620 A$="": INPUT"Z(X,Y,Konst.)=";A$: IF A$<>"" THEN CALL*414
630 INPUT"Linienzahl=";L: IF L<2 THEN L=2
640 IF L>25 THEN L=25
650 R=L+L:INPUT"X-Bereich=";XU,XO: UF=106: XF=212/(XO-XU): XW=(XO-XU)/R
660 INPUT"Y-Bereich=";YU,YO: YF=105/(YO-YU)
670 INPUT"Z-Bereich=";ZU,ZO: ZF=170/(ZO-ZU)
680 FOR X=0 TO 75: G(0,X)=999: G(1,X)=-99: NEXT: GOTO 190
690 INPUT"";A$: IF A$<>"0" GOTO 690
700 PRINT XU; XO; YU; YO; ZU; ZO
710 INPUT"";A$: IF A$<>"0" GOTO 690
720 GOTO 600
```

```
%DISPLAY 400 46F
0400  00 4F 04 00 00 88 35 36  _O▒_ ←56
0408  30 02 D5 16 FF 18 08 D5  0↑U▯→↑←U
0410  16 FE 16 03 D5 16 FD E5  ■■→▒U■Ue
0418  C5 F5 21 62 03 E5 D5 CD  Eu!b▒eUM
0420  DA C4 21 54 06 D1 7A BE  ZD!T▒Qz>
0428  23 20 FC 3E FF BE 20 F6  # ö>▯>  v
0430  06 08 23 10 FD E8 E1 AF  ▒←▒^Uk∩/
0438  BE 28 06 7E 12 23 13 18  >(▒~↓▒▯↑
0440  F6 3E 3A 12 13 3E 8E 12  v>:↓▯>▒±
0448  F1 C1 E1 D1 C9 0C 00 6E  qAaQI_n
0450  04 0A 00 9C 2D 2D 2D 2D  ▒+_▯----
0458  2D 20 47 65 72 61 64 65  - Gerade
0460  6E 20 2D 2D 2D 2D 2D 2D  n ------
0468  2D 2D 2D 2D 2D 00 93 04  -----_▯▒
```



COPY

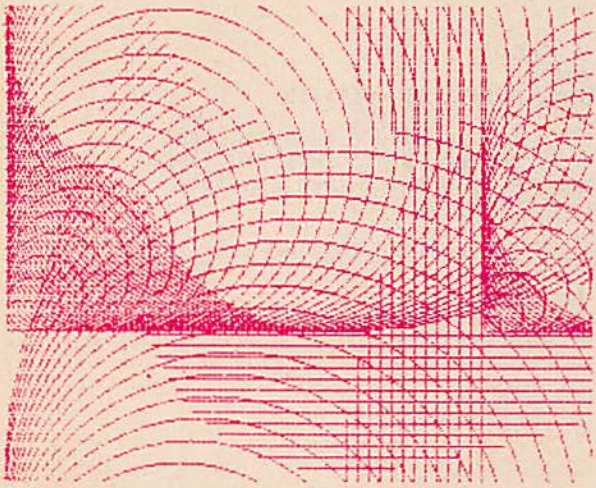
```
10 CLEAR 30, 6140: WINDOW 0,31,0,39: CLS
20 FOR X=3 TO 350 STEP 6
30 CIRCLE X+5, X+80, X, 7
40 NEXT: CALL*400: INPUT"";A
50 !alles fuer 16 K: COPY=CALL*432
60 !32 K: CLEAR 30, 22520: 406: 438
70 !SAVE in CAOS 300: (3D7/8): C08C
80 !Eigene Programme Start:35F/60

!!!!LOAD in CAOS!!!!
```

```
%DISPLAY 400 4DF
0400  D5 11 00 18 18 04 D5 11  U╤_→→▒U╤
0408  00 58 F5 DB 08 CB D7 D3  _XU▒←KWS
0410  88 E5 C5 21 00 80 01 00  ←eE!__u_
0418  28 ED B0 C1 E1 F1 D1 C9  (m0AaqQI
0420  7F 7F 43 4F 31 36 01 18  □□C016u↑
0428  09 7F 7F 43 4F 33 32 01  →□□C032u
0430  18 06 E5 21 00 18 18 04  →▒e!_→→▒
0438  E5 21 00 58 F5 DB 88 CB  e!_XU▒←K
0440  D7 D3 88 C5 D5 11 00 80  WS←EU╤__
0448  01 00 28 ED B0 CD 03 F0  u_(m0M▒P
0450  17 D1 C1 F1 E1 C9 00 76  /QAqqI_v
0458  04 0A 00 A1 20 33 30 2C  ▒+_¡ 30,

%DISPLAY 3D7 3D9
03D7  4E 06 54 06 54 06 56 04  N▒T▒T▒V▒

%DISPLAY 35F 365
035F  57 04 00 9B 00 00 00 00  W▒_%____
```



BASUHR

```
0 REM: Maschinencode
10 DEF FN A(X)=X+6*INT(X/10)
20 CLS: PRINT TAB(10)"## BASUHR ##": PRINT
30 INPUT"1:ein- 2:ausschalten 3:Stoppuhr";A
40 ON A GOSUB60,90,110: GOTO30
50 !------- einschalten -----
60 INPUT"h,m,s =";A, B, C: POKE 257,FN A(A): POKE 258,FN A(B)
70 POKE 259,FN A(C): CALL*406: RETURN
80 !-------- ausschalten -----
90 CALL*497: RETURN
100 !-------- Stoppuhr -------
110 INPUT"Zyklen=";Z: CALL*49E
120 FOR X=1 TO Z: NEXT: CALL*49E: RETURN
130 ! in 257 Stunden: 258 Minuten: 259 Sekunden: 260 1/100 Sekunden
```

```
0400  00 E3 04 00 00 8E F3 F5  _c■_ ■■su
0408  E5 21 1B 04 22 E8 01 3E  e!▒■"h u>
0410  B7 D3 8C 3E 89 D3 8C E1  7S1>+S1a
0418  F1 FB C9 F3 C5 D5 E5 F5  qaIsEUeu
0420  DB 88 CB D7 D3 88 3A 04  ■+KWS+:■
0428  01 C6 02 27 32 04 01 30  uF+'2■u0
0430  5F 3A 03 01 C6 01 27 32  _:■uF'2
0438  03 01 D6 60 20 13 32 03  ■uV` #2■
0440  01 3A 02 01 C6 01 27 32  u:+uF'2
0448  02 01 D6 60 20 13 32 02  +uV` ■2+
0450  01 3A 01 01 C6 01 27 32  u:uuF'2
0458  01 01 D6 24 20 03 32 01  uuV$ ■2u
0460  01 2A A0 B7 22 05 01 AF  u* 7"■■u/
0468  67 2E 19 22 A0 B7 21 01  g.+" 7!u
0470  01 7E 23 CD 03 F0 1C CD  u~#■■p■■■
0478  03 F0 2B 7E 23 CD 03 F0  ■p+~#■■p
0480  1C CD 03 F0 2B 7E CD 03  ■■■p+~■■
0488  F0 1C 2A 05 01 22 A0 B7  p■*■■u" 7
0490  F1 E1 D1 C1 FB ED 4D F5  qaQA■mIu
0498  3E 03 D3 8C F1 C9 F3 F5  >■S+qIsu
04A0  E5 D5 C5 DB 88 CB D7 D3  eUE■■+KWS
04A8  88 21 01 01 05 04 7E CD  +!uu■■~■M
04B0  03 F0 1C CD 03 F0 2B 23  ■p■■■p+#
04B8  10 F4 CD 03 F0 2C C1 D1  ■t■■p,AQ
04C0  E1 F1 FB C9 0C 0A 43 6F  aqaI+Co
04C8  70 79 72 69 67 68 74 20  pyright
04D0  48 2E 56 6F 65 6C 7A 20  H.Voelz
04D8  44 65 7A 2E 31 39 38 37  Dez.1987
04E0  0D 0A 00 FC 04 0A 00 94  ♪◙ u■+■■
```

```
                    ## BASUHR ##   22 41 34
1:ein- 2:ausschalten 3:Stoppuhr 1
h,m,s = 22,40,10
1:ein- 2:ausschalten 3:Stoppuhr 3
Zyklen= 200
22 40 18 44
22 40 18 86
1:ein- 2:ausschalten 3:Stoppuhr 3
Zyklen= 10000
22 40 25 64
22 40 45 42
1:ein- 2:ausschalten 3:Stoppuhr 3
Zyklen= 10
22 40 55 36
22 40 55 42
```

BASUR

```
10 CLS: PRINT TAB(13)"### BASUR ###": PRINT
20 INPUT"1:stellen 2:anzeigen 3:stoppen";A
30 ON A GOSUB 40, 50, 60: GOTO 20
40 INPUT"h,m,s =";A, B, C: POKE 29,A: POKE 30,B: POKE 31,C: RETURN
50 FOR I=1 TO 3: PRINT AT(0,25+3*I);PEEK(28+I): NEXT: GOTO 50
60 INPUT"Zyklen";Z: CALL*F1B8
70 FOR X=1 TO Z: NEXT: CALL*F1B8: RETURN
```

Zum Kapitel 5

BINOM

```
10 CLS: PRINT TAB(9);"*** BINOM ***":PRINT
20 PRINT: INPUT"N,M";N,M: A=N
30 IF M=1 THEN 50
40 FOR X=2 TO M: A=A*(N-X+1)/X: NEXT
50 PRINT"N ueber M =";A: GOTO 20
```

FIBON

```
10 CLS: PRINT TAB(9);"*** FIBON ***": PRINT
20 PRINT: INPUT"Ordnung =";N: IF N>INT(N) OR N<0 THEN 20
30 A=1: B=2: IF N=1 THEN PRINT A: GOTO 20
40 IF N=2 THEN PRINT B: GOTO 20
50 FOR I=3 TO N: C=A+B: A=B: B=C: NEXT
60 PRINT C: GOTO 20
```

GGT

```
10 CLS: PRINT TAB(9);"*** GGT ***": PRINT
20 PRINT: INPUT"N,M";N,M
30 FOR I=0 TO 1 STEP 0
40   IF M=0 THEN PRINT"GGT =";N: GOTO 20
50   IF M>N THEN A=M: M=N: N=A
60   R=N-M*INT(N/M):  N=M: M=R
70 NEXT
```

ACK 1

```
10 CLS: PRINT TAB(9);"*** ACK1 ***": PRINT
20 INPUT"Ymax =";N: DIM A(4,N)
30 !=== Zeile 0 belegen ===
40 FOR Y=0 TO N: A(0,Y)=Y+1: NEXT
50 !=== Umspeicherung fuer Y=0 ===
60 FOR X=1 TO 4: A(X,0)=A(X-1,1)
70   FOR Y=1 TO N-1
80     IF A(X,Y-1)>N THEN NEXT X: GOTO 130:! zu grosses Argument
90     IF A(X-1,A(X,Y-1))=0 THEN NEXT X: GOTO 130:! nicht berechnet
100 '  A(X,Y)=A(X-1,A(X,Y-1))
110 NEXT: NEXT
120 !======= Anzeige =====
130 FOR Y=0 TO N: M=(Y+1)/10: FOR X=0 TO 4
140 PRINT TAB(X*6);A(X,Y);
150 NEXT: PRINT: IF M=INT(M) THEN INPUT"";A
160 NEXT
```

```
          *** ACK1 ***

Ymax = 2100
 1       2       3       5       13
 2       3       5       13      0
 3       4       7       29      0
 4       5       9       61      0
 5       6       11      125     0
 6       7       13      253     0
 7       8       15      509     0
 8       9       17      1021    0
 9       10      19      2045    0
 10      11      21      0       0

 11      12      23      0       0
 12      13      25      0       0
 13      14      27      0       0
 14      15      29      0       0
 15      16      31      0       0
 16      17      33      0       0
 17      18      35      0       0
 18      19      37      0       0
 19      20      39      0       0
 20      21      41      0       0
```

ACK 2

```
10 CLS: PRINT TAB(9);"*** ACK2 ***": PRINT: Q=1
20 INPUT"Ymax =";N: DIM S(N)
30 INPUT"X,Y";S(0),S(1): T=1
40 IF T=0 THEN PRINT: PRINT"Ergebnis =";S(0): GOTO 30
50 IF S(T-1)=0 THEN S(T-1)=S(T)+1:   T=T-1:  GOTO 80
60 IF S(T)=0   THEN S(T-1)=S(T-1)-1: S(T)=1: GOTO 80
70 S(T+1)=S(T)-1: S(T)=S(T-1): S(T-1)=S(T-1)-1: T=T+1
80 FOR X=0 TO T: PRINT S(X);: NEXT: Q=Q+1: PRINT: ! Stackausdruck
90 IF Q=20 THEN INPUT"";A: Q=1
100 GOTO 40
```

```
          *** ACK2 ***

Ymax = 200
X,Y 2,2
 1   2   1
 1   1   2   0
 1   1   1   1
 1   1   0   1   0
 1   1   0   0   1
 1   1   0   2
 1   1   3
 1   0   1   2
 1   0   0   1   1
 1   0   0   0   1   0
 1   0   0   0   0   1
 1   0   0   0   2
 1   0   0   3
 1   0   0   4
 1   5
 0   1   4
 0   0   1   3
 0   0   0   1   2
 0   0   0   0   1   1
 0   0   0   0   0   1   0
 0   0   0   0   0   0   1
 0   0   0   0   0   2
 0   0   0   0   3
 0   0   0   4
 0   0   5
 0   6
 7
```

Zeiten fuer die Berechnung der Ackermann-Funktion

Mit ACK1

bis:	150	1000	2100
Sek:	16	90	180

Mit ACK2
ohne Stackanzeige Zeilen 80,90 gelöscht

x,y:	2,1	2,2	2,3	2,4	2,5	...	2,10	2,15	2,20
Sek:	2	3	3	4	5		12	25	40

x,y:	3,1	3,2	3,3	3,4	3,5
Sek:	5	23	105	440	1900

Zeiten gelten fuer KC 85/3; KC 87 ist schneller

Zum Kapitel 6

POT

```
10 CLS: CLEAR 500: PRINT TAB(10);"## POT ##": PRINT
20 INPUT "F(X) =";A$: L=LEN(A$): B=1
30 FOR I=L TO 1 STEP -1
40  IF MID$(A$,I,1)<>"^" THEN NEXT : GOTO 180
50  A=I: N=VAL(MID$(A$,A+1,1))
60  IF MID$(A$,A-1,1)=")" THEN I=0: NEXT: GOTO 150
70   FOR J=A-1 TO 1 STEP -1: B$= MID$(A$,J,1)
80    IF B$="+" OR B$="*" OR B$="-" OR B$="/" THEN B=J+1: J=1
90   NEXT: I=1: NEXT
100 C$=MID$(A$,B,A-B)+"*": D$=LEFT$(A$,B-1)
110  FOR J=1 TO N: D$=D$+C$: NEXT: D=LEN(D$)
120 D$=LEFT$(D$,D-1)+RIGHT$(A$,L-A-1)
130 PRINT D$: PRINT: GOTO 20
140 REM --- mit Klammer -----
150  FOR J=A-2 TO 1 STEP -1
160   IF MID$(A$,J,1)="(" THEN B=J: J=1: NEXT: GOTO 100
170  NEXT: PRINT"Es fehlt (": GOTO 20
180 PRINT"kein ^gefunden": GOTO 20
```

VIQUA

```
10 PRINT TAB(12);"## VIQUA ##": PRINT
20 DEF FN A(X)=INT (SQR(X))
30 INPUT"Anfang , Ende";A,B
40 FOR X=A TO B: Y=X/4
50  PRINT"###  ";X;"  ###"
60   FOR I=FN A(X) TO FN A(Y) STEP -1
70    X1=X-I*I: IF X1=0 THEN 200
80    Y=X1/3: A=FN A(X1): IF A>I THEN A=I
90     FOR J=A TO FN A(Y) STEP -1
100     X2=X1-J*J: IF X2=0 THEN 190
110     Y=X2/2: A=FN A(X2): IF A>J THEN A=J
120       FOR K=A TO FN A(Y) STEP -1
130         X3=X2-K*K: IF X3=0 THEN 180
140         A=FN A(X3): IF A>K THEN A=K
150           FOR L=A TO 1 STEP-1
160 IF X3<>L*L THEN NEXT: NEXT: NEXT: NEXT: NEXT: END
170 PRINT I;"&";J;"&";K;"&";L: NEXT: NEXT: NEXT: NEXT: NEXT: END
180 PRINT I;"&";J;"&";K: NEXT: NEXT: NEXT: NEXT: END
190   PRINT I;"&";J: NEXT: NEXT: NEXT: END
200  PRINT I: NEXT: NEXT: END
```

```
RUN
###    10    ###
 3 & 1
 2 & 2 & 1 & 1
###    11    ###
 3 & 1 & 1
###    12    ###
 3 & 1 & 1 & 1
 2 & 2 & 2
###    13    ###
 3 & 2
 2 & 2 & 2 & 1
###    14    ###
 3 & 2 & 1
###    15    ###
 3 & 2 & 1 & 1
###    16    ###
 4
 2 & 2 & 2 & 2
###    17    ###
 4 & 1
 3 & 2 & 2
###    18    ###
 4 & 1 & 1
 3 & 3
 3 & 2 & 2 & 1
###    19    ###
 4 & 1 & 1 & 1
BREAK IN 170
OK
```

PRIMEL

```
10 PRINT TAB(10)"**** PRIMEL ****": PRINT: GOTO 70
20 REM ----- TEST PRIMZAHL FUER P, Q=FLAG -------
30 Q=0: FOR X=3 TO SQR(P+1) STEP 2: T=P/X
40 IF T=INT(T) THEN X=P: Q=1
50 NEXT X: RETURN
60 REM ---------- MAIN -------------
70 INPUT"   Ende =";E: M=3E7
80 INPUT"Abstand =";A: IF A/2<>INT(A/2) OR A<=2 GOTO 80
90 PRINT: PRINT"Zahl","Zyklen","Maximum": PRINT
100 FOR Y=3 TO E STEP 2: P=Y: D=Y: Z=0: GOSUB 30
110 IF Q=1 THEN NEXT Y: END
120 P=P*P+A: Z=Z+1: IF P>D THEN D=P
130 IF P>M THEN PRINT Y;"Grenze",Z,P: NEXT Y: END
140 GOSUB 30: IF Q=0 GOTO 120
150 IF Z>1 THEN PRINT Y,Z,D
160 NEXT Y: END
```

DIFER

```
0 --- Maschinencode -- ## DIFER ## ---
10 CLS: PRINT TAB(10);"## DIFER ##": PRINT: GOTO 540
20 REM ---------- Zerlegung -------------
30 Q$="": H$="": P=0: K=0: PR$="4": L=LEN(F$)
40 FOR J=1 TO L: IF MID$(F$,J,1)="X" THEN J=L: NEXT: GOTO 60
50 NEXT: G$="0": RETURN
60 IF F$="X" THEN G$="1": RETURN
70 REM------- Klammerebene ------------
80 FOR I=1 TO L: C$=MID$(F$,I,1)
90   IF C$="(" THEN K=K+1: NEXT: GOTO 140
100  IF C$=")" THEN K=K-1: NEXT: GOTO 140
110  IF K=0 THEN GOSUB 200
120 NEXT
130 REM ------- Termtrennung ------------
140 IF P=0 THEN F$=MID$(F$,2,L-2): L=LEN(F$): GOTO 30
150 L$=LEFT$(F$,P-1): R$=RIGHT$(F$,L-P)
160 IF Q$="^" THEN 500
170 IF Q$=""  THEN 380
180 GOTO 280
190 REM -------- Prioritaeten ----------
200 P$="4": IF C$="^"    THEN P$="3"
210 IF C$="*" OR C$="/" THEN P$="2"
220 IF C$="+" OR C$="-" THEN P$="1"
230 IF C$="="           THEN P$="0"
240 IF P$+PR$="44"      THEN H$=H$+C$: P=I : RETURN
250 IF P$>PR$           THEN RETURN
260 Q$=C$: P=I: PR$=P$: RETURN
270 REM ------- Grundrechnungen --------
280 S=S+1: S$(S)=Q$: S=S+1: S$(S)=L$: S=S+1: S$(S)=R$
290 F$=L$: GOSUB 30: F$=S$(S): S=S+1: S$(S)=G$
300 GOSUB 30: GR$=G$: GL$=S$(S): S=S-1
310 R$=S$(S): S=S-1: L$=S$(S): S=S-1
320 Q$=S$(S): S=S-1
330 IF Q$="+" OR Q$="-" OR Q$="=" THEN G$=GL$+Q$+GR$
340 IF Q$="*" THEN G$="("+GL$+")*"+R$+"+"+L$+"*("+GR$+")"
350 IF Q$="/" THEN G$="(("+GL$+")*"+R$+"-"+L$+"*("+GR$+"))/("+R$+")^2)"
360 RETURN
370 REM --------- Ableitungen ----------
380 S=S+1: S$(S)=H$: S=S+1: S$(S)=R$: F$=R$: GOSUB 30
390 R$=S$(S): S=S-1: H$=S$(S): S=S-1: GR$="": GL$=""
400 IF H$="SQR"    THEN GL$="/2/SQR"
410 IF H$="SIN"    THEN GL$="*COS"
420 IF H$="COS"    THEN GL$="*-SIN"
430 IF H$="TAN"    THEN GL$="/COS": GR$="^2"
440 IF H$="ATN"    THEN GL$="/(1+": GR$="^2"
450 IF H$="LN"     THEN GL$="/"
460 IF H$="EXP"    THEN GL$="*EXP"
470 IF GL$=""      THEN GL$="*"+H$+" "
480 G$="("+G$+")"+GL$+R$+GR$: RETURN
490 REM ---------  A^B  ----------------
500 S=S+1: S$(S)=F$
510 F$="LN("+L$+")*"+R$: GOSUB 30
520 F$=S$(S): S=S-1: G$=F$+"*("+G$+")": RETURN
530 REM -----------Main ----------------
540 CLEAR 5000: DIM S$(100): S=-1
550 INPUT" F(X) =";F$: GOSUB 30
560 PRINT"F'(X) =";G$: PRINT: GOTO 550
```

ABLEI

##ABLEI## Maschinencode

```
%DISPLAY 400 487
0400 00 62 04 00 00 8E 3A 3A
0408 42 02 E5 D5 C5 F5 21 62
0410 03 E5 CD DA C4 21 47 04
0418 3E A7 BE 23 20 FC 23 3A
0420 08 04 BE 23 20 F2 23 23
0428 23 23 EB E1 AF BE 28 06
0430 7E 12 23 13 18 F6 3E 3A
0438 12 13 3E 8E 12 F1 C1 D1
0440 E1 C9 0C 0A 43 6F 70 79
0448 72 69 67 68 74 20 48 2E
0450 56 6F 65 6C 7A 20 44 65
0458 7A 2E 31 39 38 37 0D 0A
0460 0A 00 89 04 0A 00 A1 20
0468 33 30 30 30 3A 20 99 3A
0470 20 9E 20 A5 31 35 29 3B
0478 22 23 23 20 41 42 4C 45
0480 49 20 23 23 22 3A 20 9E
%
```

```
0 ------ Maschinencode ----------- ## ABLEI ## ---
10 CLEAR 3000: CLS: PRINT TAB(15);"## ABLEI ##": PRINT
20 DEF FN A(X)=X:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
30 DEF FN B(X)=X:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
40 M=30: DIM F$(M), FS$(M), FU$(M), FV$(M), F1$(M), O(M)
50 Z$="+-*/^": READ FZ: DIM X$(FZ), XS$(FZ), L(FZ)
60 FOR I=1 TO FZ: READ X$(I), XS$(I): L(I)=LEN(X$(I)): NEXT: GOTO 1190
70 Z=0: POKE 1032,65: INPUT"F(X) ="; F$(0): CALL*40A: GOSUB 110
80 PRINT "F(X) = "; FS$(0)
90 FOR I=1 TO LEN(FS$(0)): POKE 865+I,ASC(MID$(FS$(0),I,1)): NEXT
100 POKE 865+I,0: POKE 1032,66: CALL*40A: PRINT: RETURN
110 F$=F$(Z): S$=F$: GOSUB 1010
120 IF P=1 OR P=L GOTO 990
130 IF P=0 GOTO 760
140 REM====== u und v ======
150 O(Z)=O: FU$(Z)=LEFT$(F$,P-1): FV$(Z)= MID$(F$,P+1)
160 Z=Z+1: F$(Z)=FU$(Z-1): GOSUB 110: F1$(Z-1)=FS$(Z)
170 F$(Z)=FV$(Z-1): GOSUB 110: Z=Z-1
180 FU$=FU$(Z): FV$=FV$(Z): F1$=F1$(Z): F2$=FS$(Z+1): O=O(Z)
190 ON O GOTO 210, 210, 260, 260, 450
200 REM====== f=u+v usw. ========
210 IF F1$<>"0" AND F2$<>"0" THEN FS$(Z)=F1$+MID$(Z$,O,1)+F2$: RETURN
220 IF F1$<>"0" THENFS$(Z)=F1$: RETURN
230 IF F2$<>"0" THENFS$(Z)=MID$(Z$,2,O-1)+F2$: RETURN
240 FS$(Z)="0": RETURN
250 REM======== f=u*v usw. ========
260 IF F1$="0" THEN T=0: FS$=F1$: GOTO 310
270 T=1: IF F1$="1" THEN FS$=FV$: GOTO 310
280 S$=F1$: GOSUB 1010: IF O<3 THEN F1$="("+F1$+")"
290 FF$=FV$: S$=FF$: GOSUB 1010: IF O<3 THEN FF$="("+FF$+")"
300 FS$=F1$+"*"+FF$: GOTO 310
310 IF F2$="0" GOTO 390
320 FF$=F1$: S$=F1$: GOSUB 1010: IF O<3 THEN FF$="("+FF$+")"
330 T=T+1
340 FS$=MID$(FS$,1-(T=1))+MID$(Z$,O(Z)-2,1+((T=1) AND O(Z)=3))
350 IF F2$="1" THENF S$=FS$+FU$: GOTO 390
360 S$=FU$: GOSUB 1010: IF O<3 THEN FU$="("+FU$+")"
370 S$=F2$: GOSUB 1010: IF O<3 THEN F2$="("+F2$+")"
380 FS$=FS$+FU$+"*"+F2$
390 IF FS$="0" OR O(Z)=3 GOTO 430
400 IF T=2 THEN FS$="("+FS$+")"
```

```
410 S$=FV$: GOSUB 1010: IF O<3 THEN FV$="("+FV$+")"
420 FS$=FS$+"/"+FV$+"^2"
430 FS$(Z)=FS$: RETURN
440 REM======= f=u'v usw. ========
450 IF F2$<>"0" GOTO 650
460 IF F1$="0"  THEN FS$(Z)=F1$: RETURN
470 IF F1$="1"  THEN FS$="": GOTO 500
480 S$=F1$: GOSUB 1010: IF O<3 THEN F1$="("+F1$+")"
490 FS$=F1$+"*"
500 IF FV$<>"1" THEN FS$=FV$+"*"+FS$
510 IF LEFT$(FV$,1)="(" THEN FV$=MID$(FV$,2)
520 F2$=STR$(VAL(FV$)): IF F2$=" 0" GOTO 590
530 IF LEFT$(F2$,1)=" " THEN F2$=MID$(F2$,2): GOTO 530
540 IF F2$=FV$ THEN F2$=STR$(VAL(F2$)-1)
550 IF F2$<>FV$ THEN 590
560 IF LEFT$(F2$,1)=" " THEN F2$=MID$(F2$,2): GOTO 560
570 IF LEFT$(F2$,1)="-" THEN 610
580 IF LEFT$(F2$,1)<>"-" THEN 620
590 F2$=FV$: IF RIGHT$(F2$,1)=")" THEN F2$=LEFT$(F2$,LEN(F2$)-1)
600 F2$=F2$+"-1"
610 F2$="("+F2$+")"
620 IF F2$="1" THEN FS$(Z)=FS$+FU$: RETURN
630 IF F2$="0" THEN FS$(Z)=FS$+"1": RETURN
640 FS$(Z)=FS$+FU$+"^"+F2$: RETURN
650 FF$=FU$
660 S$=FF$: GOSUB 1010: IF O<3 THEN FF$="("+FF$+")"
670 S$=F1$: GOSUB 1010: IF O<3 THEN F1$="("+F1$+")"
680 S$=FV$: GOSUB 1010: IF O<3 THEN FV$="("+FV$+")"
690 S$=F2$: GOSUB 1010: IF O<3 THEN F2$="("+F2$+")"
700 FS$=F$(Z)+"*(": IF F1$<>"0" THEN FS$=FS$+F1$+"/"+FF$+"*"+FV$+"+"
710 FS$=FS$+"LN": IF LEFT$(FU$,1)<>"(" THEN FU$="("+FU$+")"
720 F$=F$+FU$
730 IF F2$="1" THEN FS$(Z)=FS$+")": RETURN
740 FS$(Z)=FS$+"*"+F2$+")": RETURN
750 REM====== Funktionssuche ======
760 FOR I=1 TO FZ: IF LEFT$(F$,L(I))<>X$(I) THEN NEXT: GOTO 940
770 II=I: I=FZ: NEXT: IF RIGHT$(F$,1)<>")" GOTO 990
780 REM====== Berechnen von u ======
790 O(Z)=II
800 Z=Z+1: F$(Z)=MID$(F$,L(II)+1,L-L(II)-1): GOSUB 110: Z=Z-1
810 F$=F$(Z): FU$=F$(Z+1): F1$=FS$(Z+1)
820 IF F1$="0" THEN FS$(Z)="0": RETURN
830 S$=F1$: GOSUB 1010: IF O<3 THEN F1$="("+F1$+")"
840 O=O(Z): IF O>10 GOTO 900
850 REM==== f'=u'*g(u) ==========
860 FS$=XS$(O)+FU$+")": IF F1$<>"1" THEN FS$=F1$+"*"+FS$
870 IF O>5 THEN FS$=FS$+"^2"
880 FS$(Z)=FS$: RETURN
890 REM===== f'=u'/g(u) ==========
900 FS$=XS$(O)+FU$+")": FS$=F1$+"/"+FS$
910 IF O<>19 THEN FS$=FS$+"^2)"
920 FS$(Z)=FS$: RETURN
930 REM====== Klammern =======
940 IF LEFT$(F$,1)<>"(" OR RIGHT$(F$,1)<>")" GOTO 970
950 S$=MID$(F$,2,LEN(F$)-2): GOSUB 1110: IF N THEN FS$(Z)="0": RETURN
960 F$(Z)=MID$(F$,2,LEN(F$)-2): GOTO 110
970 IF F$="X" THEN FS$(Z)="1": RETURN
980 FS$(Z)="0": RETURN
990 PRINT"? SYNTAX ERROR in":PRINT"? "F$: PRINT: GOTO 70
1000 REM====== Suche Rechenzeichen ========
1010 L=LEN(S$): P=0: O=6: KL=0
1020 FOR I=1 TO L: K$=MID$(S$,I,1): KL=KL+(K$="("): KL=KL-(K$=")")
1030 IF KL<>0 THEN NEXT: GOTO 1080
1040 IF K$="+" OR K$="-" THEN P=I: O=1-(K$="-")
```

```
1050 IF K$="*" OR K$="/" THEN IF O>2 THEN P=I: O=3-(K$="/")
1060 IF K$="^" THEN IF O>4 THEN P=I: O=5
1070 NEXT
1080 IF KL GOTO 990
1090 RETURN
1100 REM====== Term = Zahl ? ======
1110 FOR I=1 TO LEN(S$): K$=MID$(S$,I,1)
1120 IF K$>"/" THEN IF K$<":" THEN NEXT: N=1: RETURN
1130 IF K$="-" THEN NEXT: N=1: RETURN
1140 N=0: RETURN
1150 REM===== Funktionen =======
1160 DATA 7, SIN(, COS(, COS(, (-1)*SIN(,EXP(,EXP(,LN(,1/(
1170 DATA SQR(,.5/SQR(, TAN(,1/COS(,ATN(,1/1+(
1180 !========= MAIN =========
1190 INPUT"1:ableiten 2:rechnen";A: ON A GOSUB 70, 1200: GOTO 1190
1200 INPUT"X1, X2, N =";X1,X2,N
1210 PRINT: PRINT"  X","F(X)","F'(X)"
1220 FOR X=X1 TO X2 STEP (X2-X1)/N
1230 PRINT X,FN A(X),FN B(X)
1240 NEXT: PRINT: RETURN
1220 FOR X=X1
```

Zum Kapitel 7

WAGLA

```
10 CLS: PRINT TAB(10);"## WAGLA ##": PRINT: F$="nicht zulaessig"
20 PRINT"mit 3 Glaesern soll Zielmenge": PRINT"    erreicht werden"
30 INPUT"Zielmenge =";Z
40 INPUT"grosses Glas =";G: Q=G-Z: IF G<Z THEN PRINT F$: GOTO 40
50 INPUT"kleines Glas =";K: IF K>Z THEN PRINT F$: GOTO 50
60 INPUT"letztes Glas =";L: IF L>K THEN PRINT F$: GOTO 60
70 FOR I=0 TO 10: M=K*I
80   FOR J=0 TO 10: N=L*J
90    IF Z=M+N THEN 140
100   IF Z=M-N THEN 150
110   IF Q=M+N THEN 160
120   IF Q=M-N THEN 170
130 NEXT: NEXT: PRINT"Es gibt keine einfache Loesung !": GOTO 30
140 PRINT Z"="I"*"K"+"J"*"L: GOTO 30
150 PRINT Z"="I"*"K"-"J"*"L: GOTO 30
160 PRINT Z"="G"-"I"*"K"-"J"*"L: GOTO 30
170 PRINT Z"="G"-"I"*"K"+"J"*"L: GOTO 30
```

DAMEN

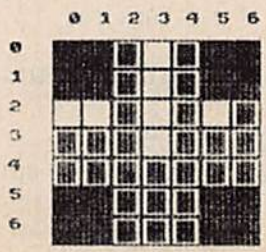
```
10 CLS: PRINT TAB(9);"## DAMEN ##": PRINT: DIM A(30): Z=1
20 INPUT"Wieviele Damen";D: IF D<4 OR D>19 OR D>INT(D) THEN 20
30 REM ---- Rechnung ------
40 FOR X=1 TO 2 STEP 0
50   IF A(Z)=D THEN 140
60   A(Z)=A(Z)+1: IF Z=1 THEN 120
70   Y=1
80    FOR I=Y TO Z-1
90     IF A(I)=A(Z) THEN Y=I: NEXT X
100    IF ABS(A(I)-A(Z))=ABS(I-Z) THEN Y=I: NEXT X
110   NEXT
120  IF Z=D THEN 170
130  Z=Z+1: NEXT
140  A(Z)=0: Z=Z-1: IF Z=0 THEN END
150 NEXT
160 REM ---- Anzeige ------
170 Q=Q+1: PRINT Q"-te Variation": PRINT
180 FOR X=1 TO D
190   FOR Y=1 TO D
200    IF A(Y)=X THEN PRINT"D ";
210    IF A(Y)<>X THEN PRINT"* ";
220 NEXT: PRINT: NEXT: PRINT: PRINT: GOTO 40
```

SOLET

```
10 WINDOW: CLS: DIM A(6,6), B(100): PRINT TAB(14);"## SOLET ##":
20 REM --------- Wertberechnungen --------
30 Y=INT(K/10): X=INT(K)-10*Y: RETURN
40 !--- Zerlegen der Eingabe ------
50 M=(K-INT(K))*10: Y1=INT(M): X1=INT((M-Y1)*10+.5)
60 REM ------ Setzroutine ------------
70 A(X,Y)=Z: H=X+X+10: S=Y+Y+21: IF R THEN 120
80 IF Z=0 THEN A=128: B=129: C=130: D=131
90 IF Z=1 THEN A=132: B=133: C=134: D=135
100 IF Z=2 THEN A=136: B=A: C=A: D=A
110 GOTO 150
120 IF Z=0 THEN A=193: B=137: C=136: D=200
130 IF Z=1 THEN A=178: B=179: C=177: D=176
140 IF Z=2 THEN A=255: B=A: C=A: D=A
150 PRINT AT(H,S);CHR$(C): PRINT AT(H-1,S);CHR$(A)
160 PRINT AT(H,S+1);CHR$(D): PRINT AT(H-1,S+1);CHR$(B): RETURN
170 IF Q THEN PRINT"nicht mehr moeglich": GOTO 380
180 PRINT"nicht erlaubt": GOTO 380
190 REM ---- Setzen der Parameter ------
200 PRINT: R=0: IF PEEK(-5)>129 THEN R=8: GOTO 230
210 FOR I=0 TO 71: READ A: POKE I,A: NEXT
220 VPOKE 14252,0: VPOKE 14253,0
230 FOR X=0 TO 6: PRINT AT(X+X+9,17);X:PRINT AT(7,X+X+21);X: NEXT
240 REM ------ verbieten ------
250 Z=2: FOR X=0 TO 1: FOR Y=0 TO 1: GOSUB 70: NEXT
260 FOR Y=5 TO 6: GOSUB 70: NEXT: NEXT
270 FOR X=5 TO 6: FOR Y=0 TO 1: GOSUB 70: NEXT
280 FOR Y=5 TO 6: GOSUB 70: NEXT: NEXT
290 REM ----- voll besetzen ------
300 Z=1: L=0: FOR Y=2 TO 4: FOR X=0 TO 6: GOSUB 70: NEXT: NEXT
310 FOR X=2 TO 4: FOR Y=0 TO 1: GOSUB 70: NEXT
320 FOR Y=5 TO 6: GOSUB 70: NEXT: NEXT
330 REM ------ Mitte leer ------
340 Z=0: X=3: Y=3: GOSUB 70
350 REM ------- Start ---------
360 PRINT"Felder als Spalte*Zeile: Ist.Ziel,": PRINT"z.B. 31.33"
370 PRINT"0: zurueck  8: vor": WINDOW 7,31-R,0,15
380 INPUT"";K: Q=0
390 GOSUB 30: IF K<=0 THEN 520
400 IF K=8 THEN Q=1: GOTO 560
410 IF X>6 OR Y>6 THEN 170
420 IF A(X,Y)<>1 THEN 170
430 GOSUB 50: IF X1>6 OR X2>6 THEN 170
440 IF A(X1,Y1) OR A((X+X1)/2,(Y+Y1)/2)<>1 THEN 170
450 IF NOT(X=X1 OR Y=Y1) THEN 170
460 IF Y=Y1 AND ABS(X-X1)<>2 THEN 170
470 IF X=X1 AND ABS(Y-Y1)<>2 THEN 170
480 Z=0: GOSUB 70
490 IF X=X1 THEN Y=(Y+Y1)/2: GOSUB 70
500 IF Y=Y1 THEN X=(X+X1)/2: GOSUB 70
510 Z=1: X=X1: Y=Y1: GOSUB 70: B(L)=K: L=L+1: GOTO 380
520 IF L=0 THEN PRINT"Anfang": GOTO 380
530 L=L-1: K=B(L): GOSUB 30: GOSUB 50: Z=1: GOSUB 70
540 X=(X+X1)/2: Y=(Y+Y1)/2: GOSUB 70
550 Z=0: X=X1: Y=Y1: GOSUB 70: GOTO 380
560 IF B(L)=0 THEN PRINT"Ende": GOTO 380
570 K=B(L): GOTO 390
580 DATA 255,128,128,128,128,128,128,128,255,1,1,1,1,1,1,1
590 DATA 128,128,1 8,128,128,128,128,255,1,1,1,1,1,1,1,255,255
600 DATA 128,159,159,159,159,159,159,255,1,249,249,249,249,249
610 DATA 249,159,159,159,159,159,159,128,255,249,249,249,249
620 DATA 249,249,1,255,255,255,255,255,255,255,255,255
```

```
            ## SOLET ##
Felder als Spalte*Zeile: Ist.Ziel,
z.B. 31.33 eingeben
0: zurueck  8: vor

 31.33
 52.32
 33.31
 30.32
 22.24
nicht erlaubt
 22.42
 2.22
```



LNIMM

```
10 WINDOW 0,23,0,39: CLS: PRINT TAB(13);"## L...   #": PRINT
20 DIM A(35,5), B(35), C(35),A$(1)
30 A$(0)="Rechner": A$(1)="Automat"
40 FOR I=1 TO 35: FOR K=1 TO 5: A(I,K)=128: NEXT: NEXT
50 WINDOW 3,12,0,39: S=35: Z=Z+1: T=T+1: IF T<TO THEN 100
60 PAUSE 10: CLS: INPUT"wie viele Spiele automatisch:";TO: T=0
70 IF TO=0 THEN CLS: GOTO 320
80 INPUT"Anzeige der Lernmatrix 0/1";SM
90 CLS: PRINT A$(0);" gegen ";A$(1)
100 IF SM THEN GOSUB 410
110 WINDOW 5,12,0,39: PRINT"Spiel";Z
120 PRINT"35  31    25    19    13    7     1"
130 PRINT"+----+-----+-----+-----+-----+-----+"
140 PRINT CHR$(2);: N=0: V=0: C=INT(2*RND(1))
150 IF S=1 THEN A=1: GOTO 200
160 L=0: FOR I=1 TO 5: L=L+A(S,I): NEXT
170 R=INT(RND(1)*L): A=0: B=0
180 A=A+1: B=B+A(S,A): IF B<R THEN 180
190 IF S-A<0 THEN 170
200 N=N+1: B(N)=A: C(N)=S: S=S-A: IF S=0 THEN 260
210 IF V<>1 THEN 240
220 IF C=0 THEN PRINT"Ich nehme davon weg   ";A: GOTO 320
230 C=0: GOTO 150
240 PRINT TAB(34-S);CHR$(82-17*C);
250 C=ABS(SGN(C)-1): GOTO 150
260 Q=.5: FOR I=N TO 1 STEP -1
270 IF A(C(I),B(I))*Q=.5 OR A(C(I),B(I))*Q=16384 THEN 290
280 A(C(I),B(I))=A(C(I),B(I))*Q
290 Q=1/Q: C(I)=0: B(I)=0
300 NEXT: PRINT: PRINT: PRINT"verloren hat ";A$(C)
310 V=0: GOTO 50
320 IF S<>1 THEN 340
330 V=0: PRINT"Sie haben verloren": GOTO 50
340 V=1: PRINT
350 PRINT"Der Stapel betraegt jetzt ";S
360 INPUT"Wieviel nehmen Sie (1-5) :";A
370 IF A<1 OR A>5 OR A<INT(A) THEN 360
380 IF NOT(S<6 AND S>1 AND S-A=0) THEN 400
390 PRINT"Das darf doch nicht wahr sein !": GOTO 350
400 C=1: GOTO 200
410 WINDOW 15,23,0,39: FOR J=1 TO 5: FOR I=35 TO 1 STEP -1
420 :Y=53+INT(A(I,J)/10): IF Y<33 THEN Y=33
430 IF Y>127 THEN Y=127
440 PRINTCHR$(Y);: NEXT: PRINT: NEXT: RETURN
450 ! Bei KC 85/1 und KC 87 Semikolon in Zeile 240 am Ende entfer-
    nen
```

Zum Kapitel 8

EINGABE

```
10 DIM A$(3): A$(1)="ASCII-Zeichen": A$(2)="Dez-Werte": A$(3)="HEX_W."
20 CLS: PRINT TAB(10);"## EINGABE ##": PRINT
30 PRINT: INPUT"Anzahl der gueltigen Stellen =";A: CLS
40 PRINT TAB(8);"----------------------"
50 PRINT TAB(8);"!   ASCII-Zeichen     !"
60 PRINT TAB(8);"!   Dez-Werte         !"
70 PRINT TAB(8);"!   HEX-Werte         !"
80 PRINT TAB(8);"----------------------"
90 PRINT: INPUT"Ihre Wahl =";B$: L=LEN (B$)
100 IF A<L THEN B$=LEFT$(B$,A): L=A
110 FOR I=1 TO 3
120   IF B$ = LEFT$(A$(I),L) GOTO 140
130 NEXT
140 CLS: ON I GOSUB 160, 230, 300, 370: GOTO 20
150 REM ----- ASCII-Zeichen ----------
160 PRINT"  Es sind Zahlen zwischen 33 und 127"
170 PRINT"    einzugeben. Die zugehoerenden"
180 PRINT"   ASCII-Zeichen werden angezeigt."
190 PRINT" andere Zahlen fuehren zum Hauptmenu": PRINT
200 INPUT"Zahl     ";B: IF B<33 OR B>127 THEN RETURN
210 PRINT"Zeichen  ";CHR$(B): GOTO 200
220 REM ------- Dez-Werte ------------
230 PRINT"   Es sind ASCII-Zeichen einzugeben"
240 PRINT" dazu werden die dezimalen Zahlen"
250 PRINT" angezeigt.  Mehr als ein Zeichen"
```

```
260 PRINT"   fuehrt zum Hauptmenu zurueck": PRINT
270 INPUT"Zeichen  ";A$: IF LEN (A$)>1 THEN RETURN
280 PRINT"dezimal ";ASC(A$): GOTO 270
290 REM -------- HEX-Werte -----------
300 PRINT"   Es sind ASCII-Zeichen einzugeben"
310 PRINT"   dazu werden die HEX-Werte gezeigt"
320 PRINT"mehr als 1 Zeichen fuehrt zum Hauptmenu": PRINT
330 INPUT"Zeichen  ";A$: IF LEN (A$)>1 THEN RETURN
340 J=ASC(A$): I=J AND 15: J=INT(J/16)
350 PRINT"HEX-Wert ";CHR$(J+48-7*(J 9));CHR$(I+48-7*(I 9)): GOTO330
360 REM ----- Fehler -----------------
370 PRINT"   **** fehlerhafte Eingabe ****": BEEP(3): PAUSE(10):RETURN
```

MENU

```
10 WINDOW 1,20,0,39: CLS: PRINT TAB(13);"## MENU ##": PRINT
20 DEF FN A(X)=(X AND 15) - 9*(X>64)
30 DEF FN B(X)=X+48 -7*(X>9)
40 GOTO 210
50 !----- Verteiler -----------
60 WINDOW 10,12,0,39: INK 4: PAPER 3: CLS
70 ON I GOSUB 90, 140, 340: GOTO 210
80 !----- hex->dez ---------
90 INPUT"HEX =";A$: X=0
100 FOR I=1 TO LEN(A$)
110   X=X*16 + FN A(ASC(MID$(A$,I,1)))
120 NEXT: PRINT"DEZ =";X: RETURN
130 !----- dez->hex ---------
140 INPUT"DEZ =";N: A$="": J=N: PRINT"HEX = ";: IF J>4095 THEN GOSUB160: J=N
150 GOSUB 170: PRINT A$: RETURN
160 I=INT(N/4096): J=N-4096*I: N=I
170 FOR X=1 TO 3
180  I= J AND 15: J=INT(J/16): A$=CHR$(FN B(I)) + A$
190 NEXT: RETURN
200 !----- Main ------------
210 WINDOW 3,5,10,30
220 PRINT"#  HEX nach DEZ": I=1
230 PRINT"   DEZ nach HEX"
240 PRINT"      Ende";
250 WINDOW 17,20,0,39:INK 3: PAPER 4: CLS
260 PRINT"Eingabe: Hoch; Tief; Bestaetigen"
270 WINDOW 19,19,10,14: INK 8: PAPER 7: CLS
280 INPUT"";A$: IF A$<>"H" AND A$<>"T" AND A$<>"B" GOTO 280
290 IF A$="B" GOTO 60
300 PRINT AT(I+2,10);" "
310 IF A$="H" THEN I=I-1: IF I=0 THEN I=1
320 IF A$="T" THEN I=I+1: IF I>3 THEN I=3
330 PRINT AT(I+2,10);"#": GOTO 280
340 WINDOW: CLS: END
```

ZEICHEN

```
10 CLS: PRINT TAB(14);"## ZEICHEN ##": PRINT
20 PRINT"Erzeugung der Zeichen"
30 PRINT"bitte einen Moment warten"
40 FOR I=0 TO 7
50 FOR J=0 TO 16 STEP 8: B=0: READ A$
60 FOR X=1 TO 8: B=B+B: A=0: IF MID$(A$,X,1)="Ä" THEN A=1
70 B=B+A: NEXT: POKE I+J,B: NEXT: NEXT
80 VPOKE 14252,0: VPOKE 14253,0
90 REM --- ABLAGE DER ZEICHEN ---
100 DATA ÄÄ....ÄÄ, ....ÄÄÄ., Ä.Ä.Ä.Ä.              Ä ≙ ■
110 DATA .ÄÄ..ÄÄ., ..ÄÄÄ..., .Ä.Ä.Ä.Ä
120 DATA ..ÄÄÄÄ.., ÄÄÄ....., Ä.Ä.Ä.Ä.
130 DATA . .ÄÄ..., ..ÄÄÄ..., .Ä.Ä.Ä.Ä
140 DATA ÄÄ....ÄÄ, ....ÄÄÄ., Ä.Ä.Ä.Ä.
150 DATA .ÄÄ..ÄÄ., ........, .Ä.Ä.Ä.Ä
160 DATA ..ÄÄÄÄ.., .ÄÄÄÄÄÄ., Ä.Ä.Ä.Ä.
170 DATA ...ÄÄ..., .ÄÄÄÄÄÄ., .Ä.Ä.Ä.Ä
180 PRINT"Darstellung der Zeichen": PRINT
190 FOR J=1 TO 3: PRINT"  ";
200 FOR I=128 TO 130: PRINT" "CHR$(I);: NEXT
210 NEXT
```

INKOR

```
10 CLS: PRINT TAB(10);"## INKOR ##": PRINT: F$="EINGABE: ":A$="123"
20 PRINT F$;A$;: FOR I=0 TO LEN(A$): PRINT CHR$(8);: NEXT
30 A=LEN(F$): B=CSRLIN(1): INPUT"";A$: LOCATE B,A: A$=""
40 FOR I=A TO 40: A$=A$+VGET$: PRINT CHR$(9);: NEXT: A=1: B=40
50 FOR I=1 TO LEN(A$): C=MID$(A$,I,1) " "
60 IF C AND I<B THEN B=I:!erstes echtes Zeichen
70 IF C AND I>A THEN A=I:!letztes echtes Zeichen
80 NEXT: A$=MID$(A$,B,A-B+1)
90 LOCATE 10,1: PRINT CHR$(2);A$:LOCATE 2,0: GOTO 20
100 ! Zeile 90 nur als Test
```

Zum Kapitel 9

CLEAR

```
10 CLS: PRINT TAB(10);"## CLEAR ##": PRINT: CLEAR
20 C=65536: B=DEEK(854): R=DEEK(944)
30 PRINT"Stringraum =";R-B;"Bytes"
40 A=R: PRINT"RAM-Ende =";A;"Dez. = ";: IF A<0 THEN A=A+C
50 D=C: FOR I=1 TO 4 : D=D/16
60 B=INT(A/D): A=A-B*D: B=B+48: IF B>57 THEN B=B+7
70 A$=A$+CHR$(B)
80 NEXT:PRINTA$;" HEX"
90 PRINT: INPUT"neuer Stringraum =";E
100 INPUT"neues RAM-Ende in HEX =";A$: A=0
110 IF LEN(A$)>4 THEN PRINT"zu gross": GOTO100
120 FOR I=1 TO LEN(A$)
130 B=ASC(MID$(A$,I,1))-48: IF B>9 THEN B=B-7
140 A=A*16+B
150 NEXT: IF A>C/2 THEN A=A-C
160 CLEAR E,A: GOTO20
```

DATEN

```
10 CLS: PRINT TAB(10)"## DATEN ##": PRINT
20 A=B=C: I=0
30 B=DEEK(987): A=DEEK(854)-100-B
40 PRINT"maximal";A;"Werte moeglich"
50 PRINT:INPUT"1=Eingabe  2=Anzeige";C
60 ON C GOSUB 70, 120: GOTO 50
70 FOR I=0 TO A
80   PRINT"NR";I;: INPUT"";C
90   IFC<0 OR C>254 THEN C=255
100   POKE B+I,C: IF C=255 THEN I=A
110 NEXT: RETURN
120 FOR I=0 TO A: C=PEEK(B+I)
130   PRINT C;: IF C=255 THEN I=A
140 NEXT: PRINT: RETURN
```

LED

```
10 CLS: PRINT TAB(15);"## LED ##"
20 C=136: A=INP(C)
30 FOR I=1 TO 20
40 OUT C, A OR 32: PAUSE 2
50 OUT C, A: PAUSE 2
60 NEXT
```

SPRICH

```
10 CLS: PRINT TAB(10);"## SPRICH ##": PRINT
20 PRINT"bitte einen Moment warten": PRINT
30 ! ---- Sprichwortzahl -  C ----
40 A=DEEK(1025)
50 FOR I=0 TO 1 STEP 0
60   B=DEEK(A): IF B THEN C=A: A=B: ELSE I=1
70 NEXT: C=(DEEK(C+2)-190)/20
80 ! -- Adresse nach RESTORE --
90 FOR I=1125 TO 5000
100   A=PEEK(I): IF A=139 THEN A=I: I=5000
110 NEXT
120 FOR K=0 TO 1 STEP 0
130 !------  ANZEIGE ------
140   FOR J=0 TO 10 STEP 10: L=20*INT(RND(1)*C)+200+J
150     FOR I=2 TO 4
160       POKE A+I ,ASC(MID$(STR$(L),I,1))
170     NEXT
180    RESTORE 250: READ A$: PRINTA$
190 NEXT: INPUT"";A$: NEXT
200 DATA Wenn der Hahn kraeht auf dem Mist
210 DATA aendert sich das Wetter - oder es bleibt - wies ist
220 DATA Wenn auch der Kuhschwanz wackelt
230 DATA so faellt er doch nicht ab
240 DATA Ehe man einmal abschneidet
250 DATA muss man zweimal messen
260 DATA Wenn der Abt zum Glase greift
270 DATA so greifen die Moenche zum Kruge
280 DATA Allen Leuten recht getan
290 DATA ist eine Kunst - die keiner kann
300 DATA Wer einmal luegt - dem glaubt man nicht
310 DATA dreist wenn er die Wahrheit spricht
```

BASGO

```
0  REM  ---- Maschinencode ----


Copyright H.Voelz  Dez. 87

10 PRINT TAB(10);"## BASGO ##"
20 INPUT"1: GOTO 2: GOSUB";A
30 ON A GOTO 50, 90
40 !--- GOTO ---
50 INPUT"GOTO Zeile ";X
60 DOKE 772,1030
70 Y=USR(X)
80 !--- GOSUB ---
90 INPUT"GOSUB Zeile ";X
100 DOKE 772,1043
110 Y=USR(X): GOTO 90
120 !--- Hilfszeilen ---
130 PRINT"Zeile 130"
140 PRINT"Zeile 140"
150 PRINT"Zeile 150": RETURN
160 PRINT"Zeile 160"
170 PRINT"Zeile 170"
180 PRINT"Zeile 180": GOTO 50
OK
```

```
>             ## BASGO ##
%DISPLAY 400 467
0400 00 4E 04 00 00 9C D1 D1  _N█_ ▢QQ
0408 D1 CD 6F C9 21 54 C8 E5  QMoI!THe
0410 C3 0A CA CD 6F C9 E1 E1  C↓JMoIaa
0418 C1 DB 88 CB D7 D3 88 0E  A█←KWS←▒
0420 03 CD 27 C3 E5 E5 2A 58  ▒M'Cee*X
0428 03 E3 3E 8C F5 33 18 DC  ▒c>1u3→I
0430 0C 0A 43 6F 70 79 72 69  1↓Copyri
0438 67 68 74 20 48 2E 56 6F  ght H.Vo
0440 65 6C 7A 20 20 44 65 7A  elz  Dez
0448 2E 20 38 37 0A 00 67 04  . 87↓_g▒
0450 0A 00 9E 20 A5 31 30 29  ↓_▢ %10)
0458 3B 22 23 23 20 42 41 53  ;"## BAS
0460 47 4F 20 23 23 22 00 81  GO ##"_u
%
```

```
                ;==========================*
                ; BASIC:  GOSUB A; GOTO A  *
                ; ##        BASGO       ##  *
                ; H.Voelz          Dez. 87 *
                ;==========================*
                ;
                            ORG     406H
                ;
C96F            EIN         EQU     0C96FH
CA0A            GOTO        EQU     0CA0AH   ; Unterprogr.GOTO
C854            RUNMOD      EQU     0C854H   ; RUN-Mode
C327            TEMO        EQU     0C327H   ; Fehlertest
0358            AKZEI       EQU       358H   ; aktuelle Zeile
008C            TGS         EQU        8CH   ; Token GOSUB
                ;
'0406  D1       JUMP        POP     DE       ; Stack
'0407  D1                   POP     DE       ; korrigieren
'0408  D1                   POP     DE
'0409  CD6FC9               CALL    EIN      ; USR- DE
'040C  2154C8   JPR         LD      HL,RUNMOD
'040F  E5                   PUSH    HL       ;auf Stack
'0410  C30ACA               JP      GOTO
                ;
'0413  CD6FC9   GOSUB       CALL    EIN
'0416  E1                   POP     HL
'0417  E1                   POP     HL
'0418  C1                   POP     BC
'0419  DB88                 IN      A,(88H) ; IMR AN
'041B  CBD7                 SET     2,A
'041D  D388                 OUT     (88H),A
'041F  0E03                 LD      C,3
'0421  CD27C3               CALL    TEMO
'0424  E5                   PUSH    HL
'0425  E5                   PUSH    HL
'0426  2A5803               LD      HL,(AKZEI)
'0429  E3                   EX      (SP),HL
'042A  3EBC                 LD      A,TGS
'042C  F5                   PUSH    AF
'042D  33                   INC     SP
'042E  18DC                 JR      JPR-$
                ;
```

STAMZA

```
10 CLS: PRINT TAB(12);"## STAMZA ##": PRINT
20 N=100: DIM A(N), B(3): A(0)=1: Z=1E38
30 PRINT: INPUT"1:Eingabe 2:Anzeige 3:Rechnen";A: CLS
40 ON A GOSUB 60, 110, 160: GOTO 30
50 REM---- EINGABE -------
60 PRINT"Eingabe der Daten": PRINT"Ende nur ENTER ": PRINT
70 FOR I=A(0) TO N
80  B=Z: INPUT B: IF B=Z THEN A(0)=I-1: I=N: NEXT: RETURN
90  A(I)=B: NEXT: A(0)=N: RETURN
100 REM--- ANZEIGE --------
110 PRINT"Nr.","Wert": A=20
120 FOR I=1 TO A(0)
130  PRINT I, A(I): A=A-1: IF A THEN NEXT: RETURN
140  INPUT"";A: A=20: GOTO 130
150 REM----- RECHNEN ----
160 FOR X=0 TO 3: B(X)=0: NEXT
170 FOR I=1 TO A(0): B=1: A=A(I)
180  FOR X=0 TO 3: B=B*A: B(X)=B(X)+B
190 NEXT: NEXT
200 FOR X=0 TO 3: B(X)=B(X)/A(0): NEXT
210 A=B(0): B=A*A: C=B*B
220 B(3)=B(3)-4*A*B(2)+6*B(1)*B-3*C
230 B(2)=B(2)-3*A*B(1)+2*A*B
240 B(1)=B(1)-B
250 PRINT"Name","Wert","Moment-Nr.": PRINT
260 PRINT"Mittelwert",B(0),"1"
270 PRINT"Streuung",B(1),"2"
280 PRINT"Stand.Abwchg.",SQR(B(1)*A(0)/(A(0)-1))
290 PRINT,B(2),"3": PRINT,B(3),"4"
300 PRINT"Schiefe",B(2)/B(1)/SQR(B(1))
310 PRINT"Steilheit",B(3)/B(1)/B(1)
320 PRINT"=Kurtosis": PRINT: RETURN
```

KORRE

```
10 CLS: PRINT TAB(12);"## KORRE ##": PRINT
20 Z=1E15: NN=100: Z2=1/Z
30 DIM W(1,NN), A(3), B(3), C(3), A$(3), B$(1): B$(0)="X": B$(1)="Y"
40 A$(0)="Y=A*X": A$(1)="Y=A/X": A$(2)="Y=A^X": A$(3)="Y=X^A"
50 REM--- START ----
60 PRINT: INPUT"1:Eingabe 2:Anzeige 3:Werte 4:X - Y";A
70 CLS: ON A GOSUB 90, 140, 200, 390: GOTO 60
80 !----- EINGABE -----
90 FOR X=N TO NN: PRINT X;: FOR Y=0 TO 1
100  A=Z: PRINT B$(Y);: INPUT"=";A: IF A=Z THEN 120
110 W(Y,X)=A: NEXT: NEXT: N=NN: RETURN
120 N=X: X=NN: Y=1: NEXT: NEXT: RETURN
130 !----- ANZEIGE -----
140 PRINT:PRINT"Nr.","X","Y": A=20
150 FOR I=0 TO N-1
160  PRINT I,W(0,I),W(1,I): A=A-1
170  IF A=0 THEN INPUT"";A: A=20
180 NEXT: RETURN
190 REM--- FIT -----
200 FORI=0 TO 3: A(I)=0: B(I)=0: C(I)=1: NEXT
210 FOR I=0 TO N-1
220  X=W(0,I): Y=W(1,I): IF ABS(X)<Z2 THEN X=Z2
230  IF Y<Z2 THEN C(2)=0: C(3)=0
240  IF X<=Z2 THEN C(3)=0
250  A=Y/X: A(0)=A(0)+A: B(0)=B(0)+A*A
260  A=Y*X: A(1)=A(1)+A: B(1)=B(1)+A*A
270  IF C(2) THEN A=LN(Y)/X: A(2)=A(2)+A: B(2)=B(2)+A*A
280  IF C(3) THEN A=LN(X): IF ABS(A)<Z2 THEN A=Z2
290  IF C(3) THEN A=LN(Y)/A: A(3)=A(3)+A: B(3)=B(3)+A*A
300 NEXT
310 PRINT"Funktion","Faktor","Stand.Abwchg."
320 FOR I=0 TO 3
330  A(I)=A(I)/N: B(I)=SQR((B(I)/N-A(I)*A(I))/(N+1))
340  IF C(I) THEN PRINT A$(I),A(I),B(I)
350 NEXT: IF C(2) THEN C(2)=EXP(A(2))
360 IF C(3) THEN C(3)=1/A(3)
370 RETURN
380 !---- REGRESSION -----
390 PRINT TAB(10);"Regression"
```

```
400 FOR I=0 TO 3: PRINTI+1;":";A$(I): NEXT
410 A=Z:INPUT A: IF A<1 OR A>4 THEN RETURN
420 IF C(A-1)=0 THEN PRINT "nicht zulaessig": GOTO 410
430 B=Z: INPUT"1: X- Y  2: Y- X";B: IF B<1 OR B>4 THEN 400
440 IF B=2 GOTO 510
450 X=Z: INPUT"X =";X: IF X=Z THEN 430
460 IF A=1 THEN PRINTA(0)*X
470 IF A=2 THEN PRINTA(1)/X
480 IF A=3 THEN PRINT C(2)^X
490 IF A=4 THEN PRINT X^A(3)
500 GOTO 450
510 Y=Z: INPUT"Y =";Y: IF Y=Z THEN 430
520 IF A=1 THEN PRINT Y/A(0)
530 IF A=2 THEN PRINT A(1)/Y
540 IF A=3 THEN PRINT LN(Y)/A(2)
550 IF A=4 THEN PRINT Y^C(3)
560 GOTO 510
```

FIT

```
10 CLS: PRINT TAB(10);"## FIT ##": NN=50: Z=1E38: Z2=1E-30: GOTO530
20 X=LN(X)
30 Y=LN(Y)
40 P=P+X: Q=Q+Y: R=R+X*X: S=S+Y*Y: T=T+X*Y: RETURN
50 X=1/X: GOTO 40
60 X=LN(X): GOTO 40
70 X=1/X
80 Y=C(J)+B(J)*X: RETURN
90 Y=C(J)+B(J)*LN(X): RETURN
100 Y=C(J)*B(J)^X: RETURN
110 Y=C(J)*X^B(J): RETURN
120 X=(Y-C(J))/B(J): RETURN
130 X=B(J)/(Y-C(J)): RETURN
140 X=EXP((Y-C(J))/B(J)): RETURN
150 X=LN(Y/C(J))/LN(B(J)): RETURN
160 X=(Y/C(J))^(1/B(J)): RETURN
170 !======= FIT =========
180 PRINT: PRINT"Zwei-Parameter-Regression"
190 PRINT:PRINT"Es existieren folgende Funktionen"
200 INPUT"1:a+bx 2:a+b/x 3:a+blnx 4:ab^x 5:ax^b";F
210 P=0: R=0: S=0: T=0: Q=0: FOR I=0 TO N-1: X=A(0,I): Y=A(1,I)
220 ON F GOSUB 40, 50, 60, 30, 20: NEXT
230 R=N*R-P*P: S=N*S-Q*Q: T=N*T-P*Q: P=P/N: Q=Q/N: B(0)=T/R: B(1)=S/T
240 IF ABS(T)<Z2 THEN T=Z2
250 S=(S-R)/(2*T): B(2)=S+SGN(B(0))*SQR(S*S+1)
260 FOR I=0 TO 2: C(I)=Q-P*B(I)
270 NEXT:IF F=4 OR F=5 THEN FOR I=0 TO 2: C(I)=EXP(C(I)): NEXT
280 IF F=4 THEN FOR I=0 TO 2: B(I)=EXP(B(I)): NEXT
290 PRINT: PRINT"Korrelation R^2 ="B(0)/B(1): PRINT
300 PRINT"Es gibt 3 Schwankungen um"
310 PRINT"1:x","2:y","3:x und y"
320 PRINT: PRINT"   a=":FOR I=0 TO 2: PRINTC(I),: NEXT: PRINT
330 PRINT"   b=":FOR I=0 TO 2: PRINTB(I),: NEXT: PRINT
340 !==== REGRESSION =====
350 INPUT"Welchen Fall zur Regression nutzen";J:J=J-1
360 G=3: INPUT"1: x- y  2: y- x 3: Abbruch";G
370 ON G GOTO 380, 400, 180
380 X=Z: INPUT" X=";X: IF X=Z GOTO 360
390 ON F GOSUB 80, 70, 90, 100, 110: PRINT"y="Y:GOTO 380
400 Y=Z: INPUT"  Y=";Y: IF Y=Z GOTO 360
410 ON F GOSUB 120, 130, 140, 150, 160: PRINT"   X=";X: GOTO 400
420 !===== EINGABE ========
430 FOR X=N TO NN: PRINT X;: FOR Y=0 TO 1
440 A=Z: PRINT A$(Y);: INPUT"=";A: IF A=Z THEN 460
450 A(Y,X)=A: NEXT: NEXT: N=NN: RETURN
460 N=X: X=NN: Y=1: NEXT: NEXT: RETURN
470 !===== ANZEIGE ========
480 A=20: PRINT "Nr.","X","Y"
490 FOR X=0 TO N-1: PRINT X,A(0,X), A(1,X): A=A-1
500 IF A=0 THEN INPUT"";A: A=20
510 NEXT: RETURN
520 !====== MAIN ========
530 DIM A(1,NN), B(2), C(2), A$(1): A$(0)="X": A$(1)="   Y"
540 PRINT: INPUT"1:Eingeben 2:Anzeigen 3: Fitten";A
550 CLS: ON A GOSUB 430, 480, 180: GOTO 540
```

Programmregister

Name	Kapitel	Inhalt
ABLEI	6	Komfortable Ableitung
ACK1	5	Ackermann-Funktion (rekursiv)
ACK2	5	Ackermann-Funktion mit Stack
BASGO	9	Zeilensprung über Variable
BASUHR***	4	BASIC mit Uhr und Zeitmessung
BASUR!!*	4	BASIC-Uhr KC 87
BINOM	5	Binominal-Koeffizient
BUB2	1	Bubble mit Pointern, verkürzt
CLEAR***	9	Test der CLEAR-Anweisung
COPY*	4	Bildschirmcopy !!LOAD!!
DAMEN	7	Damen auf Schachbrett
DATEN	9	viele Daten mit POKE eingeben
DIFER	6	einfache formale Ableitung
EINGABE	8	Eingabe mit Länge gültiger Zeichen
EULER	2	lange Eulersche Zahl
FAKUL	2	Fakultät nach Stirling
FIBON	5	Fibonacci-Zahlen
FILE	1	erzeugt Zufallszahlen, -texte
FIT	10	Zweiparameter-Anpassung
FNSA	4	bis 26 Fkt. direkt eingeben
FRAGEB*	3	Fraktale als Gebirge und normal
FRAK3*	3	Fraktale: Felder von FRAK 87
FRAK 87**	3	Felder für Fraktale KC 87
FRAKT*	3	Fraktale (s/w) hoher Auflösung
FRAKTAL*	3	farbige Fraktale
FUNKTION	4	direkte Fkt.-Eingabe und Tabelle
GGT	5	größter gemeinsamer Teiler
INKOR*	8	Eingabe mit Korrektur
KOMBA	2	alle kombinatorischen Formeln
KORRE	10	Korrelation bei zwei Werten
LED*	9	Steuerung der Tape-Diode
LNIMM	7	lernendes Nimm
MENU	8	Eingabe über Zeile
PI	2	viele Ziffern von PI
PLOT3D*	4	3D-Perspektive von Fkt.
POT	6	formale Umrechnung bei Potenzen
PRIMEL***	6	mehrfache Primzahlprodukte
PUFOR	2	Formatieren auf den Punkt
QUICK	1	Quicksort (binär)
RASTER*	4	Raster-Hintergrund m. Farbbyte
REFOR	2	rechtsseitiges Formatieren
RUND	2	Runden von Zahlen
SHAKER	1	Shakersort (hin und her)
SOLET	7	Soletude = Solohalma
SPRICH	9	Fleddern von Sprichwörtern
STAMZA	10	Statistische Maßzahlen
TEFRA	3	Testen für Fraktale
VIQUA	6	vier Quadratzahlen
WAGLA	7	Wasserglasproblem
WOBUB	1	Wortbubble
ZABUB	1	Zahlenbubble
ZEICHEN**	8	Definition von Zeichen

Die Programme sind in der Regel auf den Rechnern KC 85/1...4 und KC 87 lauffähig. Mit * bezeichnete Programme sind nur für den KC 85/2...4, mit ** bezeichnete für den KC 87 und den KC 85/1 geeignet. Mit *** gekennzeichnete Programme werden im Text nicht ausdrücklich erwähnt.