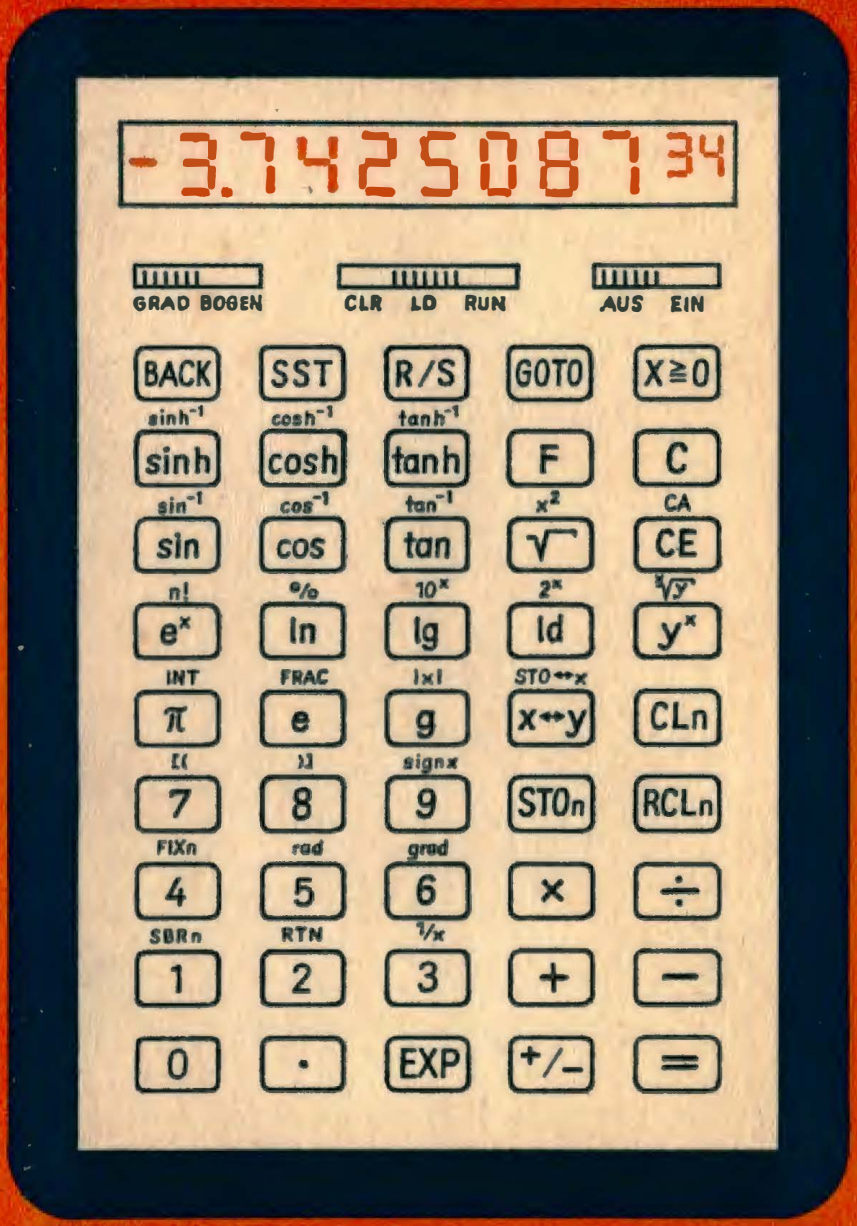
Programmierbare Taschenrechner

Kreul



Programmierbare Taschenrechner

von Dr.-Ing. Hans Kreul

Mit 26 Bildern
und einer Beilage

VEB FACHBUCHVERLAG LEIPZIG

1. Auflage
Lizenznummer 114-210/34/80
LSV 1003
Verlagslektor: Helga Fage
Printed in GDR
Satz und Druck: Messedruck Leipzig, BT Borsdorf III-18-328
Redaktionsschluß: 15. 3. 1980
Bestellnummer: 5465290
DDR 9,80 M

Vorwort

Über die Leistungsfähigkeit moderner elektronischer Datenverarbeitungsanlagen viele Worte zu verlieren ist müßig, denn es ist allgemein bekannt, mit welcher Geschwindigkeit und Exaktheit sie in der Lage sind, komplizierte und umfangreiche wissenschaftliche und technische Probleme zu lösen, daß man aber auch mit ihrer Hilfe riesige Datenmengen, wie sie beispielsweise in der Materialwirtschaft und bei der Lohnabrechnung in Großbetrieben, bei der Kontenführung in Banken und Sparkassen, bei statistischen Untersuchungen u.a.m. auftreten, bewältigen kann.
Es gibt aber eine Vielzahl „kleinerer" Probleme, zu deren Lösung man ohne weiteres elektronische Datenverarbeitungsanlagen einsetzen könnte, die aber nach wie vor mit Bleistift und Papier gerechnet werden, wie man es von jeher tat, weil viele Wissenschaftler, Lehrer, Studenten, Ingenieure und Ökonomen noch eine große Scheu vor der Nutzung moderner Rechenanlagen haben. Für diese Scheu gibt es mancherlei Gründe:

- große Rechenanlagen stehen einem nicht zu jeder Zeit und an jedem Ort zur Verfügung,
- sie sind sehr teuer, und die benötigte Rechenzeit kann recht kostspielig werden,
- viele Mitarbeiter in Rechenzentren verbreiten um sich den Nimbus, als könnten nur sie allein sinnvoll mit derartigen Anlagen rechnen, und für Außenstehende sei der Umgang mit Computern ein Buch mit sieben Siegeln,
- diese Auffassung wird unterstützt durch eine Vielzahl von Vorschriften und formalen Regeln, die beachtet werden müssen, wenn man eine Aufgabe in einem Rechenzentrum abarbeiten lassen möchte,
- man muß spezielle Programmiersprachen erlernen, um sein Problem in einer dem Computer verständlichen Sprache formulieren zu können,
- und schließlich fühlt man sich der Öffentlichkeit preis-

gegeben, wenn das erarbeitete Programm wegen eines Programmierfehlers oder wegen sonstiger Verstöße gegen eine der vielen zu beachtenden Regeln und Vorschriften vom Rechner nicht abgearbeitet wird.

Diese Schwierigkeiten werden sich in absehbarer Zeit für nicht zu umfangreiche technische und wissenschaftliche Berechnungen beheben lassen, denn die Produktion und damit auch der Einsatz von programmierbaren elektronischen Tisch- und Taschenrechnern wird stark ansteigen, und es kann vorausgesagt werden, daß der Tag nicht mehr fern ist, an dem jeder seinen Minicomputer haben wird, mit dem er seine kleineren Probleme in aller Ruhe an seinem Schreibtisch bearbeiten kann. Mit seiner Hilfe wird er dann auch Verständnis gewinnen für die Arbeitsweise und die Anforderungen der großen Rechenanlagen, und es wird ihm nicht mehr schwerfallen, die oben erwähnte Scheu vor der modernen Rechentechnik abzulegen und sich bei der Lösung komplizierterer und umfangreicherer Aufgaben schließlich auch dieser Technik zu bedienen.

Dem Verlag sei an dieser Stelle Dank gesagt für den Mut, der Erarbeitung einer Broschüre über programmierbare elektronische Taschenrechner zu einem Zeitpunkt zuzustimmen, an dem die Anzahl der Rechengeräte noch gering war. Es bleibt zu hoffen, daß dieses Büchlein allen denen von Nutzen sein möge, die einen programmierbaren Taschenrechner besitzen oder erwerben wollen, und denen es an einer ausführlichen und leicht verständlichen Anleitung zum Programmieren mangelt. Besonderer Dank gebührt auch den Herren Dipl.-Phys. DENKMANN und Dr.-Ing. WARKO, die durch ihre kritischen Hinweise und Anregungen wesentlich zur Vervollkommnung des Manuskripts beigetragen haben. Erwähnt sei auch Fräulein stud. ing.-oec. KAFFNER, die die Zeichnungsvorlagen mit großer Sorgfalt angefertigt hat.

Eine Erstauflage eines Buches wird nie vollkommen frei sein von Mängeln und Unzulänglichkeiten. Daher richten wir an die Leser, die das Buch intensiv und kritisch durcharbeiten, die Bitte, uns ihre Erfahrungen bei der Arbeit mit dem Buch, ihre Kritiken und Verbesserungsvorschläge sowie die Hinweise auf möglicherweise vorhandene Druckfehler mitzuteilen, damit diese Anregungen in den nachfolgenden Auflagen berücksichtigt werden können.

Autor und Verlag

Inhaltsverzeichnis

Vorbemerkungen

Das vorliegende Buch stellt den Versuch dar, *einen Laien auf dem Gebiet der Rechentechnik soweit mit der Arbeitsweise programmierbarer Tisch- und Taschenrechner sowie mit den Grundlagen der Programmierung vertraut zu machen*, daß er nach dem Durcharbeiten in der Lage sein müßte, kleinere Programme für ständig wiederkehrende Rechnungen aus seinem Arbeitsgebiet selbständig aufzustellen.

Das Ziel dieses Buches besteht also nicht etwa darin, möglichst viele ausgereifte Rechenprogramme für die unterschiedlichsten Anwendungsfälle und für die verschiedenartigsten Rechnermodelle bereitzustellen, sondern es soll in den folgenden Kapiteln so ausführlich und so verständlich wie möglich dargestellt werden, *wie* man zu derartigen Programmen kommen kann. Aus diesem Grund kann diese Schrift auch für denjenigen von Nutzen sein, der die Herangehensweise an eine Aufgabenstellung kennenlernen möchte, die mit Hilfe einer größeren Rechenanlage bearbeitet werden soll.

Sollte die gewählte Darstellungsweise dem einen oder dem anderen Leser, der bereits Erfahrungen auf dem Gebiet der Programmierung besitzt, zu breit und zu ausführlich erscheinen, so möge er daran denken, daß diese Einführung für einen sehr breiten Leserkreis gedacht ist, und daß aus diesem Grund so gut wie *keine Vorkenntnisse auf dem Gebiet der Algorithmierung und der Programmierung vorausgesetzt* werden sollen. Die hier erarbeiteten Programmablaufpläne und Programme stellen auch nicht in jedem Fall die „eleganteste" Lösung des jeweiligen Problems dar. Es kam vielmehr bei der Erarbeitung des Manuskripts darauf an, die einzelnen Lösungsmethoden so auszuwählen und den Weg bis zum vollständigen Programm so darzustellen, daß alle Einzelschritte vor allem *für den Anfänger verständlich und gut überschaubar* werden, so daß er sie bei ähnlich gelagerten Beispielen vari-

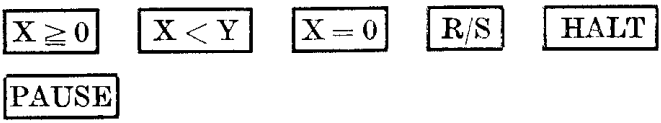
ieren und – mit eigenen neuen Gedanken versehen – auch nachvollziehen kann.
Derjenige, der sich bemüht, jeden kleinsten Schritt bei den vorgeführten Beispielen zu verstehen, wird sehr schnell erfahren, daß das *Programmieren erlernbar* ist, und er wird dann auch sehr bald selbst kleinere, einfache Programme aufstellen können. Und mit der wachsenden Zahl der selbständig erarbeiteten Programme wird er Erfahrungen im Programmieren sammeln, und eines Tages wird er dann auch für dieses oder für jenes Problem eine sehr elegante Lösung anbieten können.
Die Vielzahl der auf dem Markt befindlichen Modelle programmierbarer elektronischer Tisch- und Taschenrechner und deren ständige technische Vervollkommnung haben uns veranlaßt, für die Erarbeitung der Programme zwei Rechner zugrunde zu legen, die es in Wirklichkeit gar nicht gibt: den KREULOTRON 5 mit einer algebraischen Rechenlogik und den KREULOTRON 6, der auf der Basis der Umgekehrten Polnischen Notation arbeiten soll. An die Rechenlogik dieser beiden Rechner wurden bewußt keine besonderen Anforderungen gestellt; so wird beispielsweise nicht vorausgesetzt, daß die Rechner eine Konstantenautomatik besitzen. Das hat zur Folge, daß sich manche arithmetische Ausdrücke bei einigen handelsüblichen Rechnern mit wesentlich weniger Eingaben realisieren lassen, als dies in den hier aufgestellten Programmen der Fall ist. Es hat aber andererseits den Vorteil, daß die erarbeiteten Programme sich ohne wesentliche Veränderungen fast auf jedem Rechnertyp abarbeiten lassen.
Der Leser wird sehr bald erkennen, daß das Eintippen eines Programms in den Rechner eigentlich nur die *letzte Stufe* einer Reihe von wichtigen Vorüberlegungen und Vorarbeiten ist, die unabhängig vom verwendeten Rechnermodell ausgeführt werden müssen. *Je gründlicher man sich Gedanken macht über den zur Lösung der vorliegenden Aufgabenstellung zu verwendenden Algorithmus und darüber, wie man diesen Algorithmus am besten in einzelne Rechenschritte umsetzen kann, um so besser wird man zu einem lauffähigen Rechenprogramm kommen.*

1. Besonderheiten programmierbarer elektronischer Taschenrechner

Wir wollen voraussetzen, daß der Leser, der dieses Büchlein erworben hat, bereits sicher mit herkömmlichen elektronischen Taschenrechnern umgehen kann und daß er schon viele Aufgaben mit Hilfe eines solchen kleinen Rechenknechtes gelöst hat. Sollte dies nicht der Fall sein, so sei ihm dringend empfohlen, zu den im Literaturverzeichnis angegebenen ausführlichen Anleitungen für das Rechnen mit elektronischen Taschenrechnern [6] oder [7] zu greifen und sie gründlich durchzuarbeiten, bevor er sich mit dem hier dargebotenen Stoff beschäftigt.
Rein äußerlich unterscheiden sich programmierbare Taschenrechner kaum von den herkömmlichen Geräten ohne Programmiermöglichkeit. Es fällt lediglich auf, daß zusätzlich zu den auf den herkömmlichen Rechnern auch schon vorhandenen Tasten und Schaltern noch einige weitere Tasten und Schalter angebracht sind. So ist häufig ein Schiebeschalter mit zwei Einstellmöglichkeiten

oder mit drei Einstellmöglichkeiten

vorhanden, und es treten Tasten mit Bezeichnungen wie z.B.



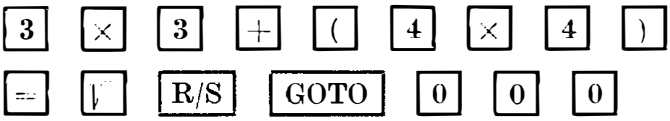
auf, die wir an einem Rechner ohne Programmiermöglichkeit nicht finden.
Programmierbare Taschenrechner lassen sich zunächst ein-

mal in genau der gleichen Weise benutzen wie die herkömmlichen Taschenrechner, wenn man nur den Schiebeschalter dazu in die Stellung RUN bringt. Dies ist, wie wir noch sehen werden, ein großer Vorteil, denn es ist nicht immer sinnvoll, jede kleine Aufgabe erst zu „programmieren", um sie danach automatisch ablaufen zu lassen.

Worin bestehen nun aber die Besonderheiten der programmierbaren Taschenrechner, die ja bekanntlich wesentlich teurer sind als die gewöhnlichen Typen und von denen man demzufolge dann auch zusätzliche Leistungen erwartet?

Wir wollen versuchen, uns diese Besonderheiten an einem ganz einfachen Beispiel klarzumachen.

Nehmen wir an, wir hätten den Rechner gerade erst eingeschaltet und den Schiebeschalter in die Stellung LOAD bzw. LD gebracht und wir würden danach die Tasten



in der angegebenen Reihenfolge betätigen. Der programmierbare Rechner würde sich dann in seinem *Programmspeicher*, den er zusätzlich zu den in jedem anderen Rechner auch vorhandenen Speichern besitzt, folgenden Rechenvorgang „merken":

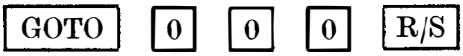
> Bilde zunächst das Produkt $3 \cdot 3$ und danach das Produkt $4 \cdot 4$ und addiere diese beiden Produkte. Aus dem Ergebnis ist dann die Quadratwurzel zu ziehen, und dann soll ein HALT erfolgen, damit das Endergebnis der Rechnung in der Zahlenausgabe abgelesen werden kann. Danach soll der gleiche Rechengang wieder von vorn begonnen werden. (Das Unterbrechen des Rechenvorganges wird durch die Taste R/S bewirkt und der Befehl, wieder von vorn zu beginnen, durch die Tastenfolge GOTO 0 0 0).

Stellt man nun den Schiebeschalter auf die Stellung RUN und drückt die R/S-Taste, so beginnt der Rechner, die oben angegebene Befehlsfolge *automatisch*, also *ohne irgendeinen weiteren Eingriff durch den Menschen*, abzuarbeiten.

Er rechnet also nun völlig selbständig die Aufgabe

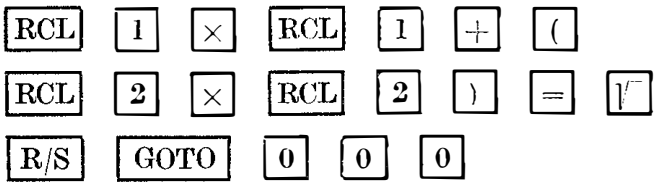
$$\sqrt{3^2 + 4^2} = 5$$

aus und teilt uns dieses Ergebnis mit, indem er an der Stelle, an der er die Rechnung beendet, anhält. Daß der Rechner arbeitet, erkennt man daran, daß in der Zahlenanzeige in sehr schneller Abfolge immer neue Zahlen aufleuchten und wieder erlöschen, bis schließlich das Ergebnis 5 stehenbleibt. Man kann jetzt, solange der Rechner nicht abgeschaltet wird, diese einfache Rechnung beliebig oft wiederholen lassen, indem man die Tastenfolge



drückt; jedesmal wird die oben genannte Aufgabe von neuem berechnet. Schaltet man allerdings den Rechner aus und versucht man nach einiger Zeit, wenn man ihn wieder benutzen will, durch die zuletzt genannte Tastenfolge den Rechengang für die Berechnung von $\sqrt{3^2 + 4^2} = 5$ erneut auszulösen, so wird der Rechner nichts mehr tun, denn er hat durch das Ausschalten „vergessen", was er tun sollte. Die früher eingegebene Befehlsfolge ist durch das Ausschalten des Rechners im Befehlsspeicher gelöscht worden. (Neuerdings gibt es aber auch schon Rechner, die derartige Befehlsfolgen auch über das Abschalten hinaus im Befehlsspeicher aufbewahren. Man nennt derartige Geräte Rechner mit einem *continuous memory*.)

Nun wird es sicher niemanden geben, der sich mehrmals in kurzen Abständen nacheinander vom Rechner bestätigen lassen wird, daß $\sqrt{3^2 + 4^2} = 5$ ist. Ändert man jedoch die ursprünglich eingegebene Befehlsfolge nur ganz geringfügig ab:



so kann der Rechner nun mit diesem „Programm" *jede* Aufgabe vom Typ

$$\sqrt{a^2 + b^2} = c$$

berechnen, wenn man nur vor dem Start des Rechenprogramms dafür sorgt, daß in den beiden Speichern 1 und 2 die Zahlen a und b untergebracht werden.
Dieses verbesserte Programm hat gegenüber der ersten Fassung den Vorteil, daß man den Wert des Ausdruckes

$$c = \sqrt{a^2 + b^2}$$

für jede beliebige Kombination für a und b berechnen lassen kann. Muß man also im Verlauf einer kürzeren Zeit mehrfach Ausdrücke dieser Form berechnen, so braucht man nicht mehr wie bei einem gewöhnlichen Taschenrechner immer wieder für jede neue Zahlenkombination für a und b die Tastenfolge

[a] [×] [a] [+] [(] [b] [×] [b]
[)] [=] [√‾]

zu betätigen, sondern man braucht nur noch die beiden Zahlen a und b in die Speicher 1 und 2 einzugeben und danach die [R/S]-Taste zu drücken. Alles übrige besorgt dann der Rechner selbständig.
Diese Vorgehensweise hat nur noch einen kleinen Schönheitsfehler: der Anwender dieses Programms muß nämlich wissen, daß die Zahlen a und b in ganz bestimmten Speicherzellen untergebracht werden müssen, wenn man erwartet, daß das Programm ordnungsgemäß laufen soll. Würde man nämlich die beiden Ausgangszahlen a und b nicht in den Speichern 1 und 2, sondern in anderen Speichern unterbringen, so würde der Rechner ein falsches Ergebnis liefern.
Auch diese sicher nicht bedeutende geistige Belastung kann man dem Menschen noch abnehmen und dem Taschenrechner übertragen, indem man der letzten Fassung des Programms noch die Tastenfolge

[R/S] [STO] [1] [R/S] [STO] [2]

voranstellt. Durch diese Tastenfolge wird der Rechner nämlich veranlaßt, anzuhalten, um die Zahl a aufzunehmen und im Speicher 1 unterzubringen, dann nochmals anzuhalten, um die Zahl b im Speicher 2 zu speichern. Danach kann dann

das Programm wie oben geschildert ablaufen. Es ist bei dieser endgültigen Form des Programms also nur noch nötig, die beiden Zahlen a und b einzugeben und nach jeder Eingabe die R/S-Taste zu drücken. Mehr wird nun vom Menschen nicht mehr verlangt, alles übrige wird dann vom Rechner für ihn erledigt.

Aus diesem einführenden Beispiel erkennen wir schon, daß es nicht sinnvoll ist, für jede kleine Aufgabe ein eigenes Programm aufzustellen, um sie danach automatisch abarbeiten zu lassen. *Die Programmierarbeit lohnt sich nur dann, wenn es sich um Aufgaben handelt, die mehrfach mit abgeänderten Eingangsparametern durchgerechnet werden müssen, und wenn bei ihnen auch ein erhöhter Rechenaufwand erforderlich ist.*

So würde es sich beispielsweise rentieren, ein Rechenprogramm für die Lösung quadratischer Gleichungen aufzustellen, wenn man im Laufe seiner täglichen Arbeit eine Vielzahl unterschiedlicher quadratischer Gleichungen mit nicht ganzzahligen Koeffizienten lösen muß, oder ein Programm zu erarbeiten, wenn man kompliziertere Funktionen wie z.B. die Funktion mit der Gleichung

$$y = (2x + 2)^{\sqrt[x]{2x+2}}$$

tabellieren und in einem Koordinatensystem darstellen soll, oder ein Programm für die Berechnung von Mittelwert und Streuung einer Meßreihe zu entwickeln, wenn man ständig wiederkehrend diese statistischen Kenngrößen für die unterschiedlichsten Meßreihen berechnen muß usw.

Die Programme sollte man dabei so formulieren, daß sie einen *möglichst allgemeingültigen Charakter* haben und *nicht auf spezielle Einzelfälle zugeschnitten* sind. Schließlich muß man sich zu jedem einzelnen Programm auch eine sogenannte „*Programmdokumentation*" anfertigen, aus der hervorgeht, in welcher Reihenfolge die auftretenden Eingangsparameter in den Rechner eingegeben werden müssen, welche Speicher vor Beginn der Rechnung evtl. durch bestimmte Zahlenwerte zu besetzen sind usw. Denn wenn man ein derartiges Programm einmal aufgestellt und erprobt hat, dann will man es ja später immer wieder benutzen, und man muß sich dann auch nach einer längeren Pause daran erinnern können, wie der genaue Ablauf des Programms zu erfolgen hat.

Aufgabe selbständig und sinnvoll abarbeitet, nennt man ein *Programm*. Die Tätigkeit des Menschen, ein solches Rechenprogramm aufzustellen, wird *programmieren* genannt.

Da der Mensch dem Rechner bis in jede Einzelheit genau vorschreibt, was getan werden soll, muß es eine Möglichkeit geben, daß man ein derartiges Programm dem Rechner mitteilen kann, damit er die gegebenen Befehle dann später ausführen kann. Dazu dient der Betriebszustand LOAD, der bei einigen Rechnertypen auch mit LD und bei anderen durch LRN (von learn (engl.) lernen) gekennzeichnet wird. Ist der Schiebeschalter in die Stellung LOAD (engl.: laden) bzw. LD oder LRN gebracht worden, so kann man die einzelnen Befehle in *genau der Reihenfolge in den Rechner eintasten, in der er sie später abarbeiten soll*. Damit der Rechner sich aber „merken" kann, was der Mensch von ihm verlangt, muß eine Gelegenheit vorhanden sein, daß der Rechner diese Befehle speichern kann. Es ist dies der *Programmspeicher*, den programmierbare Taschenrechner zusätzlich zu den üblichen *Zahlenspeichern* besitzen. Dieser Programmspeicher enthält eine von Rechnertyp zu Rechnertyp unterschiedliche Anzahl von Speicherzellen, auch *Befehlsregister* genannt, in denen jeweils nur ein einziger Befehl untergebracht werden kann.

Es gibt programmierbare Taschenrechner, deren Befehlsspeicher nur 58 bzw. 64 oder aber auch 72 Speicherplätze besitzen. In solchen Rechnern lassen sich natürlich keine Programme für umfangreiche und mit viel Rechenaufwand verbundene Probleme speichern. Es zeigt sich aber, daß schon derartige Rechner gute Dienste für viele kleinere *Routineaufgaben* leisten können.

Komfortablere Rechner besitzen Befehlsspeicher mit mehr als 100 Speicherplätzen, wobei der Trend der Weiterentwicklung dahin geht, möglichst viele Speicherplätze für Befehle in einem Rechner unterzubringen. Je mehr man nämlich Befehle in einem Rechner speichern kann, um so kompliziertere und rechenintensivere Aufgaben lassen sich mit ihm lösen.

Gibt man nun im Betriebszustand LOAD eine beliebige Tastenfolge in den Rechner ein, z.B.

[RCL] [1] [+] [RCL] [2] [=] [STO] [9],

so erscheint bei jeder neuen Eingabe in der Zahlenanzeige des Rechners jeweils eine bestimmte vier- bzw. fünfstellige Zahl. (Bei Rechnern mit weniger als 100 Befehlsspeicherplätzen treten vierstellige, bei Rechnern mit mehr als 100 Befehlsspeicherplätzen treten fünfstellige Zahlen auf.) Bei dem in Bild 1 dargestellten Rechner würden diese acht Zahlen wie folgt lauten:

65000 beim Drücken der Taste [RCL],
81001 beim Drücken der Taste [1],
84002 beim Drücken der Taste [+],
65003 beim Drücken der Taste [RCL],
82004 beim Drücken der Taste [2],
95005 beim Drücken der Taste [=],
64006 beim Drücken der Taste [STO],
63007 beim Drücken der Taste [9].

Wir wollen diese Zahlen, die bei jeder Eingabe eines neuen Befehls in der Zahlenanzeige erscheinen, *Befehlsworte* nennen. Jedes *Befehlswort* besteht aus zwei Teilen: einem zweistelligen *Operationsteil* und einem zwei- bzw. dreistelligen *Adreßteil*, wobei der Adreßteil zweistellig ist, wenn der Rechner weniger als 100 Befehlsspeicherplätze besitzt; hat der Rechner mehr als 100 Befehlsspeicherplätze, so muß der Adreßteil dreistellig sein.

Befehlswort

Aus dem *Operationsteil* eines Befehlswortes geht hervor, *welche Operation* durch den Rechner ausgeführt werden soll, während durch den *Adreßteil* angegeben wird, auf *welchem Speicherplatz des Befehlsspeichers* der jeweilige Befehl untergebracht worden ist.
Die auszuführenden Operationen sind bei jedem Rechner in einer ganz bestimmten Weise *verschlüsselt (codiert)*. Meist geschieht das in der Weise, daß der Operationsteil aus zwei Ziffern besteht, von denen die erste Ziffer die Tastenreihe

und die zweite Ziffer die Tastenspalte angibt, wo die Taste sich befindet, die zur Auslösung der entsprechenden Operation betätigt werden muß. Wenn also der Operationsteil des ersten Befehls unseres Beispiels mit 64 verschlüsselt ist, so ist die zu betätigende Taste in der 6. Reihe als vierte Taste zu finden. Ein Blick auf das Tastenfeld des Rechners in Bild 1 zeigt, daß dies die Taste [STO] ist.

Anmerkung: Bei einigen Rechnertypen werden die Zifferntasten 0 bis 9 nicht durch den hier verwendeten Reihen-Spalten-Code verschlüsselt, sondern sie werden durch Voransetzen einer 0 vor die jeweilige Ziffer codiert, so daß also beispielsweise die Zifferntaste [9] im Operationsteil durch die Ziffernfolge 09 charakterisiert wird.

Da man bei der Eingabe von Befehlen nie davor sicher ist, daß man einmal eine falsche Taste betätigt hat, wäre es gut, wenn man das gesamte eingegebene Programm noch einmal *schrittweise* auf seine Richtigkeit hin überprüfen könnte. Auch dafür ist auf fast allen Taschenrechnern eine Möglichkeit vorhanden. Da gibt es zunächst die Taste [SST] [single step (engl.) Einzelschritt], auf manchen Rechnern auch durch [STEP] gekennzeichnet. Betätigt man diese Taste im Betriebszustand LOAD des Rechners, so wird nacheinander jeder einzelne Speicherplatz des Befehlsspeichers aufgerufen, und die entsprechenden Befehlsworte können in der Zahlenanzeige abgelesen werden. Man kann also das ganze Programm noch einmal Schritt für Schritt kontrollieren. Voraussetzung dafür ist aber, daß man den Code der einzelnen Befehlsworte beherrscht und aus der entsprechenden Ziffernkombination des Befehlsteiles sofort auf die zugehörige Taste schließen kann. Am Anfang leistet hierfür eine *Codetabelle,* die jedem programmierbaren Rechner beigegeben ist, gute Dienste, bei längerer Arbeit mit einem Rechner wird man mit Sicherheit ohne diese Codetabelle auskommen.

Ähnliches geschieht bei der Betätigung der Taste [BACK] oder [BST] im Betriebszustand LOAD. Hier werden auch sämtliche Speicherplätze des Befehlsspeichers aufgerufen, so daß die jeweiligen Befehlsworte in der Zahlenanzeige sichtbar werden, nur erfolgt dieser Aufruf nicht in der an-

steigenden Reihenfolge der Speicherplätze, sondern in abfallender Folge.
Stellt man bei einer solchen Kontrolle fest, daß ein Befehl falsch eingetastet worden ist, so läßt er sich ganz einfach dadurch *korrigieren*, daß man an dieser Stelle den richtigen Befehl eintastet. *Der bisherige Inhalt des Befehlsspeicherplatzes wird dann durch den neuen Befehl überschrieben.*

Betätigt man die Tasten BACK oder BST im Betriebszustand RUN, so erfolgt *keinerlei Reaktion* des Rechners. Betätigt man hingegen im Betriebszustand RUN die Tasten STEP bzw. SST, so wird der jeweilige Programmschritt ausgeführt. Auf diese Weise können die Rechnungen eines Programms *schrittweise* ausgeführt werden, was für den *Programm*test und eine evtl. erforderliche *Fehlersuche* von großer Bedeutung ist.
Schließlich sei noch auf den Betriebszustand CLR verwiesen, der bei einigen Rechnertypen vorhanden ist. Steht der Schiebeschalter in der Stellung CLR und betätigt man in diesem Betriebszustand eine der Tasten SST bzw. STEP oder BST bzw. BACK, so werden dadurch die Speicherinhalte der Befehlsspeicherplätze *gelöscht*, die jeweils in der Zahlenanzeige ersichtlich sind. Im Betriebszustand CLR kann man also bestimmte Speicherplätze löschen, was wiederum bei der Programmkorrektur bedeutsam ist.

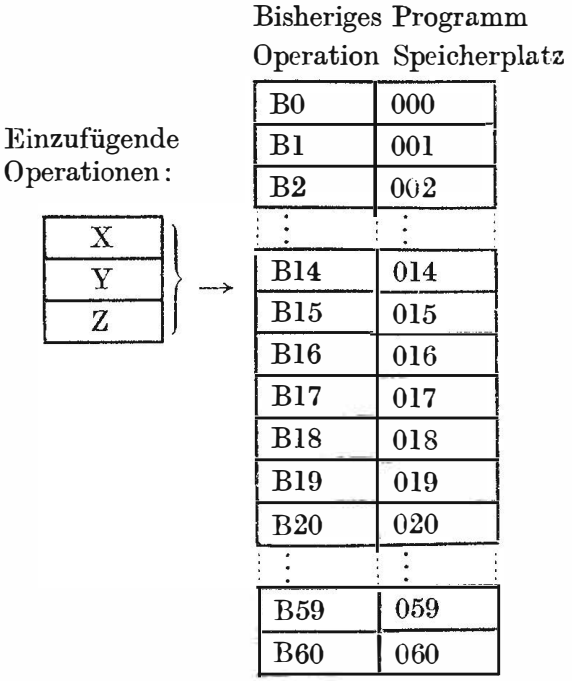
Andere Rechnertypen besitzen auch eine Taste DEL zum *Löschen einzelner Programmschritte.* Dabei erfolgt meist ein *Nachrücken der nachgeschalteten Befehle.* Soll das *Löschen eines Befehls ohne Nachrücken* des folgenden Befehls erfolgen, so kann die Taste NOP (engl: no operation) verwendet werden.

Es wurde bereits erwähnt, daß die Befehle in der Reihenfolge im Befehlsspeicher abgespeichert werden, in der sie in den Rechner eingegeben worden sind, und daß sie dann auch in genau der gleichen Reihenfolge im Betriebszustand RUN *abgearbeitet werden.*

Nun gibt es aber auch Fälle, in denen bei der Abarbeitung des Programms nicht die ursprünglich angegebene Befehlsfolge beibehalten werden kann. So haben wir beispielsweise bei unserem allerersten Programm am Ende den Befehl

GOTO 0 0 0 hinzugefügt, um zu erreichen, daß der Rechner, nachdem er den Ausdruck $c = \sqrt{a^2 + b^2}$ berechnet hat, nicht etwa zum nächsten Befehl weitergehen soll, sondern daß man ihn zwingt, wieder zum Befehl mit der Nummer (Adresse) 000 zurückzukehren. Der Befehl GOTO x y z ermöglicht es also, von der „normalen" Reihenfolge der Abarbeitung der eingegebenen Befehle abzuweichen und an die vorgeschriebene Stelle xyz des Befehlsspeichers zu „springen".

Mit Hilfe der GOTO-Taste lassen sich auch manche *Korrekturen an fehlerhaften Programmen* leicht bewerkstelligen. Nehmen wir an, wir hätten bei einem Programm, das die Speicherplätze 000 bis 060 des Befehlsspeichers besetzt, festgestellt, daß nach dem Befehl Nr. 015 noch die drei Operationen X, Y und Z eingefügt werden müssen.



Eine Möglichkeit der Korrektur des Programms würde darin bestehen, sich im Betriebszustand RUN mit Hilfe der SST-

Taste zunächst bis an den Befehl 16 heranzutasten und von da an zunächst die drei noch fehlenden Operationen X, Y und Z und danach alle folgenden Operationen B16, B17 bis B60 noch einmal einzugeben. Damit würden alle Speicherzellen von 016 bis 060 einen neuen Inhalt erhalten und an das Programm noch weitere drei Operationen angefügt.

Die [GOTO]-Taste gestattet die folgende, wesentlich elegantere Korrekturmöglichkeit, die mit einem weit geringeren Eingabeaufwand verbunden ist: Man betätigt zunächst die [SST]-Taste, bis man den Befehl Nr. 16 erreicht hat, und gibt von diesem Befehl an die vier Operationen

[GOTO] [0] [6] [1]

ein. Damit werden die Speicherzellen 016 bis 019 mit einem neuen Inhalt versehen, und der bisherige Inhalt dieser Speicherzellen geht verloren.

Im Anschluß daran verwendet man wieder die [SST]-Taste, um bis zum Befehl Nr. 061 zu gelangen. Von da an gibt man die Tastenfolge

[X] [Y] [Z] [B16] [B17] [B18] [B19]
[GOTO] [0] [2] [0]

ein, so daß das korrigierte Programm wie folgt gespeichert ist:

Operation	Speicherplatz
B0	000
B1	001
B2	002
⋮	⋮
B15	015
GOTO	016
0	017
6	018
1	019
B20	020
⋮	⋮

B59	059
B60	060
X	061
Y	062
Z	063
B16	064
B17	065
B18	066
B19	067
GOTO	068
0	069
2	070
0	071

Mit dieser Korrektur läuft das Programm nunmehr wie folgt ab: Zunächst arbeitet der Rechner gewohnheitsgemäß der Reihe nach die Operationen B0 bis B15 ab. Dann erhält er den Auftrag, die weitere Rechnung mit dem Befehl fortzusetzen, der im Speicherplatz Nr. 061 zu finden ist. Es werden also nun die Operationen X, Y, Z, B16, B17, B18 und B19 durchgeführt. Danach kommt der Auftrag, das Programm nunmehr von der Stelle 020 ab weiterzuführen, d.h., es werden nun die Operationen B20, B21 bis B60 durchgeführt. – Wie man sich leicht überzeugen kann, sind dadurch die drei bisher fehlenden Operationen X, Y und Z an der richtigen Stelle ins Programm eingefügt worden, das fehlerhafte Programm wurde richtig korrigiert.

Statt sich mit Hilfe der [SST]-Taste schrittweise bis zum Befehl B16 vorzutasten und nach Einfügen der Sprunganweisung erneut schrittweise bis zum Befehl B60 voranzugehen, kann man auch wie folgt verfahren: Man stellt den Betriebszustand RUN ein und drückt die Tastenfolge [GOTO] [0] [1] [6]. Dadurch wird der Befehlszähler des Rechners sofort auf den Befehl 016 eingestellt. Nun schaltet man um in den Betriebszustand LOAD und fügt die Tastenfolge [GOTO] [0] [6] [1] ein, die nunmehr in den Befehls-

speicherzellen 016 bis 019 untergebracht wird. Nun wird wiederum auf RUN umgeschaltet und die Tastenfolge [GOTO] [0] [6] [1] eingegeben, wodurch der Rechner sich auf den Befehl 061 einstellt. Danach kann nunmehr im Zustand LOAD wie vorn beschrieben weitergearbeitet werden. Weitere Anwendungsmöglichkeiten der GOTO-Anweisung sollen an späterer Stelle ausführlich behandelt werden.
Schließlich sei noch darauf verwiesen, daß auf programmierbaren Taschenrechnern weitere Tasten vorhanden sind mit Bezeichnungen wie

[X ≧ 0] [X = 0] [X < Y] [X < 0]

usw.
Tasten dieser Art besitzen für die Programmierung komplizierterer Aufgaben eine große Bedeutung. Ihre Wirkung während des Ablaufs eines Rechenprogramms soll zunächst am Beispiel der Taste [X ≧ 0], die auch bei einigen Rechnertypen die Bezeichnung [SKIP] besitzt, erläutert werden.
Tritt in einem Rechenprogramm an irgendeiner Stelle der Befehl [X ≧ 0] auf, so stellt der Rechner zunächst fest, ob der Inhalt des X-Registers in diesem Augenblick ≥ 0 oder ob er < 0 ist. Ist der Inhalt des X-Registers ≥ 0, so führt der Rechner als nächsten Schritt den auf den Befehl [X ≧ 0] folgenden Befehl aus. Ist dagegen der Inhalt des X-Registers < 0, so übergeht der Rechner den folgenden Befehl. Welchen Nutzen man aus dieser Eigenschaft der Taste [X ≧ 0] für die Programmierung ziehen kann, soll bei der Behandlung der Programmierbeispiele an späterer Stelle näher erläutert werden.
Um das Prinzip dieser Vergleichstasten noch an einem weiteren Beispiel zu verdeutlichen, soll die Taste [X > Y] noch einmal näher betrachtet werden. Wird die Taste [X > Y] in einem Programm verwendet, so vergleicht der Rechner an dieser Stelle die Inhalte des X- und des Y-Registers miteinander. Ist der Inhalt des X-Registers größer als der des Y-Registers, so führt der Rechner den folgenden Befehl aus. Sind die Inhalte von X- und Y-Register gleich oder ist der Inhalt des X-Registers kleiner als der des Y-Registers,

so überspringt der Rechner den folgenden Befehl des Programms.
Je mehr derartige Vergleichstasten auf einem programmierbaren Rechner vorhanden sind, um so einfacher lassen sich auftretende Vergleichsoperationen im Programm formulieren.
Auf den für unsere später folgenden Programmierbeispiele zugrunde gelegten Rechnern KREULOTRON 5 und KREULOTRON 6 (Bilder 1 und 2) ist jedoch absichtlich nur die eine Vergleichstaste $\boxed{X \geqq 0}$ vorgesehen worden, weil daran demonstriert werden soll, daß man mit einem Rechner, der nur eine einzige Vergleichstaste besitzt, jeden beliebigen Vergleich programmtechnisch bewältigen kann.
Will man beispielsweise mit Hilfe eines Rechners, der nur eine $\boxed{X \geqq 0}$-Taste besitzt, feststellen, ob eine im Verlauf einer Rechnung auftretende Größe $b < 0$ ist, so braucht man nur den Wert $-b$ ins X-Register zu bringen und zu untersuchen, ob $-b \geqq 0$ ist. — Soll festgestellt werden, ob $a \geqq b$ ist, so bringe man die Differenz $a - b$ ins X-Register und betätige die Taste $\boxed{X \geqq 0}$.
Etwas schwieriger ist das Problem, festzustellen, ob eine Größe z den Wert Null besitzt. Hier sei empfohlen, die Differenz $d = 0 - z^2$ ins X-Register zu bringen. Der Rechner wird dann die Frage $\boxed{X \geqq 0}$ nur in dem Fall mit „ja" beantworten, wenn $z = 0$ ist, in jedem anderen Fall ist die Bedingung $\boxed{X \geqq 0}$ nicht erfüllt.

Anmerkung: Es wird vielleicht verwundern, warum hier die Differenz $0 - z^2$ gebildet wurde und warum man nicht nur den Ausdruck $-z^2$ ins X-Register bringen soll. Der Grund liegt darin, daß viele Rechner die Zahl -0 nicht als Null, sondern als eine negative Zahl interpretieren, womit die Frage, ob $-0 \geqq 0$ ist, mit „nein" beantwortet würde.

3. Das Aufstellen von Rechenprogrammen

3.1. Vorbetrachtungen

In diesem Hauptabschnitt dieses Buches soll dargestellt werden, welchen Weg man einschlagen muß, um *von einer Problemstellung zum zugehörigen Rechenprogramm* und damit zur automatischen Lösung des Problems durch den Rechner gelangt. Diesem Vorhaben stellt sich aber eine Schwierigkeit in den Weg: der spezielle Aufbau der einzelnen Typen der auf dem Markt befindlichen programmierbaren Taschenrechner ist derartig unterschiedlich, daß es kaum möglich sein dürfte, Programme aufzustellen, die für *jeden* Rechnertyp brauchbar sind. Es kann uns hier nur darum gehen, die *prinzipielle Vorgehensweise* und die *einzelnen Arbeitsetappen*, die beim Programmieren durchlaufen werden müssen, so ausführlich wie möglich zu erläutern. Damit diese erforderlichen Einzelschritte bis in alle Einzelheiten besprochen werden können, legen wir den folgenden Abschnitten zwei Rechner zugrunde, die in Bild 1 und in Bild 2 dargestellt sind und die folgende Eigenschaften besitzen sollen:
Der Rechner KREULOTRON 5 (Bild 1) soll ein programmierbarer *Taschenrechner mit algebraischer Logik und mehrfachen Klammern* sein. Die mathematischen und sonstigen Funktionen, die er auszuführen in der Lage sein soll, sind aus der Darstellung in Bild 1 ersichtlich. Bei der Klammerbildung sollen mindestens vierfache Klammern ineinandergeschachtelt werden können. Die Berechnung der Funktionswerte von einer Variablen, z.B. $\ln x$, $\sin x$, e^x usw., soll nur das X-Register in Anspruch nehmen. Die in den übrigen Registern vorhandenen Zahlenwerte sollen unverändert beibehalten werden.
Es soll nicht vorausgesetzt werden, daß der Rechner eine Konstantenautomatik besitzt. Dadurch werden sich bei der Berechnung mancher Ausdrücke Rechenwege ergeben, die sich bei Rechnern mit Konstantenautomatik mit weniger

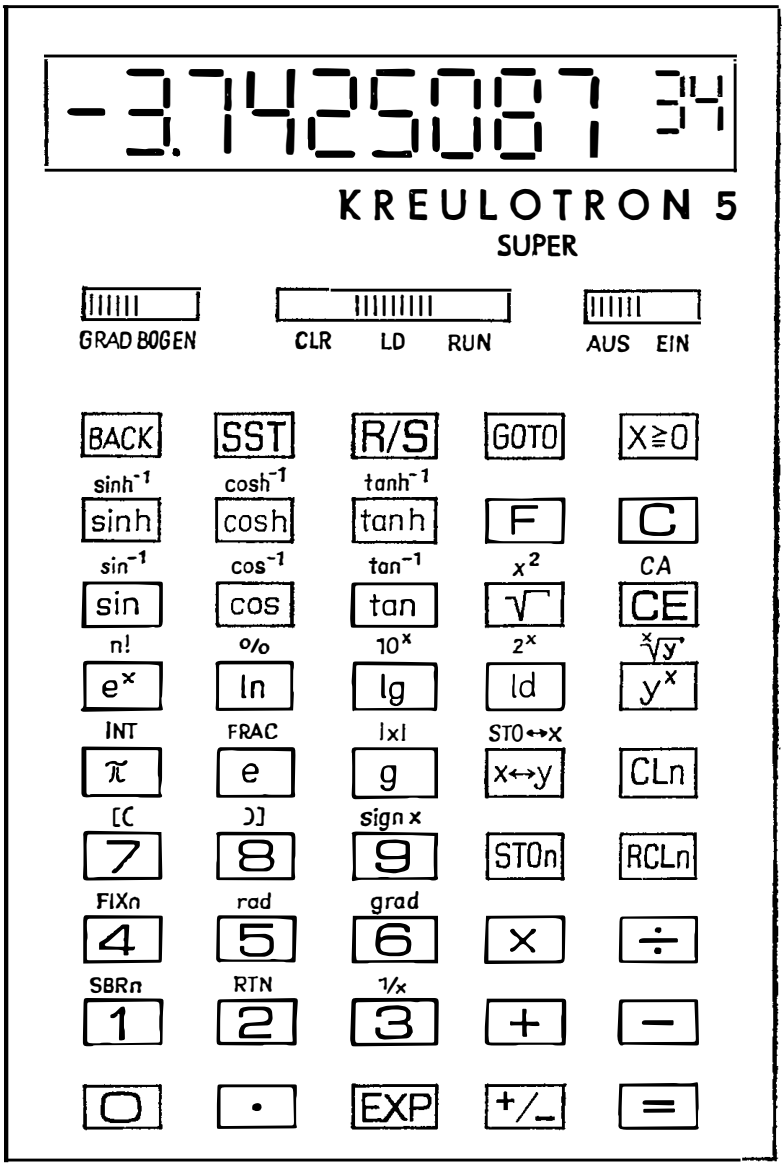

-3.1425087 34
KREULOTRON 5
SUPER
GRAD BOGEN
CLR LD RUN
AUS EIN
BACK SST R/S GOTO X≧0
sinh⁻¹ cosh⁻¹ tanh⁻¹
sinh cosh tanh F C
sin⁻¹ cos⁻¹ tan⁻¹ x² CA
sin cos tan √ CE
n! % 10ˣ 2ˣ ˣ√y
eˣ ln lg ld yˣ
INT FRAC |x| STO↔X
π e g x↔y CLn
[C]J sign x
7 8 9 STOn RCLn
FIXn rad grad
4 5 6 × ÷
SBRn RTN 1/x
1 2 3 + −
0 . EXP +/− =

Bild 1

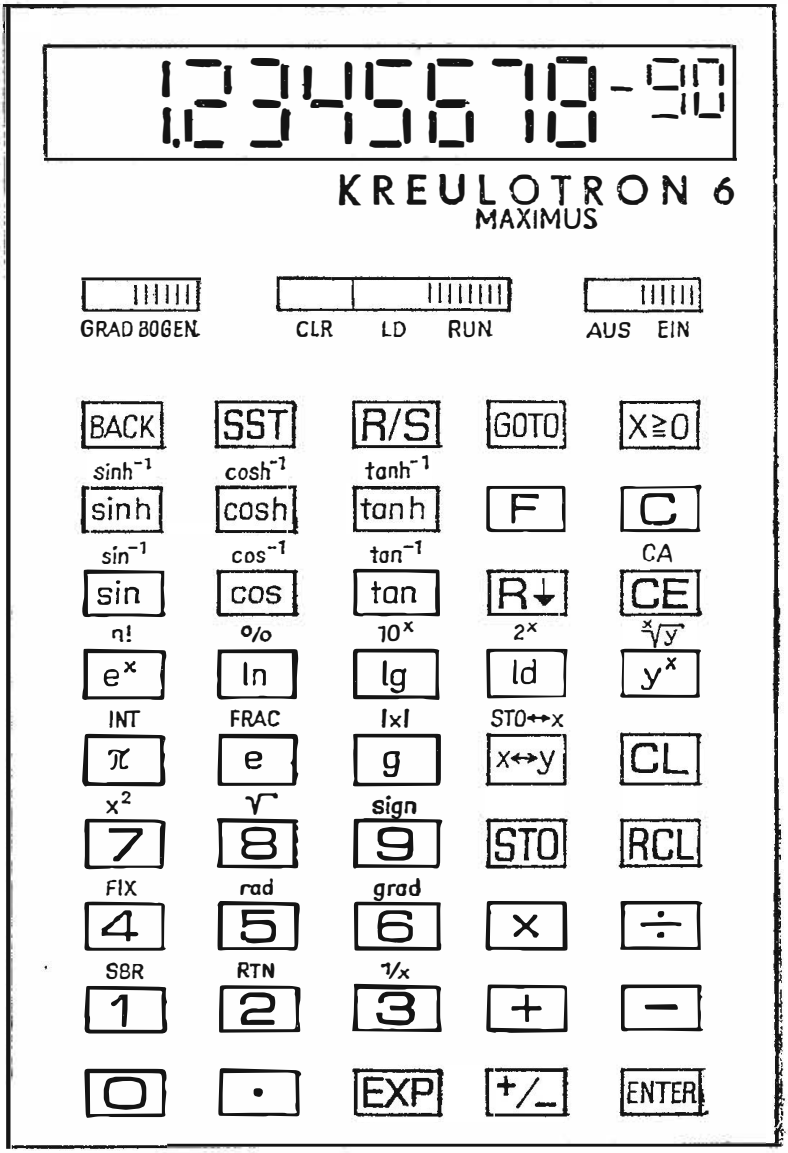

KREULOTRON 6
MAXIMUS
GRAD BOGEN
CLR LD RUN
AUS EIN
BACK SST R/S GOTO X≧0
sinh⁻¹ cosh⁻¹ tanh⁻¹
sinh cosh tanh F C
sin⁻¹ cos⁻¹ tan⁻¹ CA
sin cos tan R↓ CE
n! % 10^x 2^x
e^x ln lg ld y^x
INT FRAC |x| STO↔x
π e g x↔y CL
x² √ sign
7 8 9 STO RCL
FIX rad grad
4 5 6 × ÷
SBR RTN 1/x
1 2 3 + −
0 . EXP +/− ENTER

Bild 2

Eingaben an Befehlen realisieren lassen würden. Der Verzicht auf eine Konstantenautomatik hat aber den Vorteil, daß die hier erarbeiteten Programme eine größere Verallgemeinerungsfähigkeit aufweisen und damit auf fast jedem Rechner mit algebraischer Logik und Klammern nachvollzogen werden können.

Der Rechner KREULOTRON 6 (Bild 2) sei ein *Rechner mit Umgekehrter Polnischer Notation*. Er soll mindestens *vier Stack-Register* besitzen (vgl. dazu [6]), und im übrigen sei er in seinem Rechenkomfort bezüglich der vorhandenen mathematischen und sonstigen Funktionen dem Rechner KREULOTRON 5 sehr ähnlich.

Um auch umfangreichere Programme vorführen zu können, mögen beide Rechner mehr als 100 Programmspeicherplätze besitzen. Für die Leser, die Rechner mit weniger als 100 Programmspeicherplätze haben, bedeutet das keinen wesentlichen Nachteil, denn erstens sollen hier die Grundsätze für das Aufstellen von Rechenprogrammen dargeboten werden, und zum zweiten sind die erarbeiteten Programme so aufgebaut, daß sie meist in mehrere voneinander unabhängige Teilprogramme zerlegt werden können, die sich dann ohne weiteres in diesen kleineren Rechnern unterbringen lassen.

Ganz allgemein sei hier demjenigen, der sich einen programmierbaren Rechner neu anschaffen will, der Rat gegeben, sich nicht durch eine Vielzahl von Funktionstasten für die unterschiedlichsten mathematischen, statistischen, ökonomischen usw. Funktionen blenden zu lassen. Gewiß, je mehr Funktionstasten vorgesehen sind, um so weniger braucht sich der Nutzer zu bemühen, bestimmte Funktionen selbst zu berechnen. Aber mindestens ebenso wichtig wie ein hoher Komfort an verschiedenen Funktionstasten ist ein *möglichst umfangreicher Programmspeicher*, denn je mehr Speicherplätze der Programmspeicher eines programmierbaren Taschenrechners besitzt, um so umfangreichere Probleme lassen sich mit ihm lösen.

Dadurch, daß wir hier einen möglichst großen Programmspeicher zulassen, ergeben sich bei den Programmen, die Sprungbefehle enthalten, kleine Unterschiede zu den Programmen, die man für Rechner mit weniger als 100 Programmspeicherplätzen aufstellen muß. Während in unserem Fall nach GOTO immer eine dreistellige Ansprungadresse

folgen muß, genügt bei den Rechnern mit weniger als 100 Speicherplätzen eine zweistellige Adresse.
Schließlich sollen beide hier verwendeten Rechnertypen 10 einfache *Zahlenspeicher* besitzen, die durch die Befehle RCL 0 bis RCL 9 aufgerufen und durch die Befehle STO 0 bis STO 9 mit neuen Speicherinhalten versehen werden können.
Für die Erarbeitung von Programmen ist es vorteilhaft, wenn man sich eine ganz bestimmte Vorgehensweise angewöhnt, die man dann bei der Bearbeitung jeder Aufgabe auch konsequent beibehält. Es hat sich gezeigt, daß folgende *Arbeitsschritte* für die Aufstellung komplizierterer Programme unerläßlich sind:
Am Anfang steht eine gründliche *Analyse der gestellten Aufgabe*. Man muß sich unbedingt vollkommen klar darüber werden, was mit der Lösung der Aufgabe erreicht werden soll. Dabei ist zu ermitteln, welche Eingangsparameter für die Lösung der Aufgabe benötigt werden und welche Ergebniswerte im Lauf der Rechnung vom Rechner ausgegeben werden sollen.
Jede Unklarheit in der Erfassung der Aufgabenstellung wird irgendwann während der Erarbeitung des Rechenprogramms weitere Schwierigkeiten nach sich ziehen. Aus diesem Grund sei dringend davor gewarnt, sofort nach Kenntnisnahme der Aufgabe zum Taschenrechner zu greifen und zu versuchen, ein Programm aus den Ärmeln zu schütteln. Bei sehr einfachen Problemen mag das noch gehen, bei komplizierteren Aufgaben wird man auf diese Weise kaum ein lauffähiges Programm zustande bringen. Zur Analyse der Aufgabenstellung gehört es u.a. auch, daß man sich Gedanken darüber macht, ob die Aufgabe unter allen Umständen lösbar sein wird oder ob es vielleicht Kombinationen der Eingangsparameter geben kann, für die keine oder vielleicht auch mehrere Lösungen existieren können.
Nachdem völlige Klarheit über das Ziel der Aufgabe besteht, ist ein *geeigneter Lösungsalgorithmus* auszuwählen, mit dessen Hilfe die Aufgabe gelöst werden kann.
Handelt es sich um ein umfangreicheres Problem, das nicht ohne weiteres in seinem vollen Umfang überblickt werden kann, so empfiehlt es sich, einen *Programmablaufplan* anzufertigen, in dem in übersichtlicher Weise alle Einzelschritte aufgeführt werden, die im Lauf der Rechnung durchgeführt

werden müssen. Eine derartige zeichnerische Darstellung der zeitlichen Aufeinanderfolge aller durchzuführenden Operationen hat den Vorteil, daß man mit ihrer Hilfe vor Beginn der Programmierung noch einmal jede einzelne vorgesehene Maßnahme kritisch überprüfen kann, ob sie richtig ist und ob sie auch an der richtigen Stelle durchgeführt wird. Einen solchen sehr intensiven *Test des aufgestellten Programmablaufplanes* sollte man keinesfalls unterlassen.

Nachdem man sich überzeugt hat, daß der Rechenweg, für den man sich entschieden hat, ordnungsgemäß funktioniert, muß man für viele während der Rechnung auftretenden Rechengrößen Speicherplätze vorsehen und diese Speicherbelegung auch in einem *Speicherbelegungsplan* festhalten. Man muß als Programmierer stets einen Überblick darüber besitzen, wo die einzelnen Rechengrößen im Rechner untergebracht sind. Man kann dem Rechner nämlich nicht einfach die Anweisung geben, er möge die beiden Zahlen a und b miteinander multiplizieren und sich den Wert des Produkts $c = a \cdot b$ als Zwischenergebnis für eine später erfolgende andere Rechnung merken, sondern man muß ihm genau sagen, wo er die beiden miteinander zu multiplizierenden Zahlen in seinen Speichern findet und wo er das Ergebnis seiner Berechnung unterbringen soll, damit er später, wenn er es wieder benötigt, Zugriff zu ihm hat.

Nunmehr sind alle Vorbereitungsarbeiten so weit gediehen, daß man an die *Aufstellung des eigentlichen Rechnerprogramms* gehen kann. Nach gründlicher Übung wird man unter Umständen ein derartiges Programm aus dem vorliegenden Programmablaufplan heraus sofort in den Rechner eintippen können. Es sei aber dennoch der Rat gegeben, das ganze Programm erst einmal in all seinen Einzelschritten aufzuschreiben und es erst danach in den Rechner einzugeben. Das hat den Vorteil, daß man bei eventuell auftretenden Programmierfehlern eine schriftliche Dokumentation darüber hat, was man zu tun beabsichtigte, so daß ein Auffinden des Fehlers und seine Korrektur möglich wird. Als mindestens ebenso großer Vorteil ist es aber anzusehen, daß man das Programm nicht immer neu aufstellen muß, wenn man es zu einem späteren Zeitpunkt einmal wieder benötigt.

Die *Eingabe eines Programms* in den Rechner ist ebenso wie die Eingabe vieler Daten hintereinander ein sehr neuralgischer Punkt bei der Arbeit mit Taschenrechnern. Es kann

sehr schnell passieren, daß man vergißt, einen Befehl einzugeben, weil man gerade an dieser Stelle durch das Klingeln des Telefons oder durch irgendeinen anderen Anlaß gestört worden ist. Daher ist es unerläßlich, daß man das vollständig eingegebene Programm noch einmal aufmerksam mit Hilfe der SST-Taste *schrittweise überprüft*, um derartige Eingabefehler zu erkennen und zu *korrigieren*. Man sollte es sich aber auch zur Gewohnheit machen, ein erarbeitetes Programm mit Hilfe sehr einfacher Testzahlen als Eingangsparameter auf seine Richtigkeit hin zu überprüfen. Erst wenn dieser *Programmtest* erfolgreich verlaufen ist, sollte man das erarbeitete Programm für die Rechnung freigeben.
Da man Rechenprogramme im allgemeinen nicht nur für eine einmalige Abarbeitung am Rechner aufstellt, sondern diese Programme meist immer wieder nutzen will, ist zu empfehlen, sich eine kleine *Programmbibliothek* anzulegen, in der man alle Programme sammelt, die man bisher erfolgreich zum Laufen gebracht hat. Zu dieser Sammlung der Programme gehört es aber auch, daß man zu jedem Programm eine kleine *Programmdokumentation* anfertigt, aus der hervorgeht, in welcher Reihenfolge die einzelnen Eingangsparameter in den Rechner eingegeben werden müssen, welche sonstigen Bedienhandlungen am Rechner während der Abarbeitung des Programms erforderlich sind und in welcher Weise die Ergebnisse zu interpretieren sind, die vom Rechner angezeigt werden.
Eine solche Programmdokumentation sollte so *übersichtlich und vollständig wie möglich* angefertigt werden, damit auch ein Außenstehender, der das Programm gar nicht kennt, allein an Hand der niedergeschriebenen Befehlsfolge und der angefertigten Programmdokumentation ein gleichartiges Problem ohne weiteres lösen kann.

3.2. Geradeausprogramme

3.2.1. Lineare Gleichungssysteme mit zwei Unbekannten

Im ersten Beispiel soll ein Programm aufgestellt werden, mit dessen Hilfe die beiden Lösungen x und y des linearen Gleichungssystems

$$a_1 x + b_1 y = c_1$$
$$a_2 x + b_2 y = c_2$$

berechnet werden können. Es soll dabei vorausgesetzt werden, daß die Koeffizienten des Gleichungssystems so gewählt sind, daß ein eindeutiges Lösungspaar $(x; y)$ existiert.
Die Aufgabenstellung ist bei diesem Beispiel so klar formuliert, daß eigentlich keine Fragen offen bleiben dürften. Als Rechenverfahren bieten sich die Determinantengesetze an, mit deren Hilfe sich die beiden Lösungen x und y des Gleichungssystems in der Form

$$x = \frac{c_1 b_2 - c_2 b_1}{a_1 b_2 - a_2 b_1} \qquad y = \frac{a_1 c_2 - a_2 c_1}{a_1 b_2 - a_2 b_1}$$

darstellen lassen.

Bei der Betrachtung dieser beiden Ausdrücke fällt auf, daß in beiden Brüchen der gleiche Nenner auftritt. Es dürfte also sinnvoll sein, zuerst diesen Nenner zu berechnen, seinen Wert zwischenzuspeichern und danach erst die Berechnung von x und y zu beginnen. Auf diese Weise lassen sich einige Rechenschritte einsparen.
Der *Programmablaufplan* für die Berechnung der beiden Lösungen x und y ist in Bild 3 dargestellt.

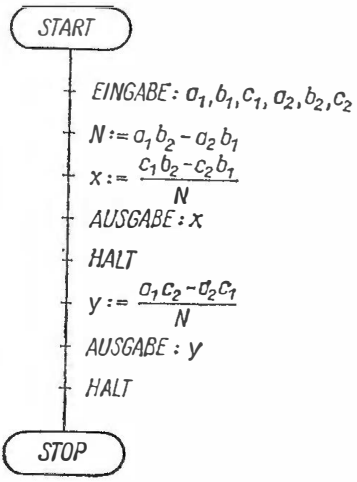


Bild 3

Jeder Programmablaufplan beginnt mit dem Symbol (START) und endet mit dem Symbol (STOP). Diese beiden Symbole sind im einfachsten Fall wie hier durch eine sogenannte *Leitlinie* miteinander verbunden, an der der Reihe nach alle wichtigen Operationen angeführt sind, die im Verlauf der Rechnung ausgeführt werden sollen.
In unserem Fall sollen also zunächst die Koeffizienten des Gleichungssystems eingegeben werden. Danach ist der Nenner N der beiden Brüche zu berechnen und schließlich die beiden Werte für x und y, die nach der jeweiligen Berechnung ausgegeben werden sollen. Der Hinweis HALT hinter der Ausgabe der Werte x und y weist darauf hin, daß beim Programmieren darauf geachtet werden muß, daß der Rechner nach der Ermittlung dieser Werte aufgefordert werden muß, anzuhalten, damit die Lösungen abgelesen werden können. Würde man im Programm keinen Haltebefehl vorsehen, so würde der Rechner unmittelbar nach der Berechnung von x zur Berechnung von y übergehen, und infolge der hohen Rechengeschwindigkeit würde der Nutzer des Programms nicht erkennen, welche Zahlenwerte für die beiden Lösungen ausgerechnet worden sind.
Ein besonderer *Test des Programmablaufplans* kann hier unterbleiben, da er sehr einfach und übersichtlich aufgebaut ist.
Im Lauf der Rechnung werden alle Koeffizienten des Gleichungssystems mehrfach benötigt, ebenso der anfangs berechnete Nenner. Es macht sich also erforderlich, für alle diese Größen Speicherplätze vorzusehen.
Die Belegung der Speicherplätze des Rechners geht aus dem folgenden Speicherbelegungsplan hervor:

Speicherbelegungsplan:

	Speicher	
	Nummer	Inhalt
Vor Rechenbeginn zu belegende Speicher:	0	a_1
	1	b_1
	2	c_1
	3	a_2
	4	b_2
	5	c_2
Arbeitsspeicher:	6	N

Anmerkung: Es wird hier unterschieden zwischen Speichern, die vor der Abarbeitung des Programms durch den Menschen belegt werden müssen, und Speichern, die während der Abarbeitung automatisch vom Rechner besetzt werden und die *Arbeitsspeicher* genannt werden sollen.

Nun kann an die *Aufstellung des Programms* gegangen werden. Alle im folgenden aufgestellten Programme werden in zwei Versionen angegeben, in einer Version für Rechner mit algebraischer Rechenlogik und einer Version für Rechner mit Umgekehrter Polnischer Notation. Beide Programmfassungen werden immer gleich unmittelbar nebeneinander angeführt. Dabei steht jeweils

in der ersten Spalte die zu betätigende Taste des Rechners mit algebraischer Rechenlogik,

in der zweiten Spalte die Verschlüsselung der Eingabe, bezogen auf den in Bild 1 dargestellten Rechner,

in der dritten Spalte die zu betätigende Taste bei einem Rechner mit Umgekehrter Polnischer Notation,

in der vierten Spalte die Verschlüsselung dieser Eingabe, bezogen auf den in Bild 2 dargestellten Rechner, und schließlich folgen in der fünften Spalte ggf. einige Bemerkungen, die den Rechengang näher kommentieren sollen.

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Poln. Notation		Bemerkungen
Eingabe	Code	Eingabe	Code	
R/S	13–000	R/S	13–000	Halt für die Eingabe der Koeffizienten des Gleichungssystems
RCL 0 [1])	65–001 91–002	RCL 0 [1])	65–001 91–002	a_1
×	74–003			
RCL 4	65–004 71–005	RCL 4	65–003 71–004	b_2
–	85–006	×	74–005	$a_1 b_2$
F	24–007			
(	61–008			
RCL 3	65–009 83–010	RCL 3	65–006 83–007	a_2
×	74–011			
RCL 1	65–012 81–013	RCL 1	65–008 81–009	b_1
F	24–014			
)	62–015	×	74–010	$a_2 b_1$
=	95–016	–	85–011	$N = a_1 b_2 - a_2 b_1$
STO 6	64–017 73–018	STO 6	64–012 73–013	Speichern von N

[1]) Für das Speichern einer Zahl in bzw. das Rückrufen einer Zahl aus dem Speicher mit der Nummer n müssen zwei Tasten betätigt werden: STO n bzw. RCL n. Um eine bessere Lesbarkeit zu erreichen, werden künftig diese beiden Symbole stets zu einem vereinigt: STO n bzw. RCL n.

RCL 2	65–019	RCL 2	65–014	
	82–020		82–015	c_1
×	74–021			
RCL 4	65–022	RCL 4	65–016	
	71–023		71–017	b_2
–	85–024	×	74–018	$c_1 b_2$
F	24–025			
(	61–026			
RCL 5	65–027	RCL 5	65–019	
	72–028		72–020	c_2
×	74–029			
RCL 1	65–030	RCL 1	65–021	
	81–031		81–022	b_1
F	24–032			
)	62–033	×	74–023	$c_2 b_1$
÷	75–034	–	85–024	$c_1 b_2 - c_2 b_1$
RCL 6	65–035	RCL 6	65–025	
	73–036		73–026	N
=	95–037	÷	75–027	$x = (c_1 b_2 - c_2 b_1)/N$
R/S	13–038	R/S	13–028	HALT für Ausgabe von x
RCL 0	65–039	RCL 0	65–029	
	91–040		91–030	a_1
×	74–041			
RCL 5	65–042	RCL 5	65–031	
	72–043		72–032	c_2
–	85–044	×	74–033	$a_1 c_2$
F	24–045			
(	61–046			
RCL 3	65–047	RCL 3	65–034	
	83–048		83–035	a_2

×	74–049			
RCL 2	65–050	RCL 2	65–036	
	82–051		82–037	c_1
F	24–052			
)	62–053	×	74–038	a_2c_1
÷	75–054	−	85–039	$a_1c_2 - a_2c_1$
RCL 6	65–055	RCL 6	65–040	
	73–056		73–041	N
=	95–057	÷	75–042	$y = (a_1c_2 - a_2c_1)/N$
R/S	13–058	R/S	13–043	HALT für Ausgabe y
GOTO 000	14–059	GOTO 000	14–044	
	91–060		91–045	
	91–061		91–046	
	91–062		91–047	Rücksprung zum Programmanfang.

Anmerkung: Durch den Befehl GOTO 000 am Ende des Programms wird der Rechner aufgefordert, nach der Abarbeitung der Aufgabe wieder zum Programmanfang zurückzuspringen. Dadurch wird es ermöglicht, weitere lineare Gleichungssysteme mit zwei Unbekannten ohne Verzug lösen zu können.

Programmtest:

Das erarbeitete Programm liefert für das Gleichungssystem

$$3x + 4y = 18$$

$$4x + 3y = 17$$

die beiden Lösungen

$$x = 2 \quad \text{und} \quad y = 3\,.$$

Beim Versuch, das Gleichungssystem

$$2x - 3y = 6$$

$$4x - 6y = 14$$

mit diesem Programm zu lösen, tritt kurz nach dem Start des Programms eine Fehlermeldung auf.
Dies ist auch zu erwarten, denn die beiden angegebenen Gleichungen widersprechen einander. (Nach der ersten Gleichung soll $2x - 3y = 6$ sein, während nach der zweiten Gleichung $2x - 3y = 7$ gelten müßte. Dies ist nicht möglich!)
In diesem Fall ist der Nenner $N = 0$, und bei der Division durch 0 kommt es automatisch zu einer Fehleranzeige.

Programmdokumentation:

1. Einstellen der gewünschten Rechengenauigkeit, z.B. [F] [FIX] [4], wenn vier Stellen nach dem Komma gefordert werden.
2. Start des Programms durch Drücken der [R/S]-Taste. Startadresse: 000.
3. Eingabe der Koeffizienten des Gleichungssystems in folgende Speicher:

   a_1 in Speicher 0 — b_1 in Speicher 1
   c_1 in Speicher 2 — a_2 in Speicher 3
   b_2 in Speicher 4 — c_2 in Speicher 5

4. Starttaste [R/S] erneut drücken.
5. Beim ersten Halt erscheint in der Anzeige der Wert der Unbekannten x.
   [R/S]-Taste drücken.
6. Beim nächsten Halt erscheint in der Anzeige der Wert der Unbekannten y.
   [R/S]-Taste drücken.
7. Tritt während der Rechnung eine Fehleranzeige auf, so ist das gegebene Gleichungssystem nicht bzw. nicht eindeutig lösbar.

3.2.2. Lösung einer transzendenten Gleichung

In der mathematischen Schülerzeitschrift „alpha“ (vgl. [4]) wurde als Wettbewerbsaufgabe das folgende Problem gestellt:

Es sind alle reellen Lösungen der Gleichung

$$(2x + 2)^{\sqrt[x]{2x+2}} = 64$$

zu ermitteln.

Da es sich hier um eine transzendente Gleichung handelt, für die es kein „geschlossenes" Lösungsverfahren gibt, liegt der Gedanke nahe, eine Wertetabelle aufzustellen und mit Hilfe dieser Wertetabelle diejenigen x-Werte zu ermitteln, für die die Funktion

$$y = (2x + 2)^{(2x+2)^{1/x}}$$

den Wert 64 annimmt.

Daß Lösungen der gegebenen Gleichung existieren, geht aus der folgenden Überlegung hervor:

Der Definitionsbereich der Funktion $y = (2x + 2)^{(2x+2)^{1/x}}$ ist $D = (-1; +\infty) \setminus \{0\}$, denn für $x < -1$ wird der Radikand negativ, und für $x = 0$ ergibt sich Null im Nenner des letzten Exponenten.

Wir wollen hier nur den Bereich $B = (0; +\infty)$ untersuchen und überlassen es dem Leser, ähnliche Überlegungen auch für den Bereich $B_1 = (-1; 0)$ anzustellen.

Für kleine Werte von x erhält man große y-Werte, die jedoch mit wachsendem x kleiner werden, wie aus der folgenden kleinen Zusammenstellung hervorgeht:

x	0,5	1,0	1,5
y	19683	256	110,612

Die y-Werte werden jedoch nicht unbegrenzt kleiner, sondern von einem vorläufig noch nicht näher bestimmten x-Wert an müssen sie wieder ansteigen, denn für sehr große Werte von x nähert sich der Exponent $\sqrt[x]{2x+2}$ dem Wert 1, so daß die betrachtete Funktion sich für große Werte von x der linearen Funktion $y_1 = 2x + 2$ immer mehr nähern muß.

Da, wie leicht nachzuprüfen ist, die Funktion y für $x = 3$ den geforderten Wert 64 annimmt, ist zu erwarten, daß es jenseits des x-Wertes 3 noch eine weitere Lösung der Gleichung geben muß, sofern $x = 3$ nicht gerade eine Minimumstelle der Funktion ist.

Wir wollen daher eine Wertetabelle aufstellen, mit deren Hilfe wir die gesuchte Lösung immer mehr eingrenzen können, und dafür das Programm schreiben. Diese Wertetabelle soll zunächst mit einem Anfangswert x_0 beginnen und dann mit einer Schrittweite Δx voranschreiten, bis der y-Wert 64 überschritten wird. Sobald dies der Fall ist, können wir vom vorletzten x-Wert ausgehen und mit einer kleineren Schrittweite den y-Wert 64 genauer annähern, und dieses Verfahren kann so lange fortgesetzt werden, bis die geforderte Genauigkeit des Ergebnisses erreicht ist. Von dem aufzustellenden Programm wollen wir noch nicht fordern, daß es die Veränderung der Schrittweite selbständig vornimmt (obwohl dies prinzipiell möglich wäre, wie wir in späteren Abschnitten noch sehen werden).
Das Programm soll so aufgestellt werden, daß neben den Funktionswerten y auch gleichzeitig als Hilfsgröße der Wert der Wurzel $\sqrt[x]{2x+2}$ vom Rechner ausgegeben wird.
Der *Programmablaufplan* für das zu erarbeitende Programm ist in Bild 4 dargestellt.

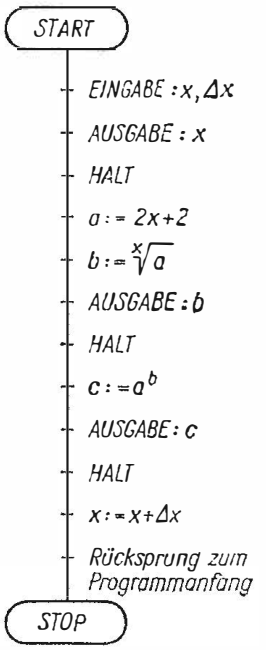


Bild 4

Aus diesem Programmablaufplan geht hervor, daß nach der Berechnung von $\sqrt[x]{2x+2}$ zunächst dieser Wurzelwert ausgegeben wird und daß danach die Berechnung und Ausgabe des Funktionswerts y erfolgt. Im Anschluß daran erfolgt die automatische Erhöhung des bisherigen x-Wertes um die Schrittweite Δx, so daß die Berechnung des nächsten y-Wertes nur noch durch die Betätigung der Taste R/S ausgelöst zu werden braucht.

Speicherbelegungsplan:

	Speicher	
	Nummer	Inhalt
Vor Beginn der Rechnung zu belegen:	0	x_0
Arbeitsspeicher:	3	Δx
	1	$2x+2$
	2	$\sqrt[x]{2x+2}$

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Poln. Notation		Bemerkungen
Eingabe	Code	Eingabe	Code	
R/S	13–000	R/S	13–000	HALT für Eingabe x_0 und Δx am Anfang der Berechnung, für eine evtl. erforderliche Veränderung im Verlauf der Berechnung
RCL 0	65–001	RCL 0	65–001	
×	91–002		91–002	x
R/S	13–003 74–004	R/S	13–003	HALT für Ausgabe von x
2	82–005	2	82–004	2
+	84–006	×	74–005	$2x$
2	82–007	2	82–006	2
=	95–008	+	84–007	$a = 2x + 2$
STO 1	64–009 81–010	STO 1	64–008 81–009	Speichern von $2x + 2 = a$
F	24–011			
$\sqrt[x]{\overline{Y}}$	45–012			
RCL 0	65–013 91–014	RCL 0	65–010 91–011	
		F	24–012	
=	95–015	$\sqrt[x]{\overline{Y}}$	45–013	$b = \sqrt[x]{2x + 2}$
STO 2	64–016 82–017	STO 2	64–014 82–015	Speichern des Exponenten b

R/S	13–018	R/S	13–016	HALT für Ausgabe des Exponenten
RCL 1	65–019	RCL 1	65–017	
	81–020		81–018	a
Y^x	45–021			
RCL 2	65–022	RCL 2	65–019	
	82–023		82–020	b
=	95–024	Y^x	45–021	$a^b = (2x+2)^{\sqrt[x]{2x+2}}$
R/S	13–025	R/S	13–022	HALT für Ausgabe von y
RCL 0	65–026	RCL 0	65–023	
	91–027		91–024	x
+	84–028			
RCL 3	65–029	RCL 3	65–025	
	83–030		83–026	Δx
=	95–031	+	84–027	$x + \Delta x$
STO 0	64–032	STO 0	64–028	Speichern des nächsten x-Wertes
	91–033		91–029	
GOTO 000	14–034	GOTO 000	14–030	
	91–035		91–031	
	91–036		91–032	
	91–037		91–033	Rücksprung zum Programmanfang

Programmtest:

Mit Hilfe des aufgestellten Programms wurde die folgende Wertetabelle für die Funktion

$$y = (2x+2)^{\sqrt[x]{2x+x}}$$

berechnet und aus ihr die Lösung der Gleichung

$$(2x+2)^{\sqrt[x]{2x+2}} = 64$$

ermittelt.

x	$\sqrt[x]{2x+2}$	$(2x+2)^{\sqrt[x]{2x+2}}$	Bemerkungen
0,5	9	19683	
1,0	4	256	
1,5	2,924	110,612	
2,0	2,449	80,551	
2,5	2,178	69,270	
3,0	2,000	64,000	Damit ist die erste Lösung der Gleichung gefunden.
3,5	1,873	61,337	
4,0	1,778	60,018	
4,5	1,704	59,475	
5,0	1,644	59,416	
5,5	1,594	59,678	Das Minimum der Funktion ist hier bereits überschritten.
6,0	1,552	60,162	
6,5	1,517	60,804	
7,0	1,486	61,562	
7,5	1,459	62,409	
8,0	1,435	63,322	
8,5	1,414	64,287	Zwischen 8,0 und 8,5 muß die zweite Lösung der Gleichung liegen. Daher wird die Wertetabelle erneut mit 8,0 begonnen und die Schrittweite auf 0,1 verringert.
8,0	1,435	63,287	
8,1	1,431	63,511	
8,2	1,426	63,702	
8,3	1,422	63,896	
8,4	1,418	64,091	Neuer Anfangswert: 8,30 neue Schrittweite: 0,01
8,30	1,422	63,896	
8,31	1,422	63,915	
8,32	1,421	63,934	
8,33	1,421	63,954	

x	$\sqrt[x]{2x+2}$	$(2x+2)^{\sqrt[x]{2x+2}}$	Bemerkungen
8,34	1,421	63,973	
8,35	1,420	63,993	
8,36	1,420	64,012	Neuer Anfangswert: 8,350 neue Schrittweite: 0,001
8,350	1,420	63,993	
8,351	1,420	63,995	
8,352	1,420	63,997	
8,353	1,420	63,999	
8,354	1,420	64,001	Neuer Anfangswert: 8,3530 neue Schrittweite: 0,0001 neue Genauigkeit: 4 Stellen
8,3530	1,4200	63,9987	
8,3531	1,4200	63,9989	
8,3532	1,4200	63,9991	
8,3533	1,4200	63,9993	
8,3534	1,4200	63,9995	
8,3535	1,4199	63,9997	
8,3536	1,4199	63,9999	
8,3537	1,4199	64,0001	

Da in der vierten Stelle der y-Werte ein lineares Anwachsen zu verzeichnen ist, kann die fünfte Stelle des Ergebnisses ohne weiteres durch lineare Interpolation bestimmt werden. Die zweite Lösung der gegebenen Gleichung lautet demnach auf fünf Stellen genau: $x_2 = 8{,}35365$.
Damit sind die möglichen positiven reellen Lösungen der Gleichung

$$(2x+2)^{\sqrt[x]{2x+2}} = 64$$

gefunden. Sie lauten

$$x_1 = 3 \quad \text{und} \quad x_2 = 8{,}35365\,.$$

Programmdokumentation:

1. Einstellen der gewünschten Rechengenauigkeit

2. Start des Programms (R/S -Taste drücken).
   Startadresse: 000
3. Eingabe des Anfangswertes x_0 für die Tabellierung in Speicher 0 und der Schrittweite Δx in Speicher 3.
4. R/S -Taste drücken.
5. Beim ersten HALT wird der Zahlenwert von x ausgegeben, beim zweiten HALT der zugehörige Wert des Exponenten $\sqrt[x]{2x+2}$ und beim dritten HALT der Funktionswert $y = (2x+2)^{\sqrt[x]{2x+2}}$.
   Nach jedem Halt ist erneut die R/S -Taste zu drücken.
6. Kommt man mit dem Funktionswert y immer näher an den Wert 64 heran, so kann man durch Verkleinerung der Schrittweite Δx (neue Schrittweite Δx in den Speicher 3 eingeben!) die Genauigkeit der Lösung der gestellten Aufgabe immer mehr verbessern.

3.3. Programmverzweigungen

Die bisher vorgeführten Beispiele stellen die einfachste Form von Programmen dar. Bei ihnen handelt es sich um Aufgaben, bei denen die Rechnung vom Anfang bis zum Ende in einem vorher festlegbaren, von den Zwischenergebnissen unabhängigen Ablauf erfolgt. Die Befehle werden dann auch vom Rechner in genau der Reihenfolge abgearbeitet, wie sie durch das Programm vorgeschrieben werden. Aus diesem Grund werden diese einfachen Programme auch *Geradeausprogramme* genannt.
Bei den meisten komplizierteren Aufgabenstellungen kommt jedoch sehr häufig der Fall vor, *daß die Rechnung in Abhängigkeit von gewissen Zwischenergebnissen in durchaus verschiedenartiger Weise fortzusetzen ist.* Derartige Stellen des Programms, an denen die weitere Rechnung einen anderen Verlauf nimmt, je nachdem, ob beispielsweise eine zuletzt berechnete Größe positiv oder negativ ist, nennt man *Verzweigungsstellen* des Programms.
Wir wollen zunächst an einigen sehr einfachen Beispielen kennenlernen, wie programmiert werden muß, wenn derartige Verzweigungen in einem Programm auftreten.

3.3.1. Berechnung und Speicherung des Betrages einer Zahl a

Es soll ein Programm aufgestellt werden, das folgende Teilziele realisiert: Nach der Eingabe der Zahl a soll diese Zahl zunächst noch einmal in der Zahlenanzeige zur Kontrolle erscheinen. Danach ist der Betrag b dieser Zahl zu bestimmen und im Speicher 1 zu speichern.
Bevor wir an die Aufstellung des Programms gehen, muß einiges zur Aufgabenstellung selbst vorangeschickt werden. Man wird gegen diese Aufgabe einwenden, daß man keinen programmierten Taschenrechner benötigt, um den Betrag einer Zahl zu ermitteln. Die meisten Rechner besitzen eine Sondertaste zur Bestimmung des Betrages, und selbst wenn eine derartige Taste nicht vorhanden ist, kann man die gestellte Aufgabe für jeden speziellen Fall sofort im Kopfe lösen.
Da es uns vordergründig nicht um die Ermittlung des Betrages einer Zahl, sondern um die Überlegungen geht, die erforderlich sind, wenn man ein Programm selbst für eine derartig primitive Aufgabe aufstellen soll, wurde dieser sehr einfache Fall als erstes Beispiel gewählt. – Zum zweiten kann ohne weiteres der Fall eintreten, daß im Verlauf einer umfangreichen Aufgabe das Problem auftritt, den Betrag eines Zwischenwertes ermitteln zu müssen, der während der Rechnung entsteht und den der Mensch wegen des automatischen Ablaufes der Rechnung gar nicht kennt. In diesem Fall könnte das hier aufgestellte Programm, vorausgesetzt der verwendete Rechner hat keine Absolutwert-Taste, ohne weiteres als Teilprogramm verwendet werden.
Die Berechnung von b mit

$$b = |a|$$

erfolgt nach der Definition des Betrages einer Zahl a:

$$b = |a| = \begin{cases} a, \text{ wenn } a \geqq 0 \\ -a, \text{ wenn } a < 0\,. \end{cases}$$

Daraus folgt, daß in Abhängigkeit des jeweiligen Wertes von a zwei *unterschiedliche Rechengänge* erforderlich sein können: Ist $a \geqq 0$, so kann $b = a$ gesetzt und gespeichert werden, ist dagegen $a < 0$, so ist $b = -a$ zu setzen und zu speichern.

Der Rechner muß also zunächst einmal feststellen, ob a positiv, Null oder negativ ist. Dies kann mit der Taste $\boxed{X \geqq 0}$ erfolgen. – Im *Programmablaufplan* wird dies durch das Zeichen $P\,\{a \geqq 0\,?\}$ veranschaulicht. Dieses Zeichen soll bedeuten, daß von der Prüfgröße a festzustellen ist, ob sie $\geqq 0$ ist oder nicht. Wird diese Frage mit „ja" beantwortet, so erfolgt der weitere Rechenablauf entsprechend demjenigen Zweig des Programmablaufplans, der mit „ja" gekennzeichnet ist. Lautet die Antwort auf die gestellte Frage dagegen „nein", so sind diejenigen weiteren Anweisungen durchzuführen, die in dem Zweig des Programmablaufplans liegen, der mit „nein" gekennzeichnet ist, d.h., es ist die Anweisung $b := -a$ auszuführen. Der Programmablaufplan ist in Bild 5 dargestellt.

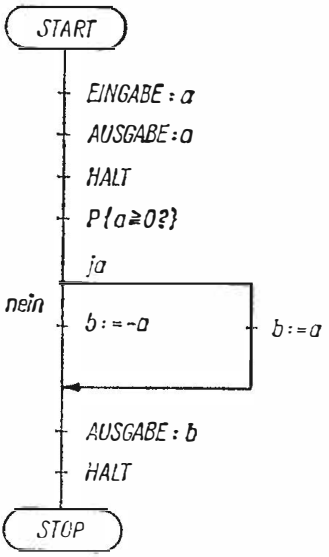


Bild 5

Wir wollen das Programm so aufstellen, daß durch den Nutzer des Programms nur noch der Zahlenwert von a eingegeben zu werden braucht. Alles weitere, also auch die Speicherbelegung, soll der Rechner selbständig ausführen.

Test des Programmablaufplans:

Der vorliegende Programmablaufplan soll für jeden Fall, der bei der Berechnung des Betrages einer Zahl auftreten kann,

getestet werden. Als Zahlenbeispiele werden hierfür die Zahlen 5, 0 und − 3 gewählt.

Programmschritt	Ablauf der Rechnung		
Eingabe: a	5	0	− 3
Ausgabe: a	5	0	− 3
$a \geqq 0$?	ja	ja	nein
$b := -a$	↓	↓	3
$b := a$	5	0	↓
Ausgabe: b	5	0	3

Es kann festgestellt werden, daß bei Befolgung des Programmablaufplans in jedem Fall der richtige Wert für den Betrag der Zahl a ausgegeben wird. Der Programmablaufplan ist also ordnungsgemäß aufgestellt.

Speicherbelegungsplan:

Vor der Rechnung brauchen keine Speicher belegt zu werden.

	Speicher	
	Nummer	Inhalt
Arbeitsspeicher:	0	a
	1	b

Programm:

Da in dem folgenden Programm nur Speicher- und keine Rechenoperationen durchgeführt werden, braucht nur eine Fassung aufgeschrieben zu werden, die sowohl für Rechner mit algebraischer Logik als auch für Rechner mit Umgekehrter Polnischer Notation gilt.

Eingabe	Code	Bemerkung
R/S	13–000	HALT für Eingabe von a
STO 0	64–001 91–002	Speichern von a
R/S	13–003	HALT für Kontrollausgabe von a
$X \geqq 0$	15–004	Entscheidung, ob a positiv oder negativ ist
GOTO 019	14–005	Dieser Sprungbefehl zur Adresse 019 wird nur ausgeführt, wenn $a \geqq 0$ ist. Ist $a < 0$, dann wird das Programm mit dem folgenden Befehl fortgesetzt. Die Zieladresse des Sprunges ist hier bereits eingetragen, sie kann aber erst festgesetzt werden, wenn der Nein-Zweig des Programms programmiert worden ist.
	91–006	
	81–007	
	63–008	
RCL 0	65–●●9	
	91–01●	a
+/−	94–011	$-a = b$
STO 1	64–012	
	81–013	Speichern von b
R/S	13–014	HALT für Ausgabe von b
GOTO 000	14–015	
	91–016	
	91–017	
	91–018	Rücksprung zum Programmanfang
RCL 0	65–019	
	91–020	a
GOTO 012	14–021	
	91–022	
	81–023	
	82–024	Sprung zum Befehl 012, durch den b gespeichert und danach ausgegeben wird

Programmtest:

Der Test des Programms kann mit den im Programmablaufplantest gewählten Beispielen durchgeführt werden. Die Richtigkeit des Programms wird damit gezeigt.

Programmdokumentation:

1. Start des Programms. Startadresse 000.
2. Eingabe der Zahl a. Danach [R/S]-Taste drücken.
3. Beim ersten Halt wird zur Kontrolle die Zahl a ausgegeben. Danach erneut die [R/S]-Taste drücken.
4. Beim nächsten Halt wird der Betrag der Zahl a ausgegeben und damit das Programm beendet.
5. a befindet sich im Speicher 0, $b = |a|$ im Speicher 1.

3.3.2. Ordnen dreier Zahlen *a*, *b* und *c*

Die drei Zahlen a, b und c sollen der Größe nach geordnet und gespeichert werden.
Auch hier handelt es sich wieder um ein Problem, das man als denkender Mensch sicher zunächst nicht einem programmierbaren Taschenrechner anvertrauen würde, denn ein Blick auf die gegebenen drei Zahlen genügt, um sie der Reihe nach von der kleinsten bis zur größten hinschreiben zu können. – Völlig anders liegen die Verhältnisse jedoch, wenn diese drei Zahlen im Verlaufe einer größeren Rechnung als Zwischenergebnisse entstehen, im Rechner gespeichert sind und nun der Größe nach geordnet benötigt werden.
Wenn wir für dieses anscheinend so triviale Problem den Programmablaufplan aufstellen werden und das zugehörige Programm erarbeiten, werden wir sehr viele wertvolle Einblicke in die Vorgehensweise erhalten, die man einschlagen muß, wenn man eine Aufgabe in all ihre kleinsten Einzelschritte zerlegen will. Diese ***Zerlegung*** in alle erforderlichen ***Elementarschritte*** ist notwendig, denn ein Rechner kann von sich aus gar nichts, und wenn man von ihm erwartet, daß er eine Aufgabe für uns löst, so muß man ihm ***jeden Schritt***, den er durchführen soll, bis in die kleinste Kleinigkeit hinein ***vorschreiben***.
Dabei wird man sich bei der Aufstellung von Programm-

ablaufplänen und Programmen über Gedankengänge klarwerden müssen, die einem beim herkömmlichen Lösen von Aufgaben gar nicht mehr bewußt werden, weil einem bestimmte Rechenverfahren und -operationen schon so zur Selbstverständlichkeit geworden sind, daß man gar nicht mehr über Einzelheiten nachzudenken braucht, wenn man sie anwenden will. Dem elektronischen Rechner muß man aber diese Einzelheiten genauestens sagen!

Will man drei beliebige Zahlen a, b und c der Größe nach ordnen, so spielen sich im einzelnen die folgenden geistigen Prozesse ab: Man muß sich zunächst einmal die drei Zahlen einprägen. Das geschieht ohne viel eigenes Zutun in unserem Denkzentrum, dem Gehirn. Im Rechner müssen wir, damit er sich die Zahlen merken kann, drei Speicherplätze reservieren, die wir mit MIN, MID und MAX bezeichnen wollen. Da wir (angeblich) die genaue Reihenfolge der drei Zahlen nicht kennen, wollen wir a, b und c so auf diese Speicherplätze verteilen, wie sie gegeben sind:

$$\text{MAX} := a, \quad \text{MID} := b, \quad \text{MIN} := c.$$

(Zeile 1 bis 9 des Programmablaufplans).

Nun lassen wir den Rechner die Werte von MAX und MID miteinander vergleichen (Zeile 10 des Programmablaufplans), indem wir die Differenz MAX − MID bilden und feststellen, ob sie $\geqq 0$ ist. Ist dies der Fall, so ist $\text{MAX} \geqq \text{MID}$, und wir haben diese beiden Zahlen schon in der richtigen Reihenfolge gespeichert. („Ja"-Zweig nach Zeile 10 des Programmablaufplans). Es kann aber doch ohne weiteres auch sein, daß $\text{MAX} - \text{MID} < 0$ und damit $\text{MAX} < \text{MID}$ ist. In diesem Fall wäre die von uns ursprünglich gewählte Speicherplatzbelegung für MAX und MID nicht sinnvoll. Es empfiehlt sich also, die Inhalte dieser beiden Speicherplätze miteinander zu vertauschen, damit in MAX die größere und in MID die kleinere der beiden Zahlen steht.

Damit stehen wir vor einem neuen, nicht ganz einfach zu lösenden Problem: *Wie vertauscht man die Inhalte zweier Speicherplätze?*

Würde man den Inhalt der Speicherzelle für MID einfach hernehmen und ihn in der Speicherzelle für MAX speichern wollen, so würde uns dadurch der bisherige Inhalt der Speicherzelle für MAX verlorengehen, weil er ja durch den Inhalt von MID „überschrieben" wird.

Folgende Überlegung kann uns weiterhelfen: Wir bringen den Inhalt der Speicherzelle für MID zunächst auf einen Hilfsspeicherplatz, den wir mit HILF bezeichnen wollen (Zeile 11 des Programmablaufplans). Damit ist der Inhalt der Zelle für MID nun zweimal in unserem Rechner vorhanden, und wir können nun ohne Bedenken den Inhalt der Speicherzelle für MAX in den für MID bestimmten Speicher bringen. Nunmehr ist MAX in zwei Speichern des Rechners vorhanden, und zwar in dem ursprünglich für MAX vorgesehenen und in dem für MID. Bringen wir schließlich den Inhalt der Hilfsspeicherzelle HILF in den für MAX vorgesehenen Speicher, so sind die ursprünglichen Inhalte der beiden Speicher für MAX und MID miteinander vertauscht (Zeile 12 und 13).

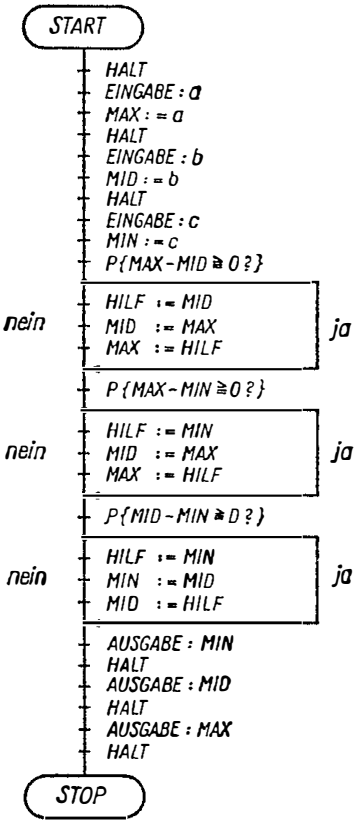


Bild 6

Nunmehr wissen wir also von zwei der drei Zahlen, welche von beiden die größere ist. Was ist aber mit der dritten Zahl, die vorläufig im für MIN vorgesehenen Speicher ein geruhsames Dasein führte? Sie kann doch ohne weiteres noch größer sein als das jetzige MAX. Daher müssen wir nun in analoger Weise MAX und MIN miteinander vergleichen und, falls MAX < MIN sein sollte, die Inhalte der Speicher für MAX und MIN miteinander vertauschen (Zeile 14 bis 17 des Programmablaufplans).
Jetzt steht von den drei gegebenen Zahlen endgültig fest, welche von ihnen die größte ist. Sie ist im Speicher für MAX untergebracht.
Es muß jetzt nur noch ermittelt werden, ob die beiden in MID und MIN gespeicherten Zahlen schon in der richtigen Reihenfolge untergebracht worden sind. Dies kann in genau der gleichen Weise geschehen, wie wir es schon zweimal getan haben (Zeile 18 bis 21 des Programmablaufplans).
Nachdem diese gleichartige Operationsfolge dreimal abgearbeitet worden ist, steht die gesuchte Reihenfolge der drei Zahlen fest, und sie können der Reihe nach ausgegeben werden (ab Zeile 22 des Programmablaufplans).

Test des Programmablaufplans:

Der Programmablaufplan soll an Hand der drei Zahlen 1, 2 und 3, für die auch beliebige andere Zahlen gesetzt werden könnten, auf seine Richtigkeit überprüft werden. Dabei sollen in diese Überprüfung sämtliche sechs Möglichkeiten einbezogen werden, in denen diese drei Zahlen angeordnet sein können.

Programmschritt	Ablauf der Rechnung					
	Fall 1	Fall 2	Fall 3	Fall 4	Fall 5	Fall 6
Eingabe: a	1	1	2	2	3	3
Eingabe: b	2	3	1	3	1	2
Eingabe: c	3	2	3	1	2	1
MAX := a	1	1	2	2	3	3
MID := b	2	3	1	3	1	2
MIN := c	3	2	3	1	2	1
MAX − MID	−1	−2	1	−1	2	1
MAX − MID $\geqq 0$?	nein	nein	ja	nein	ja	ja
HILF := MID	2	3	↓	3	↓	↓
MID := MAX	1	1	↓	2	↓	↓
MAX := HILF	2	3	↓	3	↓	↓
MAX − MIN	−1	1	−1	2	1	2
MAX − MIN $\geqq 0$?	nein	ja	nein	ja	ja	ja
HILF := MIN	3	↓	3	↓	↓	↓
MIN := MAX	2	↓	2	↓	↓	↓
MAX := HILF	3	↓	3	↓	↓	↓
MID − MIN	−1	−1	−1	1	−1	1
MID − MIN $\geqq 0$?	nein	nein	nein	ja	nein	ja
HILF := MIN	2	2	2	↓	2	↓
MIN := MID	1	1	1	↓	1	↓
MID := HILF	2	2	2	↓	2	↓
Ausgabe: MIN	1	1	1	1	1	1
Ausgabe: MID	2	2	2	2	2	2
Ausgabe: MAX	3	3	3	3	3	3

In allen Fällen liefert der Programmablaufplan das richtige Ergebnis. Er kann daher als ordnungsgemäß aufgestellt betrachtet werden.
Dem Leser sei dringend empfohlen, den Verlauf aller sechs Testrechnungen noch einmal genauestens zu verfolgen bzw. noch einmal selbständig auszuführen. Es ist dies eine gute Möglichkeit, seine Fähigkeiten und Fertigkeiten in der Programmablaufplanung zu vervollkommnen.

Speicherbelegungsplan:

Vor dem Beginn der Rechnung brauchen keine Speicher belegt zu werden.

	Speicher	
	Nummer	Inhalt
Arbeitsspeicher:	0	MAX
	1	MID
	2	MIN
	3	HILF

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Pol. Notation		Bemerkungen
Eingabe	Code	Eingabe	Code	
R/S	13–000	R/S	13–000	HALT für Eingabe *a*
STO 0	64–001	STO 0	64–001	
	91–002		91–002	Speichern von *a* (vorläufiges MAX)
R/S	13–003	R/S	13–003	HALT für Eingabe *b*
STO 1	64–004	STO 1	64–004	
	81–005		81–005	Speichern von *b* (vorläufiges MID)
R/S	13–006	R/S	13–006	HALT für Eingabe *c*
STO 2	64–007	STO 2	64–007	
	82–008		82–008	Speichern von *c* (vorläufiges MIN)
RCL 0	65–009	RCL 0	65–009	
	91–010		91–010	MAX
–	85–011			
RCL 1	65–012	RCL 1	65–011	
	81–013		81–012	MID
=	95–014	–	85–013	MAX − MID
X ≧ 0	15–015	X ≧ 0	15–014	Entscheidung, ob MAX ≧ MID
GOTO 032	14–016	GOTO 031	14–015	Sprung zum Befehl 032 bzw. 031, wenn
	91–017		91–016	MAX ≧ MID. Ist
	83–018		83–017	MAX < MID, so
	82–019		81–018	wird dieser Sprungbefehl übergangen und die Rechnung mit Befehl 020 bzw. 019 fortgesetzt.

RCL 1	65–020	RCL i	65–019	
	81–021		81–020	MID
STO 3	64–022	STO 3	64–021	
	83–023		83–022	Speichern von MID auf Hilfsspeicher
RCL 0	65–024	RCL 0	65–023	
	91–025		91–024	MAX
STO 1	64–026	STO 1	64–025	
	81–027		81–026	Speichern von MAX auf Speicherplatz von MID
RCL 3	65–028	RCL 3	65–027	
	83–029		83–028	HILF
STO 0	64–030	STO 0	64–029	
	91–031		91–030	Speichern von HILF auf Speicherplatz von MAX
RCL 0	65–032	RCL 0	65–031	
	91–033		91–032	MAX
–	85–034			
RCL 2	65–035	RCL 2	65–033	
	82–036		82–034	MIN
=	95–037	–	85–035	MAX – MIN
X ≧ 0	15–038	X ≧ 0	15–036	Entscheidung, ob MAX ≧ MIN
GOTO 055	14–039	GOTO 053	14–037	Sprung zum Befehl 055 bzw. 053, wenn
	91–040		91–038	MAX ≧ MIN. Ist
	72–041		72–039	MAX < MIN, so
	72–042		83–040	wird dieser Sprungbefehl übergangen und das Programm mit Befehl 043 bzw. 041 fortgesetzt.
RCL 2	65–043	RCL 2	65–041	
	82–044		82–042	MIN

STO 3	64–045	STO 3	64–043	
	83–046		83–044	Speichern von MIN auf Hilfsspeicher
RCL 0	65–047	RCL 0	65–045	
	91–048		91–046	MAX
STO 2	64–049	STO 2	64–047	
	82–050		82–048	Speichern von MAX auf Speicherplatz von MIN
RCL 3	65–051	RCL 3	65–049	
	83–052		83–050	HILF
STO 0	64–053	STO 0	64–051	
	91–054		91–052	Speichern von HILF auf Speicherplatz von MAX
RCL 1	65–055	RCL 1	65–053	
	81–056		81–054	MID
–	85–057			
RCL 2	65–058	RCL 2	65–055	
	82–059		82–056	MIN
=	95–060	–	85–057	MID – MIN
X ≧ 0	15–061	X ≧ 0	15–058	Entscheidung, ob MID ≧ MIN
GOTO 078	14–062	GOTO 075	14–059	Sprung zum Befehl 078 bzw. 075, wenn
	91–063		91–060	MID ≧ MIN. Ist
	61–064		61–061	MID < MIN, so
	62–065		72–062	wird dieser Sprungbefehl übergangen und das Programm mit Befehl 066 bzw. 063 fortgesetzt.
RCL 2	65–066	RCL 2	65–063	
	82–067		82–064	MIN
STO 3	64–068	STO 3	64–065	
	83–069		83–066	Speichern von MIN auf Hilfsspeicher

RCL 1	65–070	RCL 1	65–067	
	81–071		81–068	MID
STO 2	64–072	STO 2	64–069	
	82–073		82–070	Speichern von MID auf Speicherplatz von MIN
RCL 3	65–074	RCL 3	65–071	
	83–075		83–072	HILF
STO 1	64–076	STO 1	64–073	
	81–077		81–074	Speichern von HILF auf Speicherplatz von MID
RCL 2	65–078	RCL 2	65–075	
	82–079		82–076	MIN
R/S	13–080	R/S	13–077	HALT für Ausgabe von MIN
RCL 1	65–081	RCL 1	65–078	
	81–082		81–079	MID
R/S	13–083	R/S	13–080	HALT für Ausgabe von MID
RCL 0	65–084	RCL 0	65–081	
	91–085		91–082	MAX
R/S	13–086	R/S	13–083	HALT für Ausgabe von MAX
GOTO 000	14–087	GOTO 000	14–084	Rücksprung zum Programmanfang
	91–088		91–085	
	91–089		91–086	
	91–090		91–087	

Programmtest:

Das Programm kann mit den Daten, die für den Test des Programmablaufplans verwendet wurden, auf seine Richtigkeit hin überprüft werden.

Programmdokumentation:

1. Programm starten. Startadresse: 000

2. Eingabe von a. Danach [R/S]-Taste drücken.
3. Eingabe von b. Danach [R/S]-Taste drücken.
4. Eingabe von c. Danach [R/S]-Taste drücken.
5. Beim ersten Halt wird die kleinste Zahl ausgegeben. [R/S]-Taste drücken.
6. Beim nächsten Halt wird die mittlere Zahl ausgegeben. [R/S]-Taste drücken.
7. Beim letzten Halt wird die größte Zahl ausgegeben. Damit ist das Programm abgearbeitet.

3.3.3. Berechnung eines schiefwinkligen Dreiecks

Es soll ein Programm aufgestellt werden, mit dessen Hilfe die fehlende Seite und die beiden fehlenden Winkel in einem schiefwinkligen Dreieck berechnet werden können, wenn von diesem Dreieck zwei Seiten und der von ihnen eingeschlossene Winkel bekannt sind.
Die beiden gegebenen Seiten sollen mit a und b bezeichnet werden, der von ihnen eingeschlossene Winkel sei γ. Aus der ebenen Trigonometrie ist bekannt, daß sich die dritte Seite mit Hilfe des Cosinussatzes berechnen läßt:

$$c = \sqrt{a^2 + b^2 - 2ab \cdot \cos\gamma}\,.$$

Die Seite c ist hieraus eindeutig bestimmbar.
Nun kann mit Hilfe des Sinussatzes einer der fehlenden Winkel ermittelt werden. So erhält man beispielsweise aus

$$\sin\alpha : \sin\gamma = a : c$$

$$\sin\alpha = \frac{a}{c} \cdot \sin\gamma\,,$$

woraus man mit Hilfe der Umkehrfunktion des Sinus unmittelbar den Winkel α erhält:

$$\alpha = \arcsin\left(\frac{a}{c} \cdot \sin\gamma\right).$$

Aus dem Satz über die Winkelsumme im Dreieck folgt schließlich

$$\beta = 180^\circ - \alpha - \gamma\,.$$

Damit sind zunächst einmal alle fehlenden Stücke des Dreiecks formal ermittelt. Da jedoch bei der Berechnung eines arcsin mit Hilfe eines elektronischen Rechners immer nur ein Winkel aus dem Bereich $B = [-90°; +90°]$ ausgegeben wird, der Dreieckswinkel aber auch ohne weiteres zwischen 90° und 180° liegen kann, sind noch zusätzliche Überlegungen erforderlich, damit der richtige Winkel α und damit auch der richtige Winkel β ermittelt wird.
Dazu verwenden wir den Satz, daß der größeren Seite im Dreieck auch stets der größere Winkel gegenüber liegen muß. Es muß also a mit b verglichen werden und auch α mit β. Ist $a \geqq b$ bzw. $a < b$ und gilt für die berechneten Winkel α und β die entsprechende Relation $\alpha \geqq \beta$ bzw. $\alpha < \beta$, so sind die berechneten Winkel richtig. Ist jedoch $a \geqq b$ bzw. $a < b$ und gilt hingegen für die berechneten Winkel die Relation $\alpha \leqq \beta$ bzw. $\alpha > \beta$, so stehen diese Größenverhältnisse der Winkel im Widerspruch zu dem oben erwähnten Satz. In diesem Fall ist der Winkel α durch den Winkel $180° - \alpha$ zu ersetzen und der zugehörige Winkel β aus $\beta = 180° - \alpha - \gamma$ zu ermitteln. Auf diese Weise werden dann die richtigen Seiten-Winkel-Relationen hergestellt.
Der zugehörige *Programmablaufplan* ist in Bild 7 dargestellt. Es fällt auf, daß die Zahl 180, die mehrfach benötigt wird, gleich bei den ersten Eingaben mit im Rechner gespeichert werden soll. Dies geschieht aus gutem Grund, denn wenn man jedesmal, wenn die Zahl 180 benötigt wird, der Reihe nach die Tasten [1], [8] und [0] betätigen müßte, so würde das einen erheblichen Mehraufwand erfordern, als wenn man diese Zahl von Anfang an speichert und sie, wenn sie benötigt wird, aus diesem Speicher abruft. (Damit werden auch weniger Programmschritte benötigt, und der Programmspeicher wird entlastet.) Den Test des Programmablaufplans finden Sie auf S. 63.

Test des Programmablaufplans:

Für die Überprüfung des Programmablaufplans werden drei Testbeispiele ausgewählt, bei denen sich unterschiedliche Lösungswege ergeben:

Testbeispiel 1: $a = 3$ cm, $b = 4$ cm, $\gamma = 90°$,
Testbeispiel 2: $a = 4$ cm, $b = 3$ cm, $\gamma = 90°$,
Testbeispiel 3: $a = 30$ cm, $b = 15$ cm, $\gamma = 20°$.

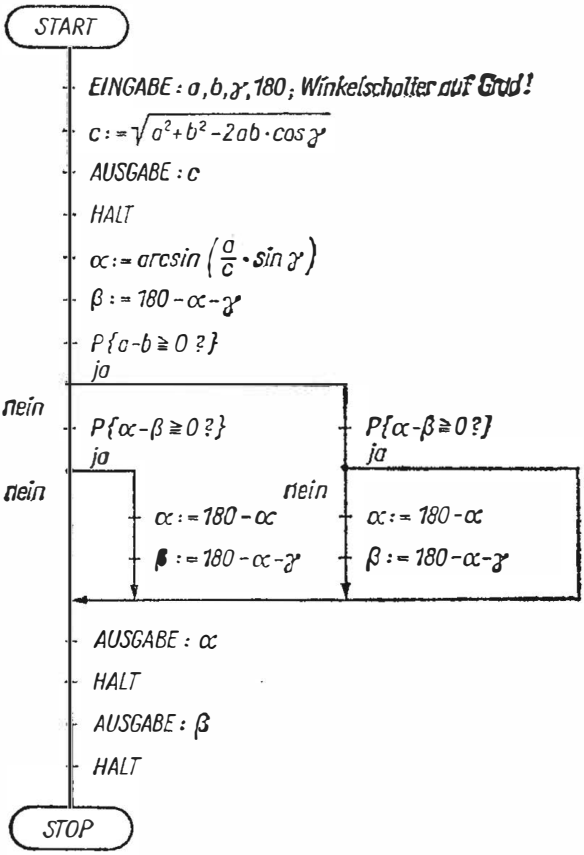

START
EINGABE : a, b, γ, 180; Winkelschalter auf Grad!
c := √(a² + b² − 2ab · cos γ)
AUSGABE : c
HALT
α := arcsin (a/c · sin γ)
β := 180 − α − γ
P{a − b ≧ 0 ?}
ja
nein
P{α − β ≧ 0 ?}
ja
nein
α := 180 − α
β := 180 − α − γ
P{α − β ≧ 0 ?}
ja
nein
α := 180 − α
β := 180 − α − γ
AUSGABE : α
HALT
AUSGABE : β
HALT
STOP

Bild 7

Programmschritt	Ablauf der Rechnung für		
	Test-beispiel 1	Test-beispiel 2	Test-beispiel 3
Eingabe: a	3	4	30
Eingabe: b	4	3	15
Eingabe: γ	90	90	20
$c :=$ $\sqrt{a^2 + b^2 - 2ab \cdot \cos\gamma}$	5	5	16.71
Ausgabe: c	5	5	16.71
$\alpha := \arcsin\left(\frac{a}{c} \cdot \sin\gamma\right)$	36.87	53.13	37.88
$\beta := 180 - \alpha - \gamma$	53.13	36.87	122.12
$a - b$	− 1	1	15
$a - b \geqq 0$?	nein	ja	ja
$\alpha - \beta$	− 16.26	16.26	− 84.34
$\alpha - \beta \geqq 0$?	nein	ja	nein
$\alpha := 180 - \alpha$			142.12
$\beta := 180 - \alpha - \gamma$			17.88
Ausgabe: α	36.87	53.13	142.12
Ausgabe: β	53.13	36.87	17.88

In sämtlichen drei Testbeispielen erweist sich der Programmablaufplan als richtig.

Der Leser führe zur Übung den Test des Programmablaufplans noch einmal selbständig für das Beispiel

$$a = 4 \text{ cm}, \qquad b = 4 \text{ cm}, \qquad \gamma = 60°$$

durch. Bei konsequenter Befolgung des Programmablaufplans muß sich das Resultat

$$c = 4 \text{ cm}, \qquad \alpha = \beta = 60°$$

ergeben.

Speicherbelegungsplan:

	Speicher	
	Nummer	Inhalt
Vor Beginn der Rechnung zu belegen:	0	a
	1	b
	2	γ
	3	180
Arbeitsspeicher:	4	c
	5	α
	6	β

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Pol. Notation		Bemerkungen
Eingabe	Code	Eingabe	Code	
R/S	13–000	R/S	13–000	HALT für die Eingabe der Werte a, b, γ und 180
RCL 0	65–001	RCL 0	65–001	
	91–002		91–002	a
×	74–003			
RCL 0	65–004	RCL 0	65–003	
	91–005		91–004	a
+	84–006	×	74–005	a^2
F	24–007			
(	61–008			
RCL 1	65–009	RCL 1	65–006	
	81–010		81–007	b
×	74–011			
RCL 1	65–012	RCL 1	65–008	
	81–013		81–009	b
F	24–014			
)	62–015	×	74–010	b^2
–	85–016	+	84–011	$a^2 + b^2$
F	24–017			
(	61–018			
2	82–019	2	82–012	2
×	74–020			
RCL 0	65–021	RCL 0	65–013	
	91–022		91–014	a

×	74–023	×	74–015	$2a$
RCL 1	65–024	RCL 1	65–016	
	81–025		81–017	b
×	74–026	×	74–018	$2ab$
RCL 2	65–027	RCL 2	65–019	
	82–028		82–020	γ
cos	32–029	cos	32–021	$\cos\gamma$
F	24–030			
)	62–031	×	74–022	$2ab \cdot \cos\gamma$
=	95–032	−	84–023	$a^2 + b^2 - 2ab \cdot \cos\gamma$
		F	24–024	
$\sqrt{\ }$	34–033	$\sqrt{\ }$	62–025	c
STO 4	64–034	STO 4	64–026	
	71–035		71–027	Speichern von c
R/S	13–036	R/S	13–028	HALT, Ausgabe von c
RCL 0	65–037	RCL 0	65–029	
	91–038		91–030	a
÷	75–039			
RCL 4	65–040	RCL 4	65–031	
	71–041		71–032	c
×	74–042	÷	75–033	a/c
RCL 2	65–043	RCL 2	65–034	
	82–044		82–035	γ
sin	31–045	sin	31–036	$\sin\gamma$
=	95–046	×	74–037	$\frac{a}{c} \cdot \sin\gamma$
F	24–047	F	24–038	
$\sin^{-1}$	31–048	$\sin^{-1}$	31–039	α
STO 5	64–049	STO 5	64–040	
	72–050		72–041	Speichern von α

+/−	94–051	+/−	94–042	$-\alpha$
+	84–052			
RCL 3	64–053	RCL 3	64–043	
	83–054		83–044	180
−	85–055	+	84–045	$180-\alpha$
RCL 2	65–056	RCL 2	65–046	
	82–057		82–047	γ
=	95–058	−	85–048	$180-\alpha-\gamma=\beta$
STO 6	64–059	STO 6	64–049	
	73–060		73–050	Speichern von β
GOTO	14–061	GOTO	14–051	
100	81–062	100	81–052	
	91–063		91–053	
	91–064		91–054	Sprung zum Befehl 100, bei dem die Endauswertung der bisherigen Ergebnisse beginnt
RCL 0	65–100	RCL 0	65–100	
	91–101		91–101	a
−	85–102			
RCL 1	65–103	RCL 1	65–102	
	81–104		81–103	b
=	95–105	−	85–104	$a-b$
X ≧ 0	15–106	X ≧ 0	15–105	Entscheidung, ob $a \geqq b$
GOTO 132	14–107	GOTO 130	14–106	Sprung nach Befehl 132 bzw. 130, wenn $a \geqq b$. Ist $a < b$, so wird dieser Sprungbefehl nicht ausgeführt, und das Programm mit dem Befehl 111 bzw. 110 weitergeführt.
	81–108		81–107	
	83–109		83–108	
	82–110		91–109	

RCL 5	65–111	RCL 5	65–110	
	72–112		72–111	α
–	85–113			
RCL 6	65–114	RCL 6	65–112	
	73–115		73–113	β
=	95–116	–	85–114	$\alpha - \beta$
X ≧ 0	15–117	X ≧ 0	15–115	Entscheidung, ob $\alpha \geqq \beta$
GOTO 143	14–118	GOTO 140	14–116	Sprung nach Befehl 143 bzw. 140, wenn
	81–119		81–117	$\alpha \geqq \beta$. Ist $\alpha < \beta$, so
	71–120		71–118	wird dieser Sprung-
	83–121		91–119	befehl nicht ausgeführt, und das Programm wird mit dem Befehl 122 bzw. 120 weitergeführt.
RCL 5	65–122	RCL 5	65–120	
	72–123		72–121	α
R/S	13–124	R/S	13–122	HALT, Ausgabe von α
RCL 6	65–125	RCL 6	65–123	
	73–126		73–124	β
R/S	13–127	R/S	13–125	HALT, Ausgabe von β
GOTO 000	14–128	GOTO 000	14–126	
	91–129		91–127	
	91–130		91–128	
	91–131		91–129	Rücksprung zum Programmanfang
RCL 5	65–132	RCL 5	65–130	
	72–133		72–131	α
–	85–134			
RCL 6	65–135	RCL 6	65–132	
	73–136		73–133	β

[=]	95–137	[−]	85–134	$\alpha - \beta$
[X ≧ 0]	15–138	[X ≧ 0]	15–135	Entscheidung, ob $\alpha \geqq \beta$
[GOTO 122]	14–139	[GOTO 120]	14–136	Sprung zum Befehl 122 bzw. 120, wenn
	81–140		81–137	$\alpha \geqq \beta$. Ist $\alpha < \beta$,
	82–141		82–138	dann wird dieser
	82–142		91–139	Sprungbefehl nicht beachtet und das Programm mit dem Befehl 143 bzw. 140 weitergeführt
[RCL 3]	65–143	[RCL 3]	65–140	
	83–144		83–141	180
[−]	85–145			
[RCL 5]	65–146	[RCL 5]	65–142	
	72–147		72–143	α
[=]	95–148	[−]	85–144	$180 - \alpha$
[STO 5]	64–149	[STO 5]	64–145	
	72–150		72–146	Speichern des neuen α
[RCL 3]	65–151	[RCL 3]	65–147	
	83–152		83–148	180
[−]	85–153			
[RCL 5]	65–154	[RCL 5]	65–149	
	72–155		72–150	α
[−]	85–156	[−]	85–151	$180 - \alpha$
[RCL 2]	65–157	[RCL 2]	65–152	
	82–158		82–153	γ
[=]	95–159	[−]	85–154	$\beta = 180 - \alpha - \gamma$
[STO 6]	64–160	[STO 6]	64–155	
	73–161		73–156	Speichern des neuen β

GOTO 122	14–162	GOTO 120	14–157	
	81–163		81–158	
	82–164		82–159	
	82–165		91–160	Sprung zur Ausgabe von α und β

Programmtest:

Zum Test des Programms wird empfohlen, die Testbeispiele zu verwenden, die für die Überprüfung des Programmablaufplans gewählt wurden.

Programmdokumentation:

1. Gewünschte Rechengenauigkeit einstellen.
   Winkelschalter auf Gradmaß stellen!
2. Programm starten. Startadresse: 000
3. Eingabe der beiden gegebenen Seiten des Dreiecks auf die Speicherplätze 0 und 1, des gegebenen Winkels auf den Speicherplatz 2 und der Zahl 180 auf den Speicherplatz 3.
   R/S -Taste drücken!
4. Beim ersten Halt wird die fehlende Dreiecksseite ausgegeben. Danach R/S -Taste drücken!
5. Bei den nächsten beiden Halts werden die beiden Dreieckswinkel ausgegeben. Nach jedem Halt ist die R/S -Taste zu drücken.
6. Wenn der zur Verfügung stehende Rechner weniger als 100 Speicherplätze besitzt, kann das Programm in zwei voneinander unabhängig arbeitende Teilprogramme zerlegt werden. Das erste Teilprogramm geht bis zum Befehl 064 bzw. 065. Danach ist ab Adresse 000 (statt 100) das zweite Teilprogramm einzugeben. Eine Neubelegung der Speicherplätze ist dabei nicht erforderlich.

3.3.4. Lineare Gleichungssysteme mit zwei Unbekannten

In 3.2.1. haben wir ein Programm aufgestellt, um Gleichungssysteme mit zwei Unbekannten lösen zu können. Da-

bei mußte zunächst der Fall, daß der Nenner bei der Berechnung von x und y Null wird, unberücksichtigt gelassen werden. So konnte es geschehen, daß bei der Abarbeitung eine Fehleranzeige auftrat, und es war nicht klar, ob diese Fehleranzeige dadurch entstand, daß die beiden gegebenen Gleichungen einander widersprechen bzw. linear abhängig voneinander sind oder daß irgendein anderer Fehler im Verlaufe des Programms aufgetreten sein könnte.
Es soll hier eine zweite Programmversion vorgestellt werden, bei der das Gleichungssystem gelöst wird, wenn es lösbar ist, und bei dem andererseits aber eine eindeutige Auskunft gegeben wird, wenn das System nicht eindeutig lösbar sein sollte.
Wenn die beiden Gleichungen eines linearen Gleichungssystems *voneinander abhängig* sind bzw. *einander widersprechen,* dann hat die Determinante

$$N = a_1 b_2 - a_2 b_1$$

den Wert Null. Wir brauchen demnach das in 3.2.1. aufgestellte Programm nur durch die Abfrage zu ergänzen, ob dieser Fall eintritt. Sollte dies der Fall sein, so soll der Rechner als Kennzeichen dafür, daß das Gleichungssystem nicht lösbar ist, die Ziffernfolge

1234.5678

ausgeben.
Der *Programmablaufplan* für diese erweiterte Programmversion ist in Bild 8 dargestellt.

Test des Programmablaufplans:

Der aufgestellte Programmablaufplan soll an Hand der beiden linearen Gleichungssysteme

$$\begin{array}{rr} 2x + 3y = 12 & \qquad 2x + 3y = 12 \\ 2x - 3y = 0 & \qquad 4x + 6y = 10 \end{array}$$

überprüft werden, von denen das erste System die beiden Lösungen

$$x = 3,\ y = 2$$

besitzt, während das zweite System nicht lösbar ist, weil die beiden Gleichungen einander widersprechen.

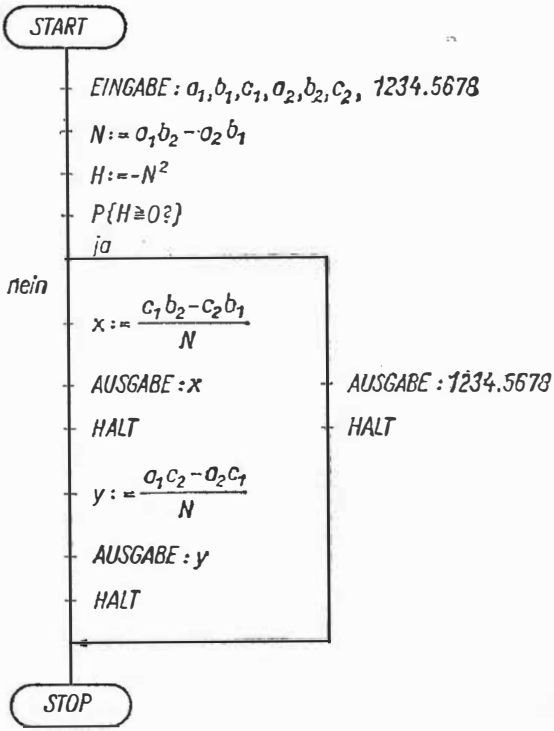

START
EINGABE: $a_1, b_1, c_1, a_2, b_2, c_2$, 1234.5678
$N := a_1 b_2 - a_2 b_1$
$H := -N^2$
$P\{H \geq 0?\}$
ja
nein
$x := \frac{c_1 b_2 - c_2 b_1}{N}$
AUSGABE: x
HALT
$y := \frac{a_1 c_2 - a_2 c_1}{N}$
AUSGABE: y
HALT
AUSGABE: 1234.5678
HALT
STOP

Bild 8

Programmschritt	Ablauf für	
	$2x+3y=12$ $2x-3y=0$	$2x+3y=12$ $4x+6y=10$
Eingabe: a_1	2	2
b_1	3	3
c_1	12	12
a_2	2	4
b_2	-3	6
c_2	0	10
$N := a_1 b_2 - a_2 b_1$	$-6-6=-12$	$12-12=0$
$H := -N^2$	$-(-12)^2=$ $=-144$	$0^2=0$
$H \geqq 0$?	nein	ja
$x := \frac{c_1 b_2 - c_2 b_1}{N}$	$\frac{-36}{-12}=+3$	↓
Ausgabe: x	3	
$y := \frac{a_1 c_2 - a_2 c_1}{N}$	$\frac{-24}{-12}=2$	
Ausgabe: y	2	
Ausgabe: 1234.5678		1234.5678

Die Testrechnungen zeigen, daß der Programmablaufplan die richtigen Ergebnisse liefert.

Da auf den für unsere Programme zugrunde gelegten Rechnern nur die Abfragemöglichkeit $\boxed{X \geqq 0}$ besteht, bereitet die Entscheidung, ob der Nenner gleich Null ist, die Schwierigkeiten, die im Abschnitt 2. bereits angedeutet worden sind. Um feststellen zu können, ob $N = 0$ ist, bilden wir den Ausdruck $H = 0 - N^2$ und untersuchen, ob $H \geqq 0$. Ist nämlich $N \neq 0$, so wird $H < 0$. Nur in dem Fall, daß $N = 0$ ist, ist auch die Bedingung $H \geqq 0$ erfüllt.

Speicherbelegungsplan:

	Speicher	
	Nummer	Inhalt
Vor Beginn der Rechnung zu belegen:	0	a_1
	1	b_1
	2	c_1
	3	a_2
	4	b_2
	5	c_2
	6	1234.5678
Arbeitsspeicher:	7	N

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Poln. Notation		Bemerkung
Eingabe	Code	Eingabe	Code	
R/S	13–000	R/S	13–000	HALT für Eingabe der Koeffizienten a_1, b_1, c_1, a_2, b_2, c_2 und der Kennzahl 1234.5678
RCL 0	65–001	RCL 0	65–001	
	91–002		91–002	a_1
×	74–003			
RCL 4	65–004	RCL 4	65–003	
	71–005		71–004	b_2
−	85–006	×	74–005	$a_1 b_2$
F	24–007			
(	61–008			
RCL 3	65–009	RCL 3	65–006	
	83–010		83–007	a_2
×	74–011			
RCL 1	65–012	RCL 1	65–008	
	81–013		81–009	b_1
F	24–014			
)	62–015	×	74–010	$a_2 b_1$
=	95–016	−	85–011	$a_1 b_2 - a_2 b_1 = N$
STO 7	64–017	STO 7	64–012	
	61–018		61–013	Speichern von N
×	74–019	ENTER	95–014	N
RCL 7	65–020			
	61–021			N

=	95–022	×	74–015	N^2
–	85–023			
0	91–024	0	91–016	0
X↔Y	54–025	X↔Y	54–017	
=	95–026	–	85–018	$H = 0 - N^2$
X ≧ 0	15–027	X ≧ 0	15–019	Überprüfung, ob $N = 0$ ist
GOTO 076	14–028 91–029 61–030 73–031	GOTO 058	14–020 91–021 72–022 62–023	Dieser Sprungbefehl zur Ausgabe der Kennzahl 1234.5678 wird nur ausgeführt, wenn $H \geqq 0$. Ist $H < 0$, so wird das Programm mit dem Befehl 032 bzw. 024 fortgesetzt.
RCL 2	65–032	RCL 2	65–024	
	82–033		82–025	c_1
×	74–034			
RCL 4	65–035	RCL 4	65–026	
	71–036		71–027	b_2
–	85–037	×	74–028	$c_1 b_2$
F	24–038			
(	61–039			
RCL 5	65–040	RCL 5	65–029	
	72–041		72–030	c_2
×	74–042			
RCL 1	65–043	RCL 1	65–031	
	81–044		81–032	b_1
F	24–045			
)	62–046	×	74–033	$c_2 b_1$

÷	75–047	−	85–034	$c_1b_2 - c_2b_1$
RCL 7	65–048	RCL 7	65–035	
	61–049		61–036	N
=	95–050	÷	75–037	$x = (c_1b_2 - c_2b_1)/N$
R/S	13–051	R/S	13–038	HALT für Ausgabe x
RCL 0	65–052	RCL 0	65–039	
	91–053		91–040	a_1
×	74–054			
RCL 5	65–055	RCL 5	65–041	
	72–056		72–042	c_2
−	85–057	×	74–043	a_1c_2
F	24–058			
(	61–059			
RCL 3	65–060	RCL 3	65–044	
	83–061		81–045	a_2
×	74–062			
RCL 2	65–063	RCL 2	65–046	
	82–064		82–047	c_1
F	24–065			
)	62–066	×	74–048	a_2c_1
÷	75–067	−	85–049	$a_1c_2 - a_2c_1$
RCL 7	65–068	RCL 7	65–050	
	61–069		61–051	N
=	95–070	÷	75–052	$y = (a_1c_2 - a_2c_1)/N$
R/S	13–071	R/S	13–053	HALT für Ausgabe y
GOTO 000	14–072	GOTO 000	14–054	Rücksprung zum Programmanfang

	91–073		91–055	
	91–074		91–056	
	91–075		91–057	
RCL 6	65–076	RCL 6	65–058	
	73–077		73–059	1234.5678
R/S	13–078	R/S	13–060	HALT für Ausgabe von 1234.5678 als Kennzeichen dafür, daß das Gleichungssystem nicht eindeutig lösbar bzw. unlösbar ist
GOTO 000	14–079	GOTO 000	14–061	
	91–080		91–062	
	91–081		91–063	
	91–082		91–064	Rücksprung zum Programmanfang

Programmtest:

Testet man das Programm mit dem bei der Überprüfung des Programmablaufplans verwendeten Gleichungssystem

$$2x + 3y = 12$$

$$2x - 3y = 0,$$

so ergeben sich die beiden Lösungen

$$x = 3 \text{ und } y = 2.$$

Verwendet man das Beispiel

$$2x + 3y = 12$$

$$4x + 6y = 10,$$

so erscheint in der Zahlenanzeige der Wert 1234.5678 als Zeichen dafür, daß das Gleichungssystem nicht lösbar bzw. nicht eindeutig lösbar ist.

Programmdokumentation:

1. Geforderte Rechengenauigkeit einstellen.
2. Programm starten. Startadresse: 000.
3. Eingabe folgender Werte:

   a_1 auf Speicher 0 — b_1 auf Speicher 1
   c_1 auf Speicher 2 — a_2 auf Speicher 3
   b_2 auf Speicher 4 — c_2 auf Speicher 5
   1234.5678 auf Speicher 6.

4. R/S-Taste drücken.
5. Erscheint in der Zahlenanzeige die Ziffernfolge 1234.5678, so sind die beiden Gleichungen linear abhängig, oder sie widersprechen einander.
   Erscheint ein anderer Wert, so ist dies der Wert der Lösung für x. — R/S-Taste drücken.
6. Beim nächsten Halt erscheint der Wert der Lösung y.

3.3.5. Quadratische Gleichungen

Es soll ein Programm aufgestellt werden, mit dessen Hilfe die reellen Lösungen der quadratischen Gleichung

$$a x^2 + b x + c = 0$$

berechnet werden können.
Die beiden Lösungen x_1 und x_2 der quadratischen Gleichung

$$a x^2 + b x + c = 0$$

sind

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Daraus ergibt sich der folgende Lösungsweg für unser Programm:
Wenn der Radikand

$$D = b^2 - 4ac$$

negativ ist, dann gibt es ***keine reellen Lösungen*** der gegebenen quadratischen Gleichung. Das Programm soll so beschaffen sein, daß es den Nutzer darauf hinweist, daß keine Lösung vorhanden ist, indem es im Fall $D < 0$ die Ziffernfolge 8888.8888 ausgibt.

Ist $D \geqq 0$, dann kann wie folgt gerechnet werden: es wird zunächst

$$x_1 = \frac{-b + \sqrt{D}}{2a}$$

ermittelt. Nach der Ausgabe von x_1 muß die zweite Lösung bestimmt werden. Dazu verwenden wir nicht noch einmal die doch recht aufwendige Lösungsformel, sondern wir wenden den Wurzelsatz von VIETA an (vgl. [1]) und bilden mit weniger Rechenaufwand

$$x_2 = -\frac{b}{a} - x_1 .$$

(Man hätte auch die Beziehung $x_1 \cdot x_2 = \frac{c}{a}$ verwenden können, doch hätte diese Beziehung zu einer Fehlermeldung geführt, wenn $x_1 = 0$ ist.)
Von einer Sonderbehandlung der beiden Spezialfälle

$$a x^2 + c = 0 \text{ und } a x^2 + b x = 0$$

wird abgesehen, da sich diese beiden Fälle mit dem hier angegebenen Lösungsverfahren mit erfassen lassen.
Der Programmablaufplan für die Lösung einer quadratischen Gleichung ist in Bild 9 angegeben.
Es sei hier darauf hingewiesen, daß bei dem hier vorgesehenen Programmablauf vom Rechner auch die Zahl 8888.8888 als Zeichen für komplexe Lösungen ausgegeben wird, wenn $D = 0$, d.h., wenn eine *reelle Doppellösung* auftritt. Diese Unzulänglichkeit des Programms soll hier in Kauf genommen werden, um den Programmaufbau bei diesen einführenden Beispielen nicht unnötig durch zusätzliche logische Überlegungen zu überlasten.
Wenn der Leser die ausreichende Übung im Programmieren hat, kann er sich selbst überlegen, wie diese Unzulänglichkeit beseitigt werden könnte.

Test des Programmablaufplans:

Die Richtigkeit des Programmablaufplans soll an Hand der beiden quadratischen Gleichungen

$$x^2 - 6x + 8 = 0 \text{ und } x^2 - 2x + 5 = 0$$

überprüft werden, von denen die erste die beiden Lösungen

$$x_1 = 2 \text{ und } x_2 = 4$$

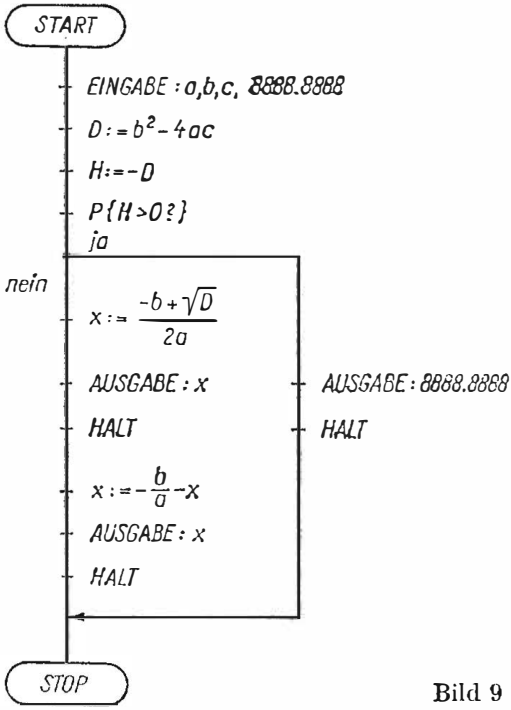


Bild 9

besitzt, während die Lösungen der zweiten Gleichung komplex sind.

Programmschritt	Ablauf für	
	$x^2 - 6x + 8 = 0$	$x^2 - 2x + 5 = 0$
a	1	1
b	-6	-2
c	8	5
$D := b^2 - 4ac$	$36 - 4 \cdot 8 = 4$	$4 - 4 \cdot 5 = -16$
$H := -D$	-4	16
$H > 0$?	nein	ja
$x := \frac{-b + \sqrt{D}}{2a}$	4	
Ausgabe: x	4	
$x := -\frac{b}{a} - x$	2	
Ausgabe: x	2	
Ausgabe: 8888.8888		8888.8888

Es kann festgestellt werden, daß der Programmablaufplan für beide möglichen Fälle, die bei quadratischen Gleichungen auftreten können, die richtigen Ergebnisse liefert.

Speicherbelegungsplan:

	Speicher	
	Nummer	Inhalt
Vor Rechenbeginn zu belegen:	0	a
	1	b
	2	c
Arbeitsspeicher:	3	8888.8888
	4	D
	5	x_1

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Poln. Notation		Bemerkung
Eingabe	Code	Eingabe	Code	
R/S	13–000	R/S	13–000	HALT für Eingabe der Koeffizienten a, b und c und der Kennziffer 8888.8888 für komplexe Lösungen der Gleichung
RCL 1	65–001	RCL 1	65–001	
	81–002		81–002	b
×	74–003	ENTER	95–003	
RCL 1	65–004			
	81–005			b
−	85–006	×	74–004	b^2
F	24–007			
(	61–008			
4	71–009	4	71–005	4
×	74–010			
RCL 0	65–011	RCL 0	65–006	
	91–012		91–007	a
×	74–013	×	74–008	$4a$
RCL 2	65–014	RCL 2	65–009	
	82–015		82–010	c
F	24–016	×	74–011	
)	62–017			$4ac$
=	95–018	−	85–012	$b^2 - 4ac = D$
STO 4	64–019	STO 4	64–013	
	71–020		71–014	Speichern von D

[+/−]	94–021	[+/−]	94–015	$-D=H$
[X ≧ 0]	15–022	[X ≧ 0]	15–016	Überprüfen, ob $H \geqq 0$, d.h., $D \leqq 0$
[GOTO 058]	14–023	[GOTO 051]	14–017	Sprung zur Ausgabe von 8888.8888,
	91–024		91–018	wenn $D \leqq 0$. Ist
	72–025		72–019	$D > 0$, so wird das
	62–026		81–020	Programm mit dem Befehl 027 bzw. 021 fortgesetzt.
[RCL 1]	65–027	[RCL 1]	65–021	
	81–028		81–022	b
[−]	85–029			
[RCL 4]	65–030	[RCL 4]	65–023	
	71–031		71–024	D
		[F]	24–025	
[√‾]	34–032	[√‾]	62–026	$\sqrt{D}$
[X↔Y]	54–033	[X↔Y]	54–027	
[÷]	75–034	[−]	85–028	$-b+\sqrt{D}$
[2]	82–035	[2]	82–029	2
[÷]	75–036			
[RCL 0]	65–037	[RCL 0]	65–030	
	91–038		91–031	a
		[×]	74–032	$2a$
[=]	95–039	[÷]	75–033	$(-b+\sqrt{D})/(2a)=x_1$
[STO 5]	64–040	[STO 5]	64–034	
	72–041		72–035	Speichern von x_1
[R/S]	13–042	[R/S]	13–036	HALT für Ausgabe von x_1
[RCL 1]	65–043	[RCL 1]	65–037	
	81–044		81–038	b
[÷]	75–045			

RCL 0	65–046	RCL 0	65–039	
	91–047		91–040	a
+/–	94–048	+/–	94–041	$-a$
–	85–049	÷	75–042	$-b/a$
RCL 5	65–050	RCL 5	65–043	
	72–051		72–044	x_1
=	95–052	–	85–045	$x_2 = -b/a - x_1$
R/S	13–053	R/S	13–046	HALT für Ausgabe von x_2
GOTO 000	14–054	GOTO 000	14–047	
	91–055		91–048	
	91–056		91–049	
	91–057		91–050	Rücksprung zum Programmanfang
RCL 3	65–058	RCL 3	65–051	
	83–059		83–052	8888.8888
R/S	13–060	R/S	13–053	HALT für Ausgabe von 8888.8888 als Zeichen dafür, daß die quadratische Gleichung komplexe Lösungen besitzt
GOTO 000	14–061	GOTO 000	14–054	
	91–062		91–055	
	91–063		91–056	
	91–064		91–057	Sprung zum Programmanfang

Programmtest:
Mit den beim Test des Programmablaufplans verwendeten Beispielen ergeben sich die dort gefundenen Zahlenwerte für die Lösungen der quadratischen Gleichungen.

Programmdokumentation:
1. Einstellen der geforderten Rechengenauigkeit

2. Start des Programms. Startadresse: 000
3. Eingabe der Koeffizienten der quadratischen Gleichung: Koeffizient des quadratischen Gliedes in Speicher 0, Koeffizient des linearen Gliedes in Speicher 1 und Absolutglied in Speicher 2.
   Eingabe der Ziffernfolge 8888.8888 in Speicher 3
4. R/S-Taste drücken!
5. Erscheint beim ersten HALT die Ziffernfolge 8888.8888, so besitzt die quadratische Gleichung keine reelle Lösung. Die Rechnung ist damit beendet.
   Erscheint ein anderer Zahlenwert, so stellt dieser die erste Lösung der quadratischen Gleichung dar. In diesem Fall ist erneut die R/S-Taste zu drücken. Beim nächsten HALT erscheint die zweite Lösung der Gleichung.

3.3.6. Rechenoperationen mit komplexen Zahlen

Gegeben seien die beiden komplexen Zahlen

$$z_1 = a + \mathrm{i}b \text{ und } z_2 = c + \mathrm{i}d.$$

Mit Hilfe eines Programms sollen diese beiden Zahlen wahlweise addiert, subtrahiert, multipliziert oder dividiert werden können.
Da elektronische Rechner zunächst einmal nur mit reellen Größen rechnen können, müssen die Real- und die Imaginärteile der beiden Zahlen z_1 und z_2 getrennt eingegeben werden. Die durchzuführenden Rechnungen sind verhältnismäßig einfach, und zwar ist

$$z_1 + z_2 = (a + c) + \mathrm{i} \cdot (b + d)$$
$$z_1 - z_2 = (a - c) + \mathrm{i} \cdot (b - d)$$
$$z_1 \cdot z_2 = (ac - bd) + \mathrm{i} \cdot (ad + bc)$$
$$z_1 : z_2 = \frac{ac + bd}{c^2 + d^2} + \mathrm{i} \cdot \frac{bc - ad}{c^2 + d^2}$$

(vgl. [1]).

Es muß dem Rechner nur noch mitgeteilt werden, welche der vier Grundrechenoperationen mit diesen beiden Zahlen durchgeführt werden soll. Dazu führen wir eine Kennzahl K

ein, die die vier Werte 1, 2, 3 und 4 annehmen kann, und es soll

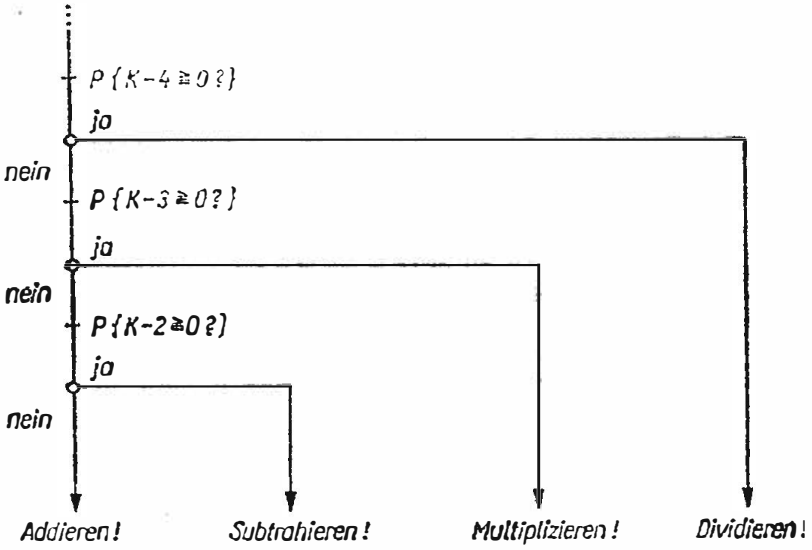
$K = 1$ sein, wenn die Zahlen addiert,

$K = 2$ sein, wenn die Zahlen subtrahiert,

$K = 3$ sein, wenn die Zahlen multipliziert und

$K = 4$ sein, wenn die Zahlen dividiert werden sollen.

Das Programm braucht dann nur so aufgestellt zu werden, daß der Rechner zunächst einmal feststellt, welche Kennziffer für die auszuführende Rechenoperation eingegeben wurde. Dies kann mit Hilfe einer sukzessiven Abfrage erfolgen:



Der ausführliche *Programmablaufplan* ist in Bild 10 dargestellt.

Test des Programmablaufplans:

Der Programmablaufplan soll hier nicht in seiner Gesamtheit überprüft werden, sondern wir wollen nur die kritischen Stellen des Rechenablaufs kontrollieren. (Daß die für die vier Grundrechenarten mit komplexen Zahlen angegebenen Formeln richtig ausgewählt wurden, wird als selbstverständlich vorausgesetzt.)

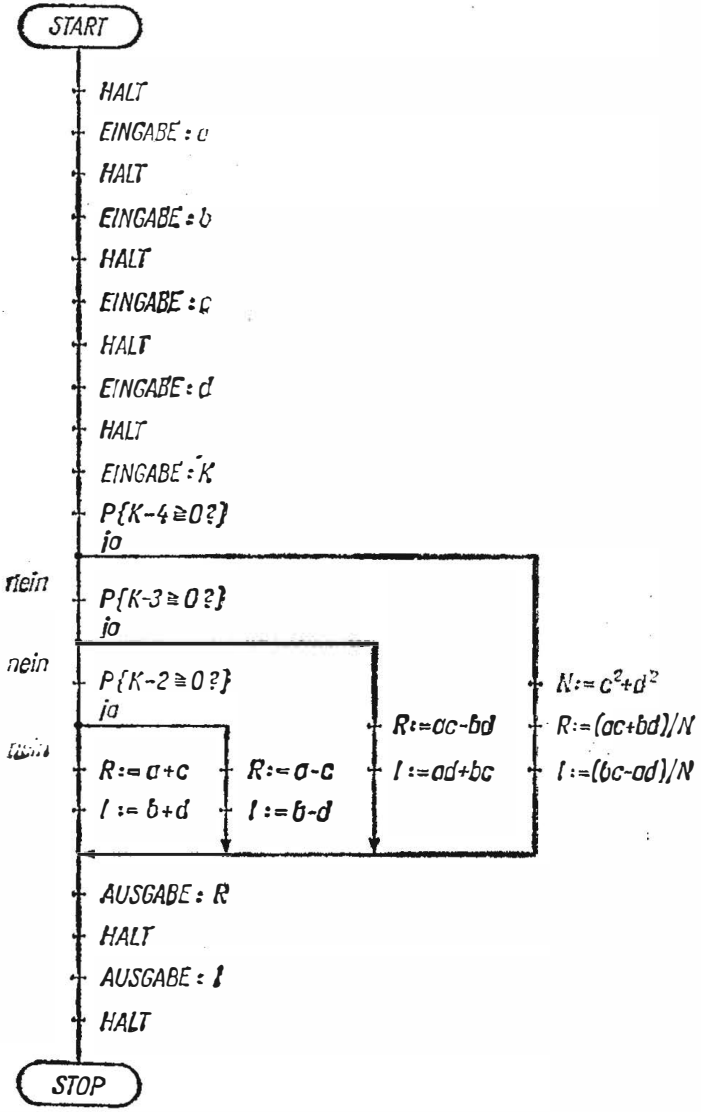

START
HALT
EINGABE : a
HALT
EINGABE : b
HALT
EINGABE : c
HALT
EINGABE : d
HALT
EINGABE : K
P{K-4≧0?}
ja
nein
P{K-3≧0?}
ja
nein
P{K-2≧0?}
ja
nein
R := a+c
I := b+d
R := a-c
I := b-d
R := ac-bd
I := ad+bc
N := c²+d²
R := (ac+bd)/N
I := (bc-ad)/N
AUSGABE : R
HALT
AUSGABE : I
HALT
STOP

Bild 10

Als *kritische Stellen eines Programmablaufplans* müssen diejenigen Stellen angesehen werden, an denen sich der Rechengang auf Grund bestimmter Bedingungen *verzweigen* kann, also an den „*Verzweigungsstellen*". – Maßgebend für derartige Verzweigungen ist in unserem Fall die den einzelnen Grundrechenarten zugeordnete Kennziffer K.

Ist $K = 4$, so ist die bei der ersten Verzweigungsstelle gestellte Frage, ob $K - 4 \geqq 0$ ist, mit „ja" zu beantworten, und die weitere Rechnung erfolgt nach dem rechts abgehenden „Ja"-Zweig. Es wird also, wie erwartet, die Division durchgeführt.

Ist $K < 4$, so kann als nächste Möglichkeit $K = 3$ sein. Hier würde die Anwort auf die zweite Entscheidungsfrage „ja" lauten, und die weitere Rechnung müßte in dem Zweig ablaufen, in dem die Multiplikation programmiert ist.

Ist $K < 3$, so verbleiben nur noch die beiden Möglichkeiten $K = 2$ und $K = 1$. Im Fall $K = 2$ wird die letzte Entscheidungsfrage mit „ja" beantwortet, so daß die weitere Rechnung in den Zweig geleitet wird, in dem die Subtraktion durchgeführt wird. Ist dagegen $K = 1$, so lautet die Antwort auf die letzte Entscheidungsfrage „nein", und der weitere Rechenablauf ist durch den letzten Teilzweig, in dem die Addition durchgeführt wird, bestimmt.

Somit kann festgestellt werden, daß der Rechner in Abhängigkeit von der eingegebenen Kennzahl für die durchzuführende Rechenoperation die jeweils gewünschten Ergebnisse liefern muß.

Speicherbelegungsplan:

Vor Beginn der Rechnung brauchen keine Speicher belegt zu werden.

	Speicher	
	Nummer	Inhalt
Arbeitsspeicher:	0	a
	1	b
	2	c
	3	d
	4	N
	5	R
	6	I

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Poln. Notation		
Eingabe	Code	Eingabe	Code	Bemerkung
R/S	13–000	R/S	13–000	HALT für Eingabe von *a* (Realteil der ersten Zahl)
STO 0	64–001	STO 0	64–001	
	91–002		91–002	Speichern von *a*
R/S	13–003	R/S	13–003	HALT für Eingabe von *b* (Imaginärteil der ersten Zahl)
STO 1	65–004	STO 1	64–004	
	81–005		81–005	Speichern von *b*
R/S	13–006	R/S	13–006	HALT für Eingabe von *c* (Realteil der zweiten Zahl)
STO 2	64–007	STO 2	64–007	
	82–008		82–008	Speichern von *c*
R/S	13–009	R/S	13–009	HALT für Eingabe von *d* (Imaginärteil der zweiten Zahl)
STO 3	64–010	STO 3	64–010	
	83–011		83–011	Speichern von *d*
R/S	13–012	R/S	13–012	HALT für Eingabe von *K*
–	85–013	ENTER	95–013	*K*
4	71–014	4	71–014	4
=	95–015	–	85–015	$K - 4$

X ≧ 0	15–016	X ≧ 0	15–016	Entscheidung, ob $K \geqq 4$
GOTO 124	14–017	GOTO 109	14–017	Sprung zur Berechnung des Quotienten, wenn $K = 4$. Ist $K < 4$, so wird das Programm mit dem Befehl 021 fortgesetzt.
	81–018		81–018	
	82–019		91–019	
	71–020		63–020	
+	84–021			$K - 4$
1	81–022	1	81–021	1
=	95–023	+	84–022	$K - 3$
X ≧ 0	15–024	X ≧ 0	15–023	Entscheidung, ob $K \geqq 3$
GOTO 083	14–025	GOTO 079	14–024	Sprung zur Berechnung des Produkts, wenn $K = 3$ ist. Ist $K < 3$, so wird das Programm mit dem Befehl 029 bzw. 028 fortgesetzt.
	91–026		91–025	
	62–027		61–026	
	83–028		63–027	
+	84–029			$K - 3$
1	81–030	1	81–028	1
=	95–031	+	84–029	$K - 2$
X ≧ 0	15–032	X ≧ 0	15–030	Entscheidung, ob $K \geqq 2$
GOTO 063	14–033	GOTO 060	14–031	Sprung zur Berechnung der Differenz, wenn $K = 2$ ist. Ist $K = 1$, so wird das Programm mit dem Befehl 037 bzw. 035 fortgesetzt.
	91–034		91–032	
	73–035		72–033	
	83–036		91–034	
RCL 0	65–037	RCL 0	65–035	Beginn der Berechnung der Summe a
	91–038		91–036	

+	84–039			
RCL 2	65–040	RCL 2	65–037	
	82–041		82–038	c
=	95–042	+	84–039	$a + c = R$
STO 5	64–043	STO 5	64–040	
	72–044		72–041	Speichern von R
RCL 1	65–045	RCL 1	65–042	
	81–046		81–043	b
+	84–047			
RCL 3	65–048	RCL 3	65–044	
	83–049		83–045	d
=	95–050	+	84–046	$b + d = I$
STO 6	64–051	STO 6	64–047	
	73–052		73–048	Speichern von I
RCL 5	65–053	RCL 5	65–049	Ergebnisausgabe
	72–054		72–050	R
R/S	13–055	R/S	13–051	HALT für Ausgabe R
RCL 6	65–056	RCL 6	65–052	
	73–057		73–053	I
R/S	13–058	R/S	13–054	HALT für Ausgabe I
GOTO 000	14–059	GOTO 000	14–055	
	91–060		91–056	
	91–061		91–057	
	91–062		91–058	Rücksprung zum Programmanfang
		SST	99–059	Befehl ohne Wirkung
RCL 0	65–063	RCL 0	65–060	Beginn der Berechnung der Differenz
	91–064		91–061	a

−	85–065			
RCL 2	65–066	RCL 2	65–062	
	82–067		82–063	c
=	95–068	−	85–064	$R = a - c$
STO 5	64–069	STO 5	64–065	
	72–070		72–066	Speichern von R
RCL 1	65–071	RCL 1	65–067	
	81–072		81–068	b
−	85–073	SST	12–069	
RCL 3	65–074	RCL 3	65–070	
	83–075		83–071	d
=	95–076	−	85–072	$I = b - d$
STO 6	64–077	STO 6	64–073	
	73–078		73–074	Speichern von I
GOTO 053	14–079	GOTO 049	14–075	Sprung zur Ausgabe von R und I (Ergebnisausgabe)
	91–080		91–076	
	72–081		71–077	
	83–082		63–078	
RCL 0	65–083	RCL 0	65–079	Beginn der Berechnung des Produkts
	91–084		91–080	a
×	74–085			
RCL 2	65–086	RCL 2	65–081	
	82–087		82–082	c
−	85–088	×	74–083	ac
F	24–089			
(	61–090			
RCL 1	65–091	RCL 1	65–084	
	81–092		81–085	b

×	74–093			
RCL 3	65–094	RCL 3	65–086	
	83–095		83–087	d
F	24–096			
)	62–097	×	74–088	bd
=	95–098	−	85–089	$ac - bd = R$
STO 5	64–099	STO 5	64–090	
	72–100		72–091	Speichern von R
RCL 0	65–101	RCL 0	65–092	
	91–102		91–093	a
×	74–103			
RCL 3	65–104	RCL 3	65–094	
	83–105		83–095	d
+	84–106	×	74–096	ad
F	24–107			
(	61–108			
RCL 1	65–109	RCL 1	65–097	
	81–110		81–098	b
×	74–111			
RCL 2	65–112	RCL 2	65–099	
	82–113		82–100	c
SST	99–114			
F	24–115			
)	62–116	×	74–101	bc
=	95–117	+	84–102	$ad + bc = I$
STO 6	64–118	STO 6	64–103	
	73–119		73–104	Speichern von I
GOTO 053	14–120	GOTO 049	14–105	Sprung zur Ergebnisausgabe
	91–121		91–106	
	72–122		71–107	
	83–123		63–108	

RCL 2	65–124	RCL 2	65–109	Beginn der Berechnung des Quotienten
	82–125		82–110	c
F	24–126	F	24–111	
X^2	34–127	X^2	61–112	c^2
+	84–128			
RCL 3	65–129	RCL 3	65–113	
	83–130		83–114	d
F	24–131	F	24–115	
X^2	34–132	X^2	61–116	d^2
=	95–133	+	84–117	$N = c^2 + d^2$
STO 4	64–134	STO 4	64–118	
	71–135		71–119	Speichern von N
RCL 0	65–136	RCL 0	65–120	
	91–137		91–121	a
×	74–138			
RCL 2	65–139	RCL 2	65–122	
	82–140		82–123	c
+	84–141	×	74–124	ac
F	24–142			
(	61–143			
RCL 1	65–144	RCL 1	65–125	
	81–145		81–126	b
×	74–146			
RCL 3	65–147	RCL 3	65–127	
	83–148		83–128	d
F	24–149			
)	62–150	×	74–129	bd

÷	75–151	+	84–130	$ac + bd$
RCL 4	65–152	RCL 4	65–131	
	71–153		71–132	N
=	95–154	÷	75–133	$R = (ac + bd)/N$
STO 5	64–155	STO 5	64–134	
	72–156		72–135	Speichern von R
RCL 1	65–157	RCL 1	65–136	
	81–158		81–137	b
×	74–159			
RCL 2	65–160	RCL 2	65–138	c
	82–161		82–139	
−	85–162	×	74–140	bc
F	24–163			
(	61–164			
RCL 0	65–165	RCL 0	65–141	
	91–166		91–142	a
×	74–167			
RCL 3	65–168	RCL 3	65–143	
	83–169		83–144	d
F	24–170			
)	62–171	×	74–145	ad
÷	75–172	−	85–146	$bc-ad$
RCL 4	65–173	RCL 4	65–147	
	71–174		71–148	N
=	95–175	÷	75–149	$I = (bc - ad)/N$
STO 6	64–176	STO 6	64–150	
	73–177		73–151	Speichern von N

GOTO 053	14–178	GOTO 049	14–152	Sprung zur Ergebnisausgabe
	91–179		91–153	
	72–180		71–154	
	83–181		63–155	

Programmtest:
Das Programm wurde mit den beiden Zahlen

$$z_1 = 9 - 7\,\mathrm{i}$$
$$z_2 = 3 + 2\,\mathrm{i}$$

getestet.
Folgende Ergebnisse traten auf:

$$z_1 + z_2 = 12 - 5\,\mathrm{i}$$
$$z_1 - z_2 = 6 - 9\,\mathrm{i}$$
$$z_1 \cdot z_2 = 41 - 3\,\mathrm{i}$$
$$z_1 : z_2 = 1 - 3\,\mathrm{i}.$$

Alle Testergebnisse sind richtig.

Programmdokumentation:
1. Einstellen der geforderten Rechengenauigkeit
2. Start des Programms. Startadresse: 000
3. Eingabe des Realteils des ersten Operanden.
   R/S-Taste drücken.
4. Eingabe des Imaginärteils des ersten Operanden.
   R/S-Taste drücken.
5. Eingabe des Realteils des zweiten Operanden.
   R/S-Taste drücken.
6. Eingabe des Imaginärteils des zweiten Operanden.
   R/S-Taste drücken.
7. Eingabe der Kennziffer für die gewünschte Rechenoperation:
   $K = 1$ für Addition, $K = 2$ für Subtraktion
   $K = 3$ für Multiplikation, $K = 4$ für Division.
   R/S-Taste drücken.

8. Beim ersten Halt wird der Realteil des Ergebnisses ausgegeben.
   R/S-Taste drücken.
9. Beim zweiten Halt wird der Imaginärteil des Ergebnisses angezeigt.
   Damit ist die Rechnung beendet.

3.4. Zyklische Programme

3.4.1. Berechnung einer Summe mit *n* Summanden

Viele Aufgaben sind dadurch gekennzeichnet, daß im Verlauf ihrer Lösung ein und derselbe Algorithmus ständig mit neuen Eingangsgrößen wiederholt werden muß. Nehmen wir an, daß ein solcher Algorithmus bei einer Aufgabe fünfmal abgearbeitet werden muß, so wäre es bei einem linearen Programm erforderlich, diesen Algorithmus fünfmal nacheinander, jeweils auf die entsprechenden Eingangsgrößen zugeschnitten, im Programm aufzuschreiben. Das erfordert eine große Menge von Speicherplätzen im Programmspeicher und ist auch von der Programmierung her sehr uneffektiv. Das Verfahren wird sofort unakzeptabel, wenn der Algorithmus nicht fünfmal, sondern mehrere hundert Male nacheinander durchgerechnet werden muß.
Die im letzten Abschnitt erläuterte Möglichkeit, Programme zu verzweigen, gibt uns ein Mittel in die Hand, das Problem der *ständigen Wiederholung gleichartiger Rechengänge mit jeweils nur neuen Zahlen* sehr elegant zu programmieren. Wir wollen die Vorgehensweise an einem sehr einfachen Beispiel erläutern.
Es sei die Aufgabe gestellt, ein Programm zu schreiben, mit dessen Hilfe die Summe der ersten n Quadratzahlen gebildet werden kann.
Ein Weg, diese Aufgabe zu lösen, bestünde darin, in einer Formelsammlung nach einer Formel für die Summe der ersten n Quadratzahlen zu suchen und für diese Formel ein lineares Programm aufzustellen. (Dies wäre eine sehr lohnenswerte Übungsaufgabe für den Leser.) Dieses Verfahren scheitert aber sofort, wenn nicht die Summe von Quadrat-

zahlen, sondern beispielsweise die Summe der Kubikwurzeln der ersten n natürlichen Zahlen berechnet werden müßte. Daher soll ein Weg beschritten werden, der in jedem ähnlich gelagerten Beispiel zum Ziel führt. Wir lassen ganz einfach den Rechner *all die Schritte* nachvollziehen, *die der Mensch auch tun müßte,* um die gestellte Aufgabe lösen zu können, wenn er keine geeignete Formel findet: Dazu teilen wir dem Rechner zunächst einmal mit, wie viele Summanden er addieren soll. Dann lassen wir den Rechner der Reihe nach alle Quadratzahlen bilden und sie eine nach der anderen aufsummieren. Dies geschieht dadurch, daß wir eine *Hilfsgröße* i wählen, ihr Quadrat bilden und dieses Quadrat zu einer Größe s hinzufügen, die jeweils die Summe aller bisher gebildeten Quadrate darstellt. Damit diese Hilfsgröße i der Reihe nach alle Werte von 1 bis n annimmt, müssen wir am Anfang festlegen, daß $i = 1$ sein soll, und nach der Summation von i^2 zur Größe s müssen wir jeweils den bisherigen Wert von i um 1 erhöhen, was im *Programmablaufplan* (vgl. Bild 11) durch die Anweisung $i := i + 1$ versinnbildlicht wird.

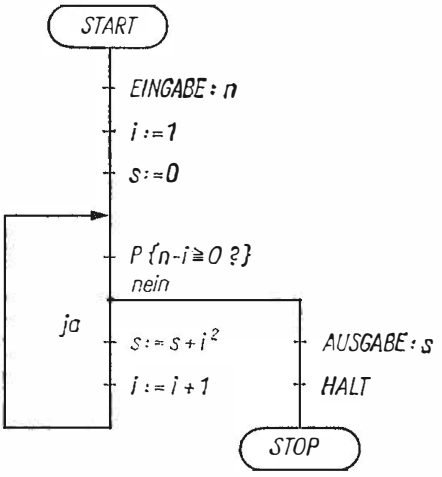


Bild 11

Dieses Erhöhen von i um den Wert 1 und das Summieren des Quadrates von i zu s muß so lange durchgeführt werden, bis i den Wert n erreicht hat. Daher muß in den Programmablauf eine Möglichkeit eingebaut werden, festzustellen, wann dies

der Fall ist. Dazu führen wir die Abfrage ein, ob $i \geqq n$ geworden ist. Am Anfang wird dies noch nicht der Fall sein. Aus diesem Grund muß die Addition der nächsten Quadratzahlen fortgesetzt werden. Sobald aber i den Wert n erreicht hat, kann die Summation abgebrochen und das Ergebnis der Summation ausgegeben werden.

Damit erhalten wir den in Bild 11 dargestellten *Programmablaufplan*, der mit geringfügigen Veränderungen stets verwendet werden kann, wenn viele Glieder einer beliebigen Folge summiert werden sollen.

Test des Programmablaufplans:

Der vorliegende Programmablaufplan soll für den Fall überprüft werden, daß die ersten vier Quadratzahlen summiert werden sollen. In diesem Fall ist zu Beginn der Rechnung die Zahl 4 einzugeben.

Beim Testen von Programmablaufplänen ist besonderes Augenmerk darauf zu richten, daß die Programmverzweigungen an den Verzweigungsstellen auch so ausgeführt werden, wie dies in der Aufgabenstellung gefordert ist. Für unser Beispiel bedeutet dies, daß die Summierung der Quadratzahlen genau nach 4 Gliedern abgebrochen werden muß und nicht etwa eher oder später.

Programmschritt	Ablauf				
Eingabe: n	4				
$i := 1$	1				
$s := 0$	0				
$n - i$	3	→2	→1	→0	→−1
$n - i \geqq 0$?	ja	ja	ja	ja	nein
$s := s + i^2$	0+1=1	1+4=5	5+9=14	14+16=30	
$i := i + 1$	2	3	4	5	
Ausgabe: s					30

Die Kontrollrechnung ergibt

$$s = 1^2 + 2^2 + 3^2 + 4^2 = 30,$$

sie stimmt mit der bei der Testrechnung erhaltenen Lösung überein. Die erforderlichen Programmverzweigungen werden ordnungsgemäß durchgeführt. Der aufgestellte Programmablaufplan erweist sich als richtig.

Speicherbelegungsplan:

Vor Beginn der Rechnung brauchen keine Speicherplätze belegt zu werden.

	Speicher	
	Nummer	Inhalt
Arbeitsspeicher:	0	n
	1	i
	2	s

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Poln. Notation		Bemerkung
Eingabe	Code	Eingabe	Code	
R/S	13–000	R/S	13–000	HALT für die Eingabe von n
STO 0	64–001	STO 0	64–001	
	91–002		91–002	n
1	81–003	1	81–003	1
STO 1	64–004	STO 1	64–004	
	81–005		81–005	Speichern des Anfangswertes von i
0	91–006	0	91–006	
STO 2	64–007	STO 2	64–007	
	82–008		82–008	Speichern des Anfangswertes von s
RCL 0	65–009	RCL 0	65–009	
	91–010		91–010	n
–	85–011			
RCL 1	65–012	RCL 1	65–011	
	81–013		81–012	i
=	95–014	–	85–013	$n - i$
X ≧ 0	15–015	X ≧ 0	15–014	Entscheidung, ob $n \geqq i$
GOTO 027	14–016	GOTO 026	14–015	Sprung zur Berechnung von $s + i^2$
	91–017		91–016	usw., wenn $n \geqq i$.
	82–018		82–017	Ist $n < i$, so wird
	81–019		73–018	diese Summation abgebrochen und mit dem Befehl 020 bzw. 019 die Ergebnisausgabe vorbereitet.

RCL 2	65–020	RCL 2	65–019	
	82–021		82–020	s
R/S	13–022	R/S	13–021	HALT für die Ausgabe von s
GOTO 000	14–023	GOTO 000	14–022	
	91–024		91–023	
	91–025		91–024	
	91–026		91–025	Rücksprung zum Programmanfang
RCL 1	65–027	RCL 1	65–026	
	81–028		81–027	i
×	74–029			
RCL 1	65–030	ENTER	95–028	
	81–031			i
+	84–032	×	74–029	i^2
RCL 2	65–033	RCL 2	65–030	
	82–034		82–031	s
=	95–035	+	84–032	$s + i^2$
STO 2	64–036	STO 2	64–033	
	82–037		82–034	Speichern des neuen s
RCL 1	65–038	RCL 1	65–035	
	81–039		81–036	i
+	84–040			
1	81–041	1	81–037	1
=	95–042	+	84–038	$i + 1$
STO 1	64–043	STO 1	64–039	
	81–044		81–040	Speichern des neuen i

GOTO 009	14–045	GOTO 009	14–041	Rücksprung zu der Stelle des Programms, an der die Testgröße $n - i$ berechnet wird. (Befehl 009)
	91–046		91–042	
	91–047		91–043	
	63–048		63–044	

Programmtest:

Das Programm wurde mit dem Wert $n = 10$ getestet. Es ergibt sich der richtige Wert der Summe der ersten 10 Quadratzahlen:

$$s = 385,$$

was durch Anwendung der Formel $s = \sum_{i=1}^{n} i^2 = \frac{n}{6} \cdot (n+1) \cdot (2n+1) = 385$ bestätigt werden kann.

Programmdokumentation:

1. Start des Programms. Startadresse: 000
2. Eingabe der Anzahl n der Glieder der Reihe.
   R/S -Taste drücken.
3. Die Rechnung wird beendet mit der Ausgabe der Summe der ersten n Quadratzahlen.

3.4.2. Berechnung einer Summe mit beliebig vielen Summanden

Im letzten Beispiel war die Anzahl der Summanden der zu berechnenden Summe als bekannt vorausgesetzt worden. Man brauchte in diesem Fall nur eine Zählgröße i in das Programm aufzunehmen, die bei der Abarbeitung des nächsten Summanden jeweils um 1 zu erhöhen war, und die das Signal zum Abbruch der Summation gab, sobald $i = n$ erreicht war.
Nun gibt es aber vor allem im Zusammenhang mit der *Auswertung von Meßreihen* sehr viele Beispiele, in denen nicht von vornherein bekannt ist, wie viele Einzelsummen aufsummiert werden müssen, sondern wo diese einzelnen Glieder aus einer evtl. sehr langen Meßreihe entstammen, bei der es

sich nicht lohnt, abzuzählen, wie viele Glieder vorhanden sind.
In einem solchen Fall muß man den Rechner zum einen befähigen, mitzuzählen, wie viele Meßwerte er bereits erfaßt hat, und zum anderen muß man ihn auch befähigen, zu erkennen, wann die Meßreihe zu Ende ist. Am folgenden Beispiel soll gezeigt werden, wie man diese Aufgabe lösen kann.
Es soll der *Mittelwert der Quadrate der Meßwerte einer beliebig langen Meßreihe* ermittelt werden, wobei die einzelnen Meßwerte der Reihe nach in den Rechner eingegeben werden sollen.
Damit der Rechner selbst zählen kann, wie viele Meßwerte eingegeben worden sind (die Anzahl der Meßwerte benötigt er ja zur Bildung des Mittelwertes), führen wir wieder wie im letzten Beispiel eine *Zählgröße i* ein.
Damit der Rechner erkennen kann, wann das *Ende der Meßreihe* erreicht ist, speichern wir zu Beginn der Rechnung ein sogenanntes *Schlußkennzeichen skz* und wählen hierfür eine Zahl, die sicher kleiner ist als der kleinste auftretende Meßwert. (Eine solche Zahl läßt sich bei Meßreihen aus physikalischen oder technischen Versuchen stets bestimmen.) Lassen wir nun den Rechner jeden Meßwert mit diesem Schlußkennzeichen *skz* vergleichen, so wird er feststellen, daß jeder Meßwert größer als *skz* ist. Hängt man schließlich an das Ende der Meßreihe noch einen fiktiven Meßwert an, der noch kleiner gewählt wird als das Schlußkennzeichen, so wird der Rechner beim Vergleich dieses fiktiven Meßwertes mit *skz* feststellen, daß dieser im Gegensatz zu den vorangegangenen Fällen kleiner ist als *skz*. Dies ist für ihn als Zeichen zu betrachten, daß die Meßreihe abgeschlossen ist. Er kann damit zur Endauswertung, in unserem Beispiel zur Berechnung des Mittelwertes der Quadrate aller Meßwerte, übergehen.
Ein entsprechender *Programmablaufplan* ist in Bild 12 dargestellt.

Test des Programmablaufplans:

Der Programmablaufplan soll am Beispiel der Berechnung des Mittelwerts der Quadrate der Zahlen

$$x_1 = 5, \quad x_2 = 7, \quad x_3 = 3 \quad \text{und} \quad x_4 = 6$$

überprüft werden.

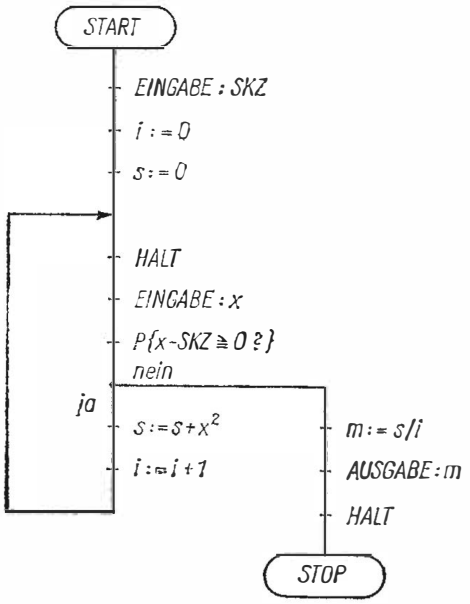


Bild 12

Als Schlußkennzeichen werde die Zahl

$$skz = 1$$

gewählt, so daß im Verlauf der Abarbeitung des Programms entsprechend der Programmdokumentation der Reihe nach die Zahlen

5, 7, 3, 6, − 5

eingegeben werden müssen, wobei die ersten dieser vier Zahlen die Werte der Meßreihe x_1 bis x_4 darstellen und die letzte Zahl dazu dient, daß der Rechner den Abschluß der Meßreihe erkennt. Diese Zahl soll nach der vorliegenden Programmierung kleiner gewählt werden als das Schlußkennzeichen.

Programmschritt	Ablauf				
Eingabe:					
skz	1				
$i := 0$	0				
$s := 0$	0				
Eingabe: x	5	→7	→3	→6	→−5
$x - skz$	4	6	2	5	−6
$x - skz \geqq 0$?	ja	ja	ja	ja	nein
$s := s + x^2$	0+25=25	25+49=74	74+9=83	83+36=119	↓
$i := i + 1$	1	2	3	4	
$m := s/i$					119/4 = 29.75
Ausgabe: m					29.75

Der aufgestellte Programmablaufplan erweist sich als richtig.

Speicherbelegungsplan:

Vor Beginn der Rechnung brauchen keine Speicherplätze belegt zu werden.

	Speicher	
	Nummer	Inhalt
Arbeitsspeicher:	0	*skz*
	1	*i*
	2	*s*
	3	Meßwert

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Poln. Notation		
Eingabe	Code	Eingabe	Code	Bemerkung
R/S	13–000	R/S	13–000	HALT für Eingabe des Schlußkennzeichens *skz*
STO 0	64–001 91–002	STO 0	64–001 91–002	Speichern von *skz*
0	91–003	0	91–003	Eingabe 0
STO 1	64–004 81–005	STO 1	64–004 81–005	Speichern des Anfangswertes 0 für die Zählgröße *i*
STO 2	64–006 82–007	STO 2	64–006 82–007	Speichern des Anfangswertes 0 für die Summenbildung *s*
R/S	13–008	R/S	13–008	HALT für die Eingabe eines Meßwertes *x*
STO 3	64–009 83–010	STO 3	64–009 83–010	Speichern des Meßwertes *x*
–	85–011			
RCL 0	65–012 91–013	RCL 0	65–011 91–012	*skz*
=	95–014	–	85–013	$x - skz$
X ≧ 0	15–015	X ≧ 0	15–014	Entscheidung, ob $x \geqq skz$, d.h., ob *x* ein Meßwert ist oder ob *x* das Ende der Meßreihe ankündigt

GOTO 031	14–016	GOTO 029	14–015	
	91–017		91–016	Sprung zur Verarbeitung des Meßwertes, wenn $x \geqq skz$. Ist $x < skz$, so wird das Programm mit Befehl 20 bzw. 19 fortgesetzt, wo die Endauswertung beginnt.
	83–018		82–017	
	81–019		63–018	
RCL 2	65–020	RCL 2	65–019	
	82–021		82–020	s
÷	75–022			
RCL 1	65–023	RCL 1	65–021	
	81–024		81–022	i
=	95–025	÷	75–023	$m = s/i$
R/S	13–026	R/S	13–024	HALT für die Ausgabe von m
GOTO 000	14–027	GOTO 000	14–025	
	91–028		91–026	
	91–029		91–027	
	91–030		91–028	Rücksprung zum Programmanfang
RCL 3	65–031	RCL 3	65–029	Beginn der Verarbeitung eines Meßwertes x
	83–032		83–030	x
×	74–033	ENTER	95–031	
RCL 3	65–034			
	83–035			x
+	84–036	×	74–032	x^2
RCL 2	65–037	RCL 2	65–033	
	82–038		82–034	s

=	95–039	+	84–035	$s + x^2$
STO 2	64–040	STO 2	64–036	
	82–041		82–037	Speichern des neuen s
RCL 1	65–042	RCL 1	65–038	
	81–043		81–039	i
+	84–044			
1	81–045	1	81–040	1
=	95–046	+	84–041	$i + 1$
STO 1	64–047	STO 1	64–042	
	81–048		81–043	Speichern des neuen i
GOTO 008	14–049	GOTO 008	14–044	
	91–050		91–045	
	91–051		91–046	
	62–052		62–047	Rücksprung zum Befehl 008, bei dem die Verarbeitung des nächsten Meßwertes beginnt

Programmtest:

Das Programm wurde mit der bei der Überprüfung des Programmablaufplans verwendeten Zahlenfolge getestet und brachte das dort angegebene Ergebnis.

Programmdokumentation:

1. Geforderte Rechengenauigkeit einstellen.
2. Start des Programms. Startadresse: 000.
3. Eingabe des Schlußkennzeichens *skz*. Es muß kleiner als der kleinste Meßwert sein.
   R/S -Taste drücken.
4. Bei jedem folgenden Halt ist der nächste Meßwert einzugeben. Danach R/S -Taste drücken.

5. Nach Abschluß der Meßreihe ist eine zusätzliche Zahl einzugeben, die kleiner als das Schlußkennzeichen *skz* gewählt werden muß.

   Danach [R/S]-Taste drücken.

6. Beim letzten Halt erscheint in der Zahlenanzeige der gesuchte Wert

$$s = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2 .$$

3.4.3. Mittelwert und Streuung einer Meßreihe

Es sollen *Mittelwert und Streuung einer Meßreihe* berechnet werden, die aus den Meßwerten

$$x_1, x_2, x_3, \ldots$$

bestcht. Dabei soll wiederum offengelassen werden, wie viele Meßwerte vorhanden sind.
Im Unterschied zur letzten Aufgabe soll als zusätzlicher fiktiver Meßwert am Ende der Meßreihe diesmal das unter denselben Voraussetzungen wie in 3.4.2. gewählte *Schlußkennzeichen skz* eingegeben werden. Daher muß der Rechner für jeden Meßwert x nachprüfen, ob $x = skz$ ist. Ist dies der Fall, so ist die Meßreihe zu Ende, und es kann mit der Berechnung von Mittelwert und Streuung begonnen werden.
Die Berechnung des Mittelwertes erfolgt nach der Formel

$$\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i .$$

Zur Berechnung der Streuung wird meist die Formel

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2}$$

verwendet.
Ihre Anwendung würde aber voraussetzen, daß man alle eingegebenen Meßwerte speichern müßte, bis der Mittelwert $\bar{x}$ berechnet ist, damit daran anschließend die Differenzen $x_i - \bar{x}$ gebildet werden können. Dazu reicht jedoch die Speicherkapazität eines Taschenrechners nicht aus.

Nun ist aber

$$\sum_{i=1}^{n} (x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i^2 - 2 \cdot \sum_{i=1}^{n} x_i \cdot \bar{x} + \sum_{i=1}^{n} \bar{x}^2.$$

Da $\bar{x}$ eine konstante Größe ist, kann man dafür auch schreiben (vgl. [2])

$$\sum_{i=1}^{n} (x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i^2 - 2 \cdot \bar{x} \cdot \sum_{i=1}^{n} x_i + n \cdot \bar{x}^2.$$

Beachtet man noch die Definition für den Mittelwert $\bar{x}$, so folgt daraus für die Streuung

$$\sigma = \sqrt{\frac{1}{n-1} \cdot \left(\sum_{i=1}^{n} x_i^2 - n \cdot \bar{x}^2 \right)}.$$

Diese Formel ist für die Berechnung der Streuung mit Hilfe eines Taschenrechners besser geeignet als die zuerst angegebene, denn man kann nun sofort neben der Summierung aller Meßwerte x_i, die zur Mittelwertbestimmung benötigt wird, auch gleichzeitig die Summierung aller Quadrate der Meßwerte vornehmen, so daß die Zwischenspeicherung aller Meßwerte nicht mehr erforderlich ist.
Die sonstigen Überlegungen zur Aufstellung des *Programmablaufplans* sind die gleichen wie sie in 3.4.2. angestellt wurden. Den Programmablaufplan zeigt Bild 13.

Test des Programmablaufplans:

Für die Überprüfung des Programmablaufplans soll die leicht zu überblickende Meßreihe mit den Meßwerten

$$x_1 = 1,\ x_2 = 2,\ x_3 = 3,\ x_4 = 4$$

gewählt werden.
Der Mittelwert dieser Meßreihe ist

$$\bar{x} = 2.5,$$

und die Streuung beträgt

$$\sigma = 1.29.$$

Entsprechend der Programmdokumentation müssen der Reihe nach die folgenden Zahlen eingegeben werden, wenn als Schlußkennzeichen die Zahl

$$skz = 0$$

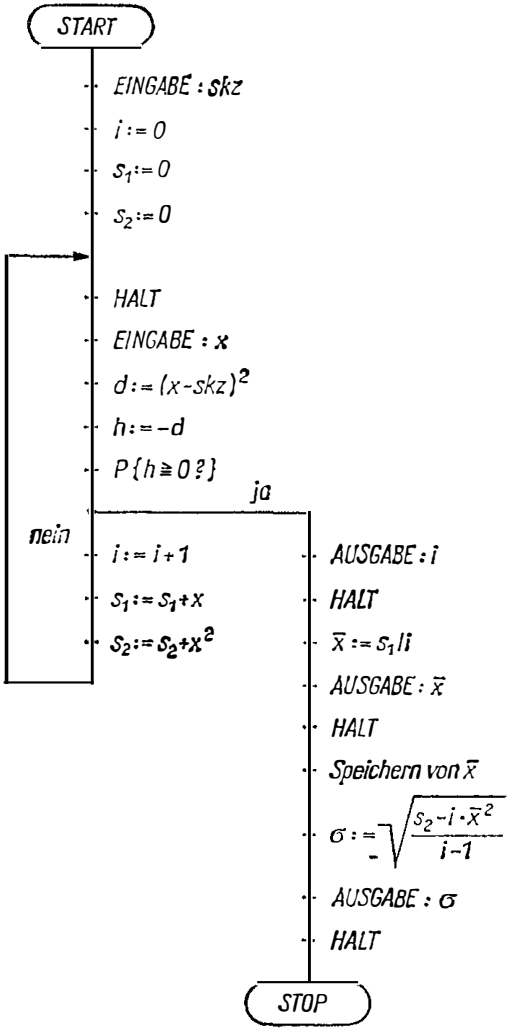


Bild 13

festgelegt wird:

0	(Schlußkennzeichen)
1	(erster Meßwert)
2	(zweiter Meßwert)
3	(dritter Meßwert)
4	(vierter Meßwert)
0	(Schlußkennzeichen als Hinweis dafür, daß die Meßreihe abgeschlossen ist).

Programmschritt	Ablauf				
Eingabe: skz	0				
$i := 0$	0				
$s_1 := 0$	0				
$s_2 := 0$	0				
Eingabe: x	1	→2	→3	→4	→0
$d := (x - skz)^2$	1	4	9	16	0
$h := -d$	−1	−4	−9	−16	0
$h \geqq 0$?	nein	nein	nein	nein	ja
$i := i + 1$	1	2	3	4	
$s_1 := s_1 + x$	0+1=1	1+2=3	3+3=6	6+4=10	
$s_2 := s_2 + x^2$	0+1=1	1+4=5	5+9=14	14+16=30	
		↓			
Ausgabe: i		4			
$\bar{x} := s_1/i$		10/4=2.5			
Ausgabe: $\bar{x}$		2.5			
$\sigma := \sqrt{\frac{s_2 - i \cdot \bar{x}^2}{i - 1}}$		1.29			
Ausgabe: σ		1.29			

Der Programmablaufplan erweist sich als richtig.

Speicherbelegungsplan:

Vor Beginn der Rechnung brauchen keine Speicherplätze belegt zu werden.

	Speicher	
	Nummer	Inhalt
Arbeitsspeicher:	0	skz
	1	i
	2	s_1
	3	s_2
	4	$\bar{x}$

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Poln. Notation		
Eingabe	Code	Eingabe	Code	Bemerkung
R/S	13–000	R/S	13–000	HALT für Eingabe des Schlußkennzeichens *skz*
STO 0	64–001	STO 0	64–001	
	91–002		91–002	*skz* wird gespeichert.
0	91–003	0	91–003	0
STO 1	64–004	STO 1	64–004	
	81–005		81–005	Speichern des Anfangswertes von i
STO 2	64–006	STO 2	64–006	
	82–007		82–007	Speichern des Anfangswertes von s_1
STO 3	64–008	STO 3	64–008	
	83–009		83–009	Speichern des Anfangswertes von s_2
R/S	13–010	R/S	13–010	HALT für Eingabe eines Meßwertes. Beginn der Auswertung der Meßreihe
STO 5	64–011	STO 5	64–011	
	72–012		72–012	Speichern des Meßwertes
–	85–013			
RCL 0	65–014	RCL 0	65–013	
	91–015		91–014	*skz*
=	95–016	–	85–015	$x - skz$
STO 6	64–017			
	73–018			
×	74–019			

RCL 6	65–020			
	73–021	ENTER	95–016	$x - skz$
=	95–022	×	74–017	$d = (x - skz)^2$
−	85–023			
0	91–024	0	91–018	0
X ↔ Y	54–025	X ↔ Y	54–019	
=	95–026	−	85–020	$h = 0 - d =$ $= -(x - skz)^2$
X ≧ 0	15–027	X ≧ 0	15–021	Entscheidung, ob das Schlußkennzeichen aufgetreten ist ($h = 0$) oder nicht ($h < 0$)
GOTO 062	14–028	GOTO 052	14–022	
	91–029		91–023	Sprung zur Endauswertung, wenn $h =$ $= 0$. Sonst wird der Sprungbefehl übergangen und das Programm mit dem Befehl 032 bzw. 026 fortgesetzt.
	73–030		72–024	
	82–031		82–025	
RCL 1	65–032	RCL 1	65–026	
	81–033		81–027	i
+	84–034			
1	81–035	1	81–028	1
=	95–036	+	84–029	$i + 1$
STO 1	64–037	STO 1	64–030	
	81–038		81–031	Speichern des neuen i
RCL 2	65–039	RCL 2	65–032	
	82–040		82–033	s_1
+	84–041			

RCL 5	65–042	RCL 5	65–034	
	72–043		72–035	x
=	95–044	+	84–036	$s_1 + x$
STO 2	64–045	STO 2	64–037	
	82–046		82–038	Speichern des neuen s_1
RCL 5	65–047	RCL 5	65–039	
	72–048		72–040	x
×	74–049			
RCL 5	65–050			
	72–051	ENTER	95–041	
+	84–052	×	74–042	x^2
RCL 3	65–053	RCL 3	65–043	
	83–054		83–044	s_2
=	95–055	+	84–045	$s_2 + x^2$
STO 3	64–056	STO 3	64–046	
	83–057		83–047	Speichern von s_2
GOTO 010	14–058	GOTO 010	14–048	Sprung zur Eingabe des neuen Meßwertes (Befehl 010)
	91–059		91–049	
	81–060		81–050	
	91–061		91–051	
RCL 1	65–062	RCL 1	65–052	Beginn der Endauswertung
	81–063		81–053	i
R/S	13–064	R/S	13–054	HALT für die Ausgabe von i (Anzahl der Meßwerte)
÷	75–065			
RCL 2	65–066	RCL 2	65–055	
	82–067		82–056	s_1
X↔Y	54–068	X↔Y	54–057	
=	95–069	÷	75–058	$\bar{x} = s/i$

STO 4	64–070	STO 4	64–059	
	71–071		71–060	Speichern von $\bar{x}$
R/S	13–072	R/S	13–061	HALT für Ausgabe des Mittelwertes
×	74–073			
RCL 4	65–074			
	71–075	ENTER	95–062	$\bar{x}$
×	74–076	×	74–063	$\bar{x}^2$
RCL 1	65–077	RCL 1	65–064	
	81–078		81–065	i
−	85–079	×	74–066	$i \cdot \bar{x}^2$
RCL 3	65–080	RCL 3	65–067	
	83–081		83–068	s_2
X↔Y	54–082	X↔Y	54–069	
=	95–083	−	85–070	$s_2 - i \cdot \bar{x}^2$
STO 6	64–084	STO 6	64–071	
	73–085		73–072	Speichern von $s_2 - i \cdot \bar{x}^2$
RCL 1	65–086	RCL 1	65–073	
	81–087		81–074	i
−	85–088			
1	81–089	1	81–075	1
÷	75–090	−	85–076	$i - 1$
RCL 6	65–091			
	73–092			
X↔Y	54–093	X↔Y	54–077	
=	95–094		75–078	$(s_2 - i \cdot \bar{x}^2)/(i - 1) = \sigma^2$
		F	24–079	
√	34–095	√	62–080	σ
R/S	13–096	R/S	13–081	HALT für Ausgabe von σ

GOTO 000	14–097	GOTO 000	14–082	
	91–098		91–083	
	91–099		91–084	
	91–100		91–085	Rücksprung zum Programmanfang

Programmtest:

Das Programm wurde mit den Daten getestet, die bei der Überprüfung des Programmablaufplans verwendet wurden. Es ergaben sich die gleichen Ergebnisse.

Programmdokumentation:

1. Geforderte Rechengenauigkeit einstellen.
2. Programm starten. Startadresse: 000
3. Eingabe des Schlußkennzeichens *skz*, das kleiner sein muß als der kleinste Meßwert. Danach [R/S]-Taste drücken.
4. Bei jedem Halt ist ein neuer Meßwert einzugeben und danach die [R/S]-Taste zu drücken.
5. Nach dem letzten Meßwert ist noch einmal das Schlußkennzeichen *skz* einzugeben. [R/S]-Taste drücken.
6. Beim ersten Halt zeigt der Rechner die Anzahl der erfaßten Meßwerte an. [R/S]-Taste drücken.
7. Beim zweiten Halt zeigt der Rechner den Mittelwert der Meßreihe an. Danach erneut [R/S]-Taste drücken.
8. Beim letzten Halt zeigt der Rechner die Streuung der Meßreihe an.
   Damit ist die Rechnung abgeschlossen.

3.4.4. Lineare Regression

Häufig besteht zwischen zwei Größen ein *funktioneller Zusammenhang*, dessen exakte mathematische Formulierung jedoch Schwierigkeiten bereitet. So existieren beispielsweise zwischen der Körpergröße eines Menschen und dessen Gewicht, zwischen der Flughöhe eines Flugzeugs und dem dort

herrschenden Luftdruck, zwischen der Tageszeit und dem zu diesem Zeitpunkt auftretenden Energiebedarf eines Territoriums u.a.m. ganz sicherlich enge Beziehungen.
Will man nun einen derartigen Zusammenhang genauer untersuchen, so geht man folgendermaßen vor: Soll der Zusammenhang beispielsweise zwischen Körpergröße und Gewicht von Menschen näherungsweise ermittelt werden, so wählt man eine gewisse Anzahl von Testpersonen aus und erfaßt von diesen die Körpergröße x_i und das zugehörige Gewicht y_i $(i = 1\ (1)\ n)$. Die so erhaltenen Wertepaare (x_i, y_i) würden, in ein Koordinatensystem als diskrete Punkte eingetragen, eine Punktmenge ergeben, wie sie etwa in der oberen Skizze des Bildes 14 dargestellt ist. Stellt man nun an Hand einer derartigen Punktmenge fest, daß ein annähernd linearer Zusammenhang zwischen den Größen x_i und y_i besteht, so kann man die Koeffizienten der Gleichung einer Geraden

$$y = a + b\,x$$

berechnen, die sich optimal in die gegebene Punktmenge einfügt. Diese Gerade ist in der unteren Skizze des Bildes 14 eingetragen worden.
Man nennt die Funktion $y = a + b\,x$ die *Regressionsfunktion* und die zugehörige Gerade die *Regressionsgerade.*
Die Koeffizienten a und b der Regressionsfunktion

$$y = a + b\,x$$

ermittelt man am besten aus den Beziehungen (vgl. [2])

$$b = \frac{n \cdot \sum_{i=1}^{n} x_i \cdot y_i - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i}{n \cdot \sum_{i=1}^{n} (x_i)^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$a = \frac{1}{n} \cdot \left(\sum_{i=1}^{n} y_i - b \cdot \sum_{i=1}^{n} x_i\right).$$

Der zugehörige *Programmablaufplan* ist in Bild 15 dargestellt.

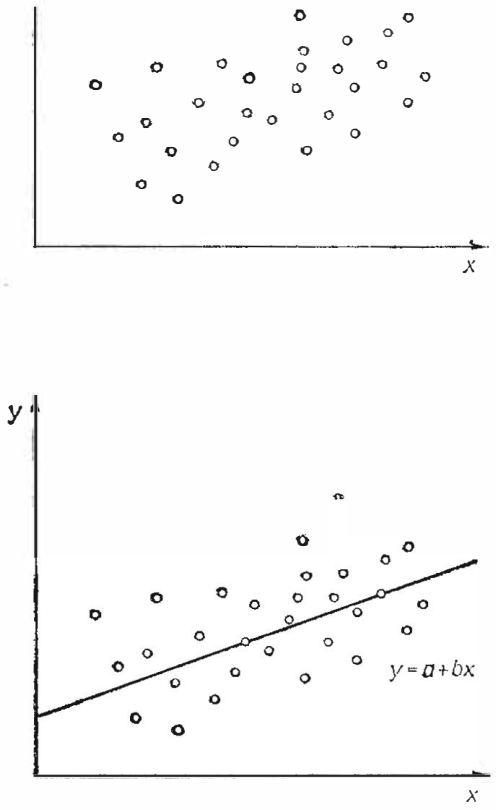


Bild 14

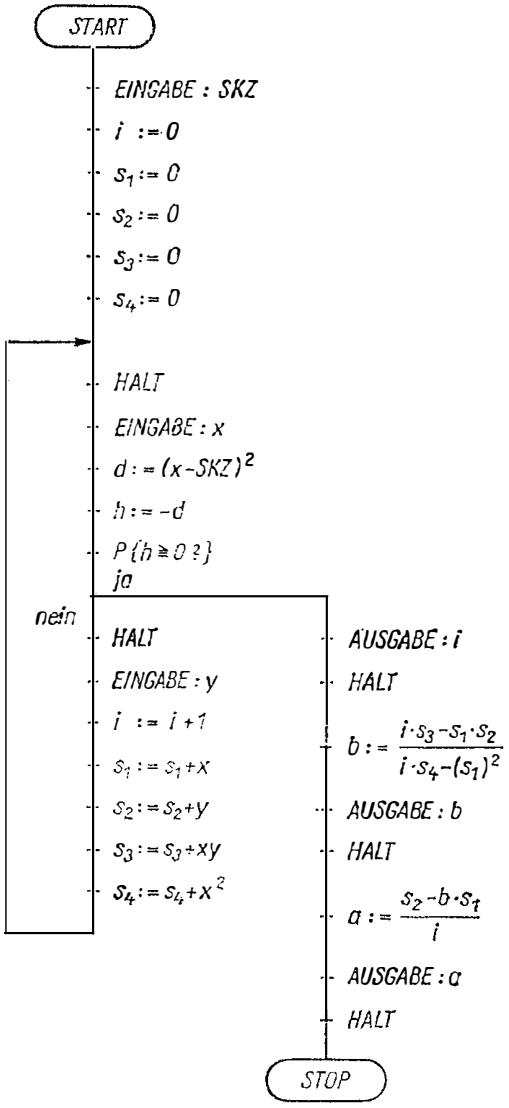

START
EINGABE : SKZ
i := 0
s1 := 0
s2 := 0
s3 := 0
s4 := 0
HALT
EINGABE : x
d := (x-SKZ)²
h := -d
P{h ≧ 0 ?}
ja
nein
HALT
EINGABE : y
i := i + 1
s1 := s1 + x
s2 := s2 + y
s3 := s3 + xy
s4 := s4 + x²
AUSGABE : i
HALT
b := (i·s3 - s1·s2) / (i·s4 - (s1)²)
AUSGABE : b
HALT
a := (s2 - b·s1) / i
AUSGABE : a
HALT
STOP

Bild 15

Test des Programmablaufplans:

Zur Überprüfung des Programmablaufplans werden die folgenden Wertepaare gewählt:

x	2	6	10	16
y	11	9	7	4

Wie sich leicht nachprüfen läßt, gehören alle vier Wertepaare der Funktion mit der Funktionsgleichung

$$y = 12 - 0{,}5 \cdot x$$

an.

Wenn also der Programmablaufplan richtig aufgestellt worden ist, dann müssen sich bei der Ausgabe der Ergebnisse folgende Zahlen ergeben:

$i = 4$	(Anzahl der Wertepaare)
$b = -0{,}5$	(Regressionskoeffizient)
$a = 12$	(Absolutglied in der Geradengleichung).

Als Schlußkennzeichen werde hier die Zahl

$$skz = 0$$

gewählt.

Mit diesen Festlegungen ergibt sich die folgende Reihenfolge der Zahleneingaben während des Ablaufs der Rechnung:

0	(Schlußkennzeichen)
2	(erster x-Wert)
11	(zugehöriger y-Wert)
6	(zweiter x-Wert)
9	(zugehöriger y-Wert)
10	(dritter x-Wert)
7	(zugehöriger y-Wert)
16	(vierter x-Wert)
4	(zugehöriger y-Wert)
0	(Schlußkennzeichen).

Programmschritt	Ablauf				
Eingabe: skz	0				
$i := 0$	0				
$s_1 := 0$	0				
$s_2 := 0$	0				
$s_3 := 0$	0				
$s_4 := 0$	0				
Eingabe: x	2	→ 6	→ 10	→ 16	→ 0
$d := (x - skz)^2$	4	36	100	256	0
$h := -d$	− 4	− 36	− 100	− 256	0
$h \geqq 0$?	nein	nein	nein	nein	ja
Eingabe: y	11	9	7	4	↓
$i := i + 1$	1	2	3	4	
$s_1 := s_1 + x$	0+2=2	2+6=8	8+10=18	18+16=34	
$s_2 := s_2 + y$	0+11=11	11+9=20	20+7=27	27+4=31	
$s_3 := s_3 + xy$	0+22=22	22+54=76	76+70=146	146+64=210	
$s_4 := s_4 + x^2$	0+4=4	4+36=40	40+100=140	140+256=396	
Ausgabe: i					4
$b := \frac{is_3 - s_1 s_2}{is_4 - s_1^2}$				$\frac{4 \cdot 210 - 34 \cdot 31}{4 \cdot 396 - 34^2} = -0.5$	
Ausgabe: b					-0.5
$a := \frac{s_2 - bs_1}{i}$				$\frac{31 - (-0{,}5) \cdot 34}{4} = 12$	
Ausgabe: a					12

Der Programmablaufplan erweist sich als richtig.

Speicherbelegungsplan:

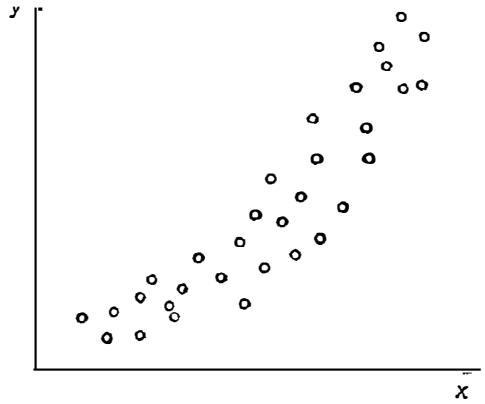
Vor der Berechnung brauchen keine Speicherplätze belegt zu werden.

	Speicher		
	Nummer	Inhalt	Bedeutung
Arbeits-speicher:	0	*skz*	Schlußkennzeichen
	1	i	Zählgröße
	2	s_1	Summe der x-Werte
	3	s_2	Summe der y-Werte
	4	s_3	Summe der Produkte xy
	5	s_4	Summe der Quadrate x^2
	6	x	x-Werte
	7	y	y-Werte

Da der vorliegende Programmablaufplan in seinen wesentlichen Einzelheiten fast genau mit dem des letzten Beispiels übereinstimmt, sei dem Leser empfohlen, das Rechenprogramm als Übung selbst aufzustellen. Das Programm ist am Ende des Buches zum Vergleich angegeben.
Für den Programmtest empfehlen wir, das Zahlenbeispiel zu wählen, das als Testbeispiel für die Überprüfung des Programmablaufplans gewählt wurde.
Der Leser stelle auch eine ausführliche Programmdokumentation entsprechend dem Beispiel aus den vorangegangenen Abschnitten zusammen.

3.4.5. Nichtlineare Regression

Nicht immer wird der erkennbare Zusammenhang zwischen zwei voneinander abhängigen Größen so schön linear erscheinen, wie dies in Bild 14 dargestellt wurde, sondern es wird auch Fälle geben, wo der mutmaßliche Verlauf der Regressionskurve deutlich von einer Geraden abweicht. In Bild 16 ist ein derartiges Beispiel dargestellt.
Hier könnte man vermuten, daß zwischen den beiden Variablen x und y ein *exponentieller Zusammenhang* besteht. Daher wird als Regressionsfunktion eine Funktion mit der



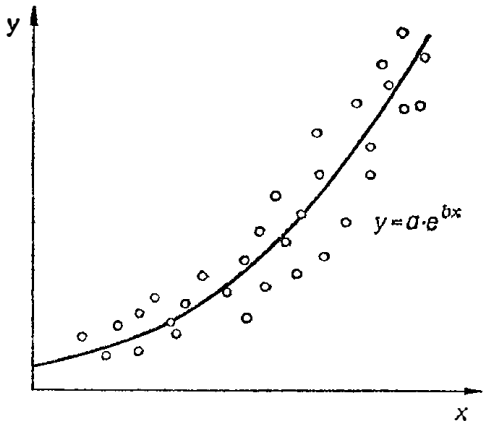


Bild 16

Gleichung

$$y = a \cdot e^{bx}$$

angesetzt (vgl. [5]).
(Es könnte natürlich auch eine Potenzfunktion der Form

$$y = a \cdot x^k + b$$

als Ansatz verwendet werden, jedoch würde in diesem Fall die Bestimmung der drei Parameter a, b und k wesentlich größere Schwierigkeiten bereiten als bei einem Ansatz mit Hilfe einer Exponentialfunktion.)

Die beiden Koeffizienten a und b für den oben erwähnten Exponentialansatz errechnen sich aus den Beziehungen

$$b = \frac{n \cdot \sum_{i=1}^{n} x_i \cdot \ln y_i - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} \ln y_i}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2}$$

und

$$a = \mathrm{e}^z \quad \text{mit} \quad z = \frac{1}{n} \cdot \left(\sum_{i=1}^{n} \ln y_i - b \cdot \sum_{i=1}^{n} x_i \right).$$

Der *Programmablaufplan* für diese Aufgabe ist in Bild 17 dargestellt. Er gleicht bis auf die beiden Ausdrücke für s_2 und s_3 sowie die Berechnung von a völlig dem Programmablaufplan für die lineare Regression.

Es sei daher dem Leser überlassen, den Programmablaufplan zu testen, das Programm aufzustellen und zu testen sowie eine ausführliche Programmdokumentation hierfür anzufertigen. Sie finden die Lösung dieser Aufgabe wiederum am Ende dieses Buches.

Für den Test des Programmablaufplans und des Programms empfehlen wir folgendes, aus [5] entnommene Beispiel:

x-Werte	1,3	0,6	0,03	1,42	0,93
y-Werte	147,28	11,05	1,34	229,6	37,6

Als Endresultate müssen sich dabei ergeben:

$i = 5$

$s_1 = 4{,}28$ $\qquad$ $s_3 = 19{,}033$

$s_2 = 16{,}7508$ $\qquad$ $s_4 = 4{,}9322$

sowie als Koeffizienten der Gleichung der Regressionsfunktion:

$a = 1{,}2001$ und $\qquad$ $b = 3{,}7006.$

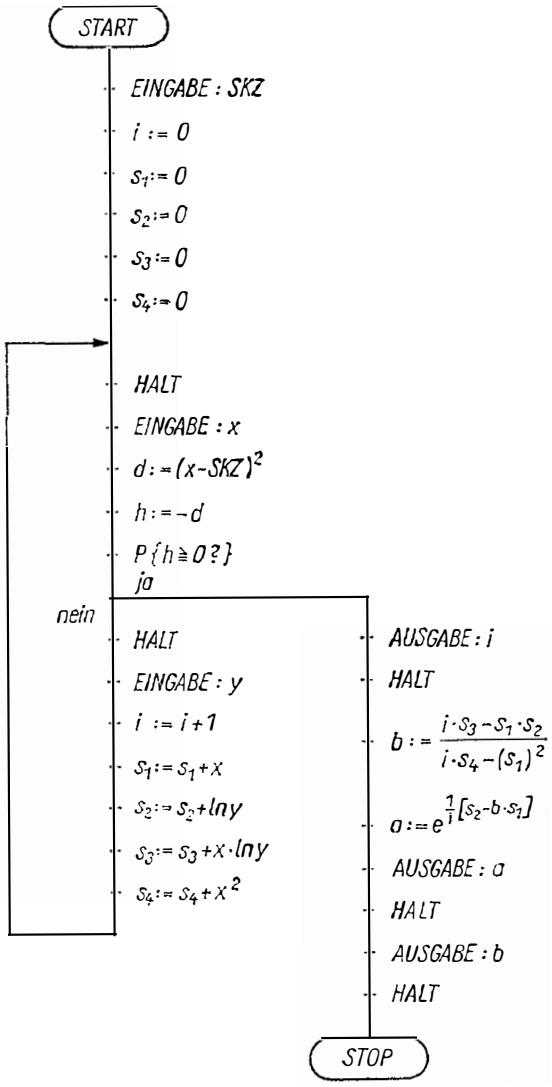

START
EINGABE : SKZ
i := 0
s1 := 0
s2 := 0
s3 := 0
s4 := 0
HALT
EINGABE : x
d := (x-SKZ)²
h := -d
P{h ≧ 0?}
ja
nein
HALT
EINGABE : y
i := i+1
s1 := s1+x
s2 := s2+lny
s3 := s3+x·lny
s4 := s4+x²
AUSGABE : i
HALT
b := (i·s3 - s1·s2) / (i·s4 - (s1)²)
a := e^(1/i [s2-b·s1])
AUSGABE : a
HALT
AUSGABE : b
HALT
STOP

Bild 17

3.4.6. Das Hornersche Schema zur Berechnung von Funktionswerten ganzrationaler Funktionen

Ist der Funktionswert

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_2 x^2 + a_1 x + a_0$$

für ein beliebiges Argument x zu berechnen, so bedeutet dies, daß man in der Regel von der ersten bis zur n-ten Potenz alle Potenzen von x bilden muß, die jeweils mit einem Koeffizienten a_i $(i = 0\,(1)\,n)$ zu multiplizieren sind, und daß schließlich alle so entstehenden Produkte $a_i x$ aufsummiert werden müssen. Ist das Argument x dann auch noch eine „unbequeme" mehrstellige Dezimalzahl, so entsteht doch ein recht erheblicher Rechenaufwand, der auch beim Vorhandensein eines elektronischen Taschenrechners einige Mühe bereitet.
Dieser Aufwand läßt sich wesentlich verringern, wenn man die gegebene Gleichung in folgender Weise umformt:
Klammert man aus dem ersten bis zum vorletzten Glied x aus, so erhält man

$$f(x) = x \cdot (a_n x^{n-1} + a_{n-1} x^{n-2} + \ldots + a_2 x + a_1) + a_0.$$

Nun kann man innerhalb der Klammer wiederum in gleicher Weise ausklammern

$$f(x) = x \cdot (x \cdot (a_n x^{n-2} + a_{n-1} x^{n-3} + \ldots + a_3 x + a_2) + a_1) + a_0.$$

Setzt man dieses Ausklammern innerhalb der innersten Klammer konsequent fort, so ergibt sich schließlich ein der Ausgangsdarstellung für $f(x)$ äquivalenter Ausdruck

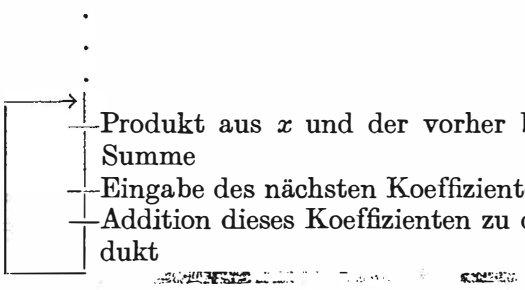
$$f(x) = x \cdot (x \cdot (x \cdot \ldots \cdot x \cdot (a_n x + a_{n-1}) + a_{n-2}) + \ldots + a_3) + a_2) + a_1) + a_0.$$

Im ersten Augenblick sieht dieser Ausdruck viel komplizierter aus als die ursprüngliche Darstellung für $f(x)$. Analysiert man ihn jedoch genauer, so stellt man fest, daß er für die Aufstellung eines Rechenprogramms wesentlich geeigneter ist als die Darstellung mit sämtlichen Potenzen von x.

Man erkennt folgendes: Wenn man die Produktdarstellung der ganzrationalen Funktion $f(x)$ für einen beliebigen x-Wert berechnen soll, dann muß man zunächst a_n mit x multiplizieren und dazu a_{n-1} addieren. Diese Summe wird wiederum mit x multipliziert, und dazu ist a_{n-1} zu addieren. Nun ist die neue Summe mit x zu multiplizieren und dazu a_{n-2} zu addieren usw., bis man schließlich bei a_0 angelangt ist.
Die ursprüngliche Aufgabe, die n Potenzen x^n, x^{n-1}, ..., x^2 und x zu bilden, ferner n Produkte $a_i x^i$ zu berechnen und diese Produkte schließlich zu summieren, reduziert sich damit auf den n-mal sich wiederholenden gleichartigen Rechenvorgang

> Multiplizieren der zuletzt erhaltenen Summe mit x,
> Addieren des nächsten Koeffizienten a_i.

Man nennt dieses Verfahren das HORNERsche Schema zur Berechnung der Funktionswerte ganzrationaler Funktionen (vgl. [3]).
Das *Hornersche Schema* ist vor allem für die Nutzung programmierbarer Rechner geeignet, da man nun nur noch ein Programm zu schreiben hat, das, von der Ein- und Ausgabeorganisation abgesehen, den folgenden Zyklus n-mal wiederholt:



Um das Programm für Funktionen beliebigen Grades verwendbar zu gestalten, ist nur noch eine geeignete *Abbruchbedingung* zu finden, die gewährleistet, daß alle n Durchläufe des angegebenen Zyklus realisiert werden. Im *Programmablaufplan* (Bild 18) ist eine Lösungsmöglichkeit angegeben. Diesmal lassen wir die Zählgröße i nicht wie in den vorangegangenen Beispielen von 0 bzw. 1 bis zum Endwert laufen, sondern wir beginnen mit $i = n$ und verringern i jeweils um 1

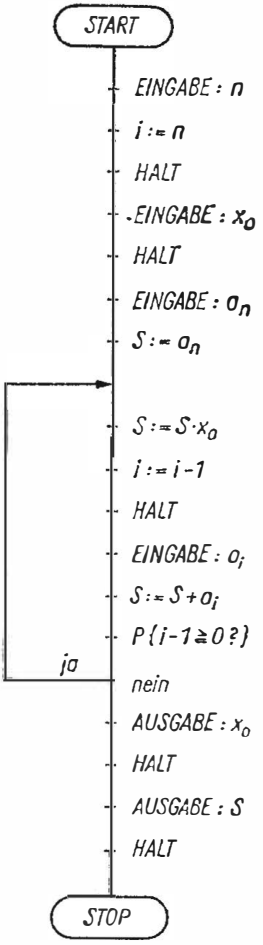


Bild 18

nach einem Durchlauf des Zyklus, bis wir schließlich bei $i = 0$ angelangt sind. Dies hat den Vorteil, daß die Zählgröße i jeweils mit dem Index des entsprechenden Koeffizienten a_i übereinstimmt, wodurch eine bessere Übersichtlichkeit des aufgestellten Programms entsteht.

Test des Programmablaufplans:

Der Programmablaufplan soll an der Funktion

$$\begin{aligned} y &= a_3x^3 + a_2x^2 + a_1x + a_0 \\ &= x \cdot [x \cdot (a_3x + a_2) + a_1] + a_0 \end{aligned}$$

getestet werden.

Programmschritt	Ablauf			
n	3			
i	3			
x_0	x			
a_n	a_3			
s	a_3			
$s := s \cdot x_0$	a_3x	$\rightarrow x \cdot (a_3x + a_2)$	$\rightarrow x \cdot [x \cdot (a_3x + a_2) + a_1]$	
$i := i - 1$	2	1	0	
a_i	a_2	a_1	a_0	
$s := s + a_i$	$a_3x + a_2$	$x \cdot (a_3x + a_2) + a_1$	$x \cdot [x \cdot (a_3x + a_2) + a_1] + a_0$	
$i - 1$	1	0	-1	
$i - 1 \geqq 0$?	ja	ja	nein	
x_0			x	
s			$x \cdot [x \cdot (a_3x + a_2) + a_1] +$	

Der Programmablauf erweist sich als richtig.

Will man das Programm nutzen, um eine *Wertetabelle* für die Funktion $f(x)$ zu berechnen, die mit einem Argument x_0 beginnt und bei einer Schrittweite Δx bis zu einem Endwert x_1 voranschreitet, so kann man die Erhöhung des Argumentwertes x um die Schrittweite Δx auch noch vom Rechner selbständig durchführen lassen. Der ursprüngliche Programmablaufplan braucht dazu nur geringfügig erweitert zu werden. Sie finden diesen erweiterten Programmablaufplan in Bild 19. Das nachfolgende Programm ist auf diese erweiterte Aufgabenstellung bezogen.

Als nachteilig macht es sich dabei bemerkbar, daß bei jeder neuen Berechnung eines Funktionswertes jedesmal wieder sämtliche n Koeffizienten a_n bis a_0 eingetastet werden müssen. Will man auch noch diesen Nachteil beseitigen und das Programm dahingehend verbessern, daß auch die Koeffizienten a_n bis a_0 nur einmal am Anfang des Programms eingegeben werden müssen, so sind dazu programmtechnische Hilfsmittel erforderlich, die den Rahmen dieser Einführung in die Programmierung überschreiten würden und die sich auch nicht auf jedem programmierbaren Taschenrechner realisieren lassen.

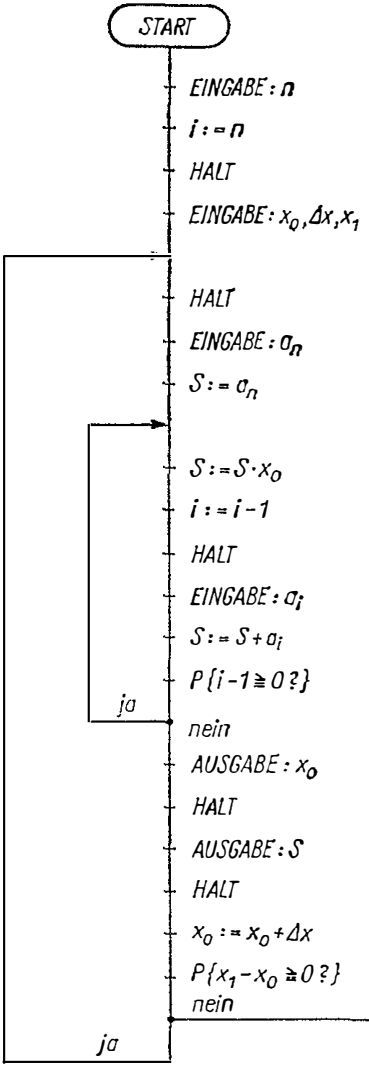

START
EINGABE: n
i := n
HALT
EINGABE: $x_0, \Delta x, x_1$
HALT
EINGABE: a_n
$S := a_n$
$S := S \cdot x_0$
$i := i - 1$
HALT
EINGABE: a_i
$S := S + a_i$
$P\{i - 1 \geq 0 ?\}$
ja
nein
AUSGABE: x_0
HALT
AUSGABE: S
HALT
$x_0 := x_0 + \Delta x$
$P\{x_1 - x_0 \geq 0 ?\}$
nein
ja
STOP

Bild 19

Speicherbelegungsplan:

Bei Nutzung des erweiterten Programms zur Aufstellung einer Wertetabelle sind vor Beginn der Rechnung einzugeben:

Arbeitsspeicher:

Speicher Nummer	Inhalt
4	Δx
5	x_1
0	i
1	x_0
2	a_i
3	s
6	n

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Poln. Notation		
Eingabe	Code	Eingabe	Code	Bemerkung
R/S	13–000	R/S	13–000	HALT für Eingabe des Grades n der ganz-rationalen Funktion
STO 6	64–001	STO 6	64–001	
	73–002		73–002	Speichern von n
STO 0	64–003	STO 0	64–003	Speichern des Anfangswertes von i
	91–004		91–004	
R/S	13–005	R/S	13–005	HALT für Eingabe des Arguments x_0
STO 1	64–006	STO 1	64–006	
	81–007		81–007	Speichern von x_0
R/S	13–008	R/S	13–008	HALT für Eingabe des Koeffizienten a_n
STO 2	64–009	STO 2	64–009	
	82–010		82–010	Speichern von a_n
STO 3	64–011	STO 3	64–011	
	83–012		83–012	Speichern des Anfangswertes von s

RCL 3	65–013	RCL 3	65–013	Beginn der zyklischen Berechnung. Siehe Anmerkung am Ende des Programms
	83–014		83–014	s
×	74–015			
RCL 1	65–016	RCL 1	65–015	
	81–017		81–016	x_0
=	95–018	×	74–017	$s \cdot x_0$
STO 3	64–019	STO 3	64–018	Speichern des neuen
	83–020		83–019	s
RCL 0	65–021	RCL 0	65–020	
	91–022		91–021	i
−	85–023			
1	91–024	1	91–022	1
=	95–025	−	85–023	$i - 1$
STO 0	64–026	STO 0	64–024	Speichern des neuen
	91–027		91–025	i
R/S	13–028	R/S	13–026	HALT für die Eingabe des nächsten Koeffizienten a_i
STO 2	64–029	STO 2	64–027	
	82–030		82–028	Speichern von a_i
+	84–031			
RCL 3	65–032	RCL 3	65–029	
	83–033		83–030	s
=	95–034	+	84–031	$s + a_i$

STO 3	64–035 83–036	STO 3	64–032 83–033	Speichern des neuen s
RCL 0	65–037 91–038	RCL 0	65–034 91–035	i
–	85–039			
1	91–040	1	91–036	1
=	95–041	–	85–037	$i - 1$
X ≧ 0	15–042	X ≧ 0	15–038	Entscheidung, ob $i > 0$ ist
GOTO 013	14–043 91–044 81–045 83–046	GOTO 013	14–039 91–040 81–041 83–042	Sprung zum Beginn des Zyklus, wenn $i > 0$, um die nächste Klammeroperation auszuführen. Ist $i \leqq 0$ erreicht, so bedeutet dies, daß alle Koeffizienten abgearbeitet sind. Dann wird dieser Sprungbefehl übergangen und das Programm mit dem Befehl 047 (043) fortgesetzt, mit dem die Ausgabe des Ergebnisses beginnt.
RCL 1	65–047 81–048	RCL 1	65–043 81–044	x_0
R/S	13–049	R/S	13–045	HALT für Ausgabe x_0
RCL 3	65–050 83–051	RCL 3	65–046 83–047	s
R/S	13–052	R/S	13–048	HALT für Ausgabe $s = f(x_0)$

GOTO 000	14–053	GOTO 000	14–049	Dieser Sprungbefehl zum Programmanfang ist nur zu programmieren, wenn nur ein einziger Funktionswert berechnet werden soll. Sollen Wertetabellen aufgestellt werden, so ist dieser Sprungbefehl zu ersetzen durch den folgenden Befehl:
	91–054		91–050	
	91–055		91–051	
	91–056		91–052	
RCL 6	65–053	RCL 6	65–049	
	73–054		73–050	n
STO 0	64–055	STO 0	64–051	
	91–056		91–052	Speichern des Anfangswertes für i für die Berechnung des nächsten Funktionswertes
RCL 1	65–057	RCL 1	65–053	x_0
	81–058		81–054	
+	84–059			
RCL 4	65–060	RCL 4	65–055	
	71–061		71–056	Δx
=	95–062	+	84–057	$x_0 + \Delta x$
STO 1	64–063	STO 1	64–058	
	81–064		81–059	Speichern des neuen Arguments x_0
−	85–065			
RCL 5	65–066	RCL 5	65–060	
	72–067		72–061	x_1
=	95–068	−	85–062	$x_0 - x_1$
+/−	94–069	+/−	94–063	$x_1 - x_0$
X ≧ 0	15–070	X ≧ 0	15–064	Entscheidung, ob $x_1 \geqq x_0$

GOTO 008	14–071	GOTO 008	14–065	Sprung zum Befehl 008 (Beginn der Berechnung des nächsten Funktionswertes), wenn $x_1 \geqq x_0$. Ist $x_1 < x_0$, so wird dieser Sprungbefehl nicht ausgeführt und es wird zum nächsten Befehl (075 bzw. 069) übergegangen, mit dem zum Programmanfang zurückgesprungen und damit das Programm beendet wird.
	91–072		91–066	
	91–073		91–067	
	62–074		62–068	
GOTO 000	14–075	GOTO 000	14–069	
	91–076		91–070	
	91–077		91–071	
	91–078		91–072	

Anmerkung: Zunächst ist nicht ohne weiteres einzusehen, warum an dieser Stelle s noch einmal ins X-Register zurückgeholt wird, obwohl es sich auf Grund des vorangegangenen Befehls bereits dort befindet. — Es ist aber zu beachten, daß nach der Abarbeitung des Zyklus und dem dann erfolgenden Sprung zum Befehl 008 nicht s im X-Register steht, sondern $i - 1$.

Programmdokumentation:

1. Gewünschte Rechengenauigkeit einstellen.
2. Soll das Programm zur Berechnung einer Wertetabelle genutzt werden, so ist der Endwert x_1 des Arguments im Speicher 5, die Schrittweite Δx der Tabelle im Speicher 4 zu speichern.
3. Start des Programms. Startadresse: 000
4. Eingabe des Grades n der gegebenen ganzrationalen Funktion. R/S-Taste drücken.

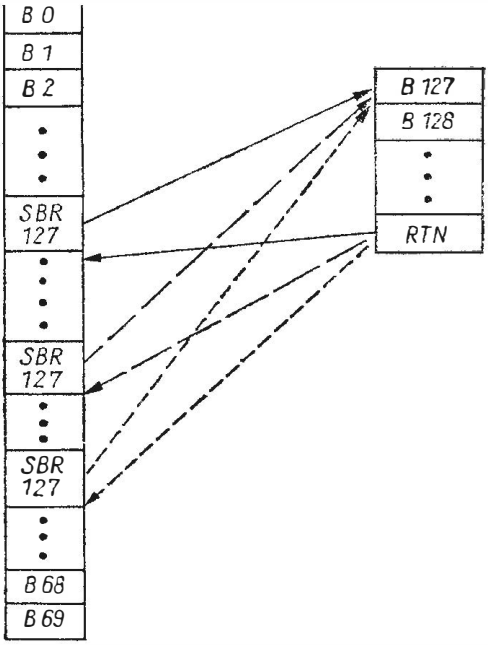
5. Eingabe des Anfangswertes x_0, für den $f(x)$ berechnet werden soll.
   R/S-Taste drücken.
6. Bei jedem folgenden Halt sind der Reihe nach die Koeffizienten a_n, a_{n-1}, a_{n-2}, ..., a_2, a_1, a_0 einzugeben und danach jeweils die R/S-Taste zu drücken.
   ACHTUNG! Es muß für jede Potenz von x ein Koeffizient eingegeben werden. Tritt eine Potenz von x nicht auf, so ist als zugehöriger Koeffizient die Zahl 0 einzugeben.
   Beispiel: Für die Funktion
   $f(x) = 6x^5 - x^3 + 2x^2 - 1{,}5$
   sind der Reihe nach die Koeffizienten
   6, 0, − 1, 2, 0 und − 1.5
   einzugeben.
7. Nach der Eingabe des letzten Koeffizienten a_0 erscheint in der Zahlenanzeige beim ersten HALT das Argument x_0, für das der Funktionswert berechnet wurde. R/S-Taste drücken.
8. Beim nächsten HALT zeigt der Rechner den zugehörigen Funktionswert $f(x_0)$ an. — R/S-Taste drücken.
9. Bei der Anwendung des erweiterten Programms ist nach der Ausgabe des Funktionswertes $f(x_0)$ wieder ab Punkt 5 zu verfahren.

3.5. Unterprogramme

3.5.1. Die SIMPSONsche Regel zur näherungsweisen Berechnung bestimmter Integrale

Mit den Zyklen haben wir eine Möglichkeit kennengelernt, wie wir es erreichen können, daß der Rechner gleichartige Rechenvorgänge, jeweils nur mit anderen Zahlen, ständig wiederholen kann, ohne daß wir diese Rechenvorgänge jedesmal wieder neu programmieren müssen, sondern daß wir sie nur ein einziges Mal im Programm niederschreiben. Voraussetzung für die Anwendung derartiger Programmzyklen ist

es jedoch, daß sich diese gleichartigen Rechenvorgänge *unmittelbar nacheinander* wiederholen, ohne daß etwa nach jeder Wiederholung irgendwelche andere Rechnungen durchgeführt werden müssen.
Nun gibt es aber auch Aufgaben, bei denen gleichartige Algorithmen zwar mehrmals innerhalb des Programms abgearbeitet werden müssen, wo aber zwischen diesen einzelnen Abarbeitungen jedesmal andere Zwischenrechnungen durchzuführen sind, so daß eine Programmierung in Form eines Programmzyklus nicht möglich ist.
Auch in solchen Fällen braucht man den immer wieder auftretenden Algorithmus nur ein einziges Mal zu programmieren, wenn man ihn als *Unterprogramm* schreibt. Derartige Unterprogramme formuliert man *außerhalb des eigentlichen Hauptprogramms*. Man speichert es meist im Anschluß an das Hauptprogramm ab und braucht es dann mit der Anfangsadresse vom Hauptprogramm aus nur mit Hilfe der Taste SBR (subroutine, Unterprogramm) „aufzurufen". Wichtig ist dabei allerdings, daß man die richtigen „Anschlußbedingungen" für die Abarbeitung dieses Unterprogramms schafft, d.h., daß die Größen, die für die Abarbeitung des Unterprogramms benötigt werden, in den richtigen Speichern bzw. Rechenregistern untergebracht werden.
Tritt in einem Hauptprogramm an irgendeiner Stelle der Befehl SBR xyz auf, so merkt sich der Rechner automatisch die Befehlsnummer, von der aus er in das Unterprogramm gegangen ist. Er arbeitet dann den durch das Unterprogramm vorgeschriebenen Algorithmus ab und kehrt dort, wo er am Ende des Unterprogramms den Befehl RTN [RTN (engl.) return zurückkehren] findet, *automatisch* an die Stelle des Hauptprogramms zurück, wo er es vorher verlassen hatte. [Bei manchen Rechnern muß man statt RTN die Tasten INV SBR drücken.]
Die Skizze auf S. 139, die ein Programm darstellen soll, in dem ein Unterprogramm mehrfach aufgerufen wird, soll diesen Vorgang veranschaulichen:
Wie man derartige Unterprogramme in ein Hauptprogramm einbauen kann, soll an der folgenden Aufgabe gezeigt werden:
Es soll ein Programm aufgestellt werden, mit dessen Hilfe die *näherungsweise Integration einer beliebigen Funktion* $f(x)$



in einem Intervall von a bis b durchgeführt werden kann. Als Integrationsverfahren soll die SIMPSONsche Regel verwendet werden.
Bei der näherungsweisen Integration einer Funktion $f(x)$ in den Grenzen von a bis b teilt man das Intervall $[a, b]$ in $2n$ gleich breite Streifen ein, so daß eine Streifenbreite

$$h = \frac{b - a}{2 \cdot n}$$

entsteht. Den angenäherten Wert des bestimmten Integrals erhält man dann aus dem Ausdruck

$$\int_a^b f(x) \cdot \mathrm{d}x = \frac{h}{3} \cdot [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots + 2f(x_{2n-2}) + 4f(x_{2n-1}) + f(x_{2n})],$$

wobei

$$x_0 = a \text{ und } x_{2n} = b$$

gesetzt wurde.

Abgesehen vom ersten und letzten Glied in der Klammer ist eine Summe zu bilden, die wie folgt aufgebaut ist: die Funktionswerte für Argumente mit ungeradem Index sind vor der Summation mit 4, die Funktionswerte für Argumente mit geradem Index nur mit dem Faktor 2 zu multiplizieren.
Wie kann nun der Rechner erkennen, ob eine ganze Zahl i gerade oder ungerade ist?
Es gibt bei vielen Rechnern eine häufig sehr wenig beachtete Funktionstaste, die meist mit [INT] bzw. mit [x] bezeichnet ist.
Die Funktion int(x) ist dabei wie folgt definiert:

int(x) ist der ganzzahlige Anteil einer reellen Zahl x.

Entsprechend gibt es auf vielen Rechnern auch eine Taste [FRAC], die die Funktion frac(x) kennzeichnet. Dabei ist frac(x) wie folgt definiert:

frac(x) ist der gebrochene Anteil der reellen Zahl x.

Beispiele:

int $2.5 = 2$	frac $2.5 = 0.5$
int $13.98 = 13$	frac $13.98 = 0.98$
int $\pi = 3$	frac $\pi = .1415927$

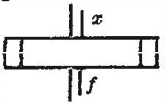
Diese int-Funktion gibt uns nun ein sehr bequemes Hilfsmittel in die Hand, um den Rechner erkennen zu lassen, ob eine ganze Zahl gerade oder ungerade ist.
Teilt man nämlich eine ganze Zahl durch 2, so geht diese Division auf, wenn die Zahl gerade ist, d.h., es ist

$$\text{int}(n/2) = n/2, \text{ wenn } n \text{ gerade und } n \in N.$$

Ist die Zahl n hingegen ungerade, so bleibt bei der Division durch 2 der Rest 0.5, d.h., es ist

$$\text{int}(n/2) \neq n/2, \text{ wenn } n \text{ ungerade und } n \in N.$$

Diese Eigenschaft wollen wir in unserem Programm nutzen, um die Geradzahligkeit eines Index feststellen zu lassen.
Damit erhalten wir für die Aufstellung unseres Programms einen *Ablaufplan*, wie er in Bild 20 dargestellt ist.

Das Symbol 

im Programmablaufplan gibt dabei

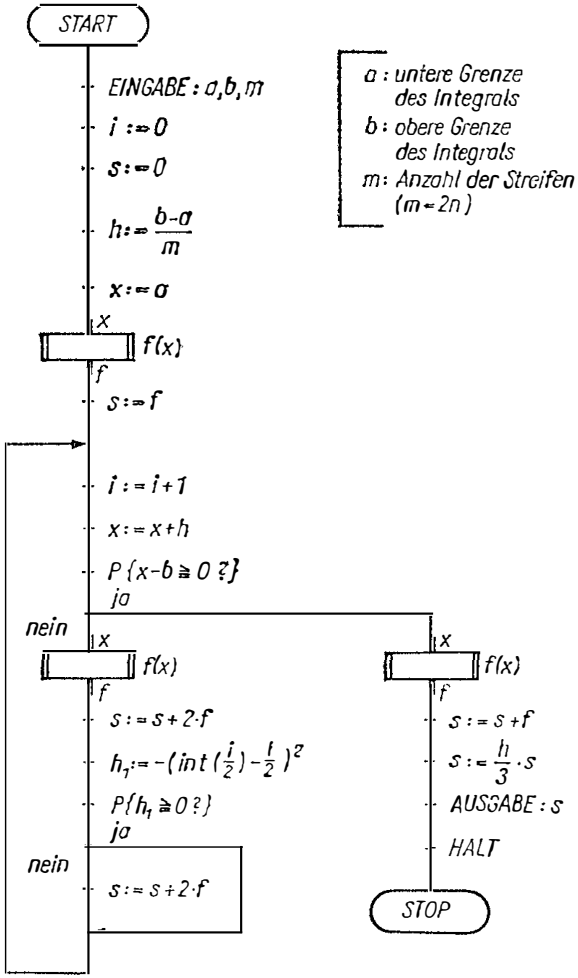

START
EINGABE: a,b,m
i := 0
s := 0
h := (b-a)/m
x := a
x
f(x)
f
s := f
i := i+1
x := x+h
P{x-b ≧ 0 ?}
ja
nein
x
f(x)
f
s := s+2·f
h1 := -(int(i/2) - i/2)²
P{h1 ≧ 0 ?}
ja
nein
s := s+2·f
x
f(x)
f
s := s+f
s := h/3 · s
AUSGABE: s
HALT
STOP
a: untere Grenze des Integrals
b: obere Grenze des Integrals
m: Anzahl der Streifen (m = 2n)

Bild 20

an, daß wir die Funktion $f(x)$ mit Hilfe eines Unterprogramms berechnen lassen wollen. Dabei ist als *Eingangsgröße* für dieses Unterprogramm jeweils nur der Argumentwert x bereitzustellen, der berechnete Funktionswert, der als *Ergebnis des Unterprogramms* entsteht, soll mit f bezeichnet werden.

Die Benutzung eines Unterprogramms für die Funktion $f(x)$ hat auch den Vorteil, daß das Rahmenprogramm für die SIMPSONsche Regel nur einmal formuliert zu werden braucht und daß wir dann für jede neue Funktion, die integriert werden soll, nur noch ein entsprechendes Unterprogramm zu formulieren brauchen.

Das Unterprogramm ist dann in Bild 21 für die Funktion

$$f(x) = (\arcsin(1 - x^2) + 5) \cdot e^{-\sqrt{1-x^2}}$$

dargestellt. Dabei ist es üblich, bei Unterprogrammen den Beginn und das Ende nicht durch das uns von den Hauptprogrammen her bekannte Symbol (START) bzw. (STOP) zu kennzeichnen, sondern durch die beiden Symbole (A) und (B).

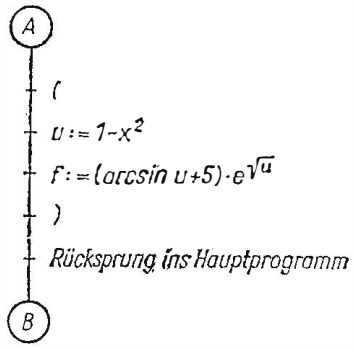


Bild 21

Den Test des Programmablaufplans finden Sie auf Seite 144/145.

Test des Programmablaufplans:

Für den Test wurden folgende Werte angenommen: $m = 6$, $a = x_0$, $b = x_6$, die Zwischenwerte wurden mit x_1 bis x_5 bezeichnet.

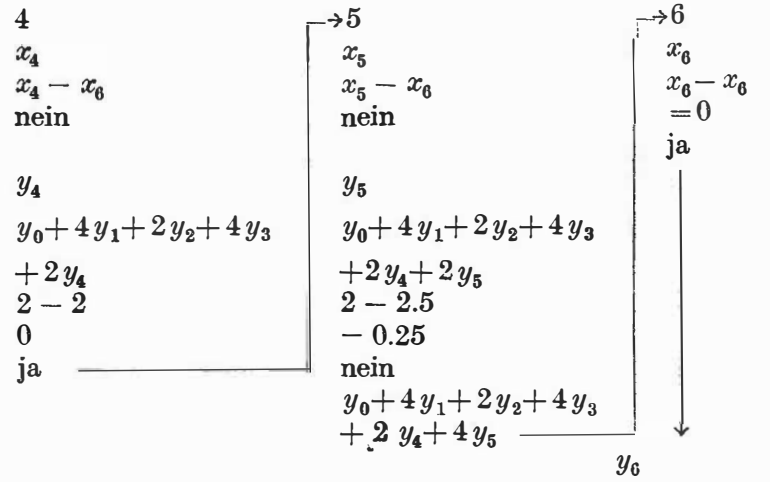
Speicherbelegungsplan:

Vor Beginn der Rechnung brauchen keine Speicherplätze belegt zu werden.

	Speicher	
	Nummer	Inhalt
Arbeitsspeicher:	0	a
	1	b
	2	m
	3	i
	4	s
	5	h
	6	x
	7	f
	8, 9	frei für evtl. Besetzung durch das Unterprogramm

Programmschritt				
Eingabe	x_0, x_6, 6			
i	0			
s	0			
h	$\frac{x_6 - x_0}{6}$			
x	x_0			
f	y_0			
$s := f$	y_0			
$i := i + 1$	1	→ 2	→ 3	
$x := x + h$	x_1	x_2	x_3	
$x - b$	$x_1 - x_6$	$x_2 - x_6$	$x_3 - x_6$	
$x - b \geqq 0$?	nein	nein	nein	
f	y_1	y_2	y_3	
$s := s + 2f$	$y_0 + 2y_1$	$y_0 + 4y_1 + 2y_2$	$y_1 + 4y_1 + 2y_2 + 2y_3$	
$\text{int}\frac{i}{2} - \frac{i}{2}$	$0 - 0.5$	$1 - 1$	$1 - 1.5$	
h_1	-0.25	0	-0.25	
$h_1 \geqq 0$?	nein	ja	nein	
$s := s + 2f$	$y_0 + 4y_1$		$y_0 + 4y_1 + 2y_2 + 4y_3$	
f				
$s := s + f$				
$s := \frac{h}{3} \cdot s$				
Ausgabe				

Der Programmablaufplan erweist sich als richtig.


4
x_4
$x_4 - x_6$
nein
y_4
$y_0+4y_1+2y_2+4y_3$
$+2y_4$
$2-2$
0
ja
→5
x_5
$x_5 - x_6$
nein
y_5
$y_0+4y_1+2y_2+4y_3$
$+2y_4+2y_5$
$2-2.5$
-0.25
nein
$y_0+4y_1+2y_2+4y_3$
$+2y_4+4y_5$
→6
x_6
x_6-x_6
$=0$
ja
y_6
y_0+4y_1
$+2y_2+4y_3$
$+2y_4+4y_5$
$+y_6$
$\frac{h}{3}\cdot(y_0$
$+4y_1+2y_2$
$+4y_3+2y_4$
$+4y_5+y_6)$
dgl.

Programm:

Rechner mit algebr. Logik		Umgek. Poln. Notation		Bemerkung
Eingabe	Code	Eingabe	Code	
R/S	13–000	R/S	13–000	HALT für Eingabe der unteren Grenze a des Integrals
STO 0	64–001	STO 0	64–001	
	91–002		91–002	Speichern von a
R/S	13–003	R/S	13–003	HALT für Eingabe der oberen Grenze b des Integrals
STO 1	64–004	STO 1	64–004	Speichern von b
	81–005		81–005	
R/S	13–006	R/S	13–006	HALT für Eingabe von m
STO 2	64–007	STO 2	64–007	
	82–008		82–008	Speichern von m
0	91–009	0	91–009	0
STO 3	64–010	STO 3	64–010	
	83–011		83–011	Speichern des Anfangswertes von i
STO 4	64–012	STO 4	64–012	
	71–013		71–013	Speichern des Anfangswertes von s
RCL 1	65–014	RCL 1	65–014	
	81–015		81–015	b
–	85–016			
RCL 0	65–017	RCL 0	65–016	
	91–018		91–017	a
÷	75–019	–	85–018	$b - a$

RCL 2	65–020	RCL 2	65–019	
	82–021		82–020	m
=	95–022	÷	75–021	$h = (b - a)/m$
STO 5	64–023	STO 5	64–022	
	72–024		72–023	Speichern von h
RCL 0	65–025	RCL 0	65–024	
	91–026		91–025	$a = x_0$
STO 6	64–027	STO 6	64–026	Speichern des Anfangwertes von x
	73–028		73–027	
F	24–029	F	24–028	
SBR	81–030	SBR	81–029	Aufruf des Unterprogramms zur Berechnung des Funktionswertes $f(x_0)$
1	81–031	1	81–030	
5	72–032	5	72–031	
0	91–033	0	91–032	
STO 4	64–034	STO 4	64–033	
	71–035		71–034	Speichern von $s = f(x_0)$
RCL 3	65–036	RCL 3	65–035	Beginn des Zyklus
	83–037		83–036	i
+	84–038			
1	81–039	1	81–037	1
=	95–040	+	84–038	$i + 1$
STO 3	64–041	STO 3	64–039	Speichern des neuen Wertes von i
	83–042		83–040	
RCL 6	65–043	RCL 6	65–041	
	73–044		73–042	x
+	84–045			

RCL 5	65–046	RCL 5	65–043	
	72–047		72–044	h
=	95–048	+	84–045	$x + h$
STO 6	64–049	STO 6	64–046	
	73–050		73–047	Speichern des neuen x
–	85–051			
RCL 1	65–052	RCL 1	65–048	
	81–053		81–049	b
=	95–054	–	85–050	$x - b$
X ≧ 0	15–055	X ≧ 0	15–051	Entscheidung, ob der Zyklus abgebrochen werden kann ($x = b$) oder nicht ($x < b$).
GOTO	14–056	GOTO	14–052	Sprung zur Endauswertung, wenn $x = b$. Ist $x < b$, so wird dieser Sprungbefehl nicht beachtet und das Programm wird mit Befehl 060 bzw. 056 fortgesetzt.
1	81–057	1	81–053	
1	81–058	0	91–054	
7	61–059	8	62–055	
RCL 6	65–060	RCL 6	65–056	
	73–061		73–057	x
F	24–062	F	24–058	
SBR	81–063	SBR	81–059	Aufruf des Unterprogramms zur Berechnung der Funktion $f(x)$

1	81–064	1	81–060	
5	72–065	5	72–061	
0	91–066	0	91–062	
STO 7	64–067	STO 7	64–063	
	61–068		61–064	Speichern des Funktionswertes f
×	74–069			f
2	82–070	2	82–065	2
+	84–071	×	74–066	$2f$
RCL 4	65–072	RCL 4	65–067	
	71–073		71–068	s
=	95–074	+	84–069	$s + 2f$
STO 4	64–075	STO 4	64–070	
	71–076		71–071	Speichern des neuen s
RCL 3	65–077	RCL 3	65–072	
	83–078		83–073	i
÷	75–079			
2	82–080	2	82–074	2
=	95–081	÷	75–075	$i/2$
F	24–082	F	24–076	
INT	51–083	INT	51–077	int $(i/2)$
×	74–084			
2	82–085	2	82–078	2
–	85–086	×	74–079	$2 \cdot$ int $(i/2)$
RCL 3	65–087	RCL 3	65–080	i
	83–088		83–081	
÷	75–089	–	85–082	$2 \cdot$ int $(i/2) - i$
2	82–090	2	82–083	2

[=]	95–091	[÷]	75–084	$(2 \cdot \text{int}(i/2) - i)/2 = \text{int}(i/2) - i/2$
[F]	24–092	[F]	24–085	
[X²]	34–093	[X²]	61–086	$(\text{int}(i/2) - i/2)^2$
[−]	85–094			
[0]	91–095	[0]	91–087	0
[X↔Y]	54–096	[X↔Y]	54–088	
[=]	95–097	[−]	85–089	$h_1 = -$ $(\text{int}(i/2) - i/2)^2$
[X ≧ 0]	15–098	[X ≧ 0]	15–090	Entscheidung, ob i gerade oder ungerade ist
[GOTO]	14–099	[GOTO]	14–091	Sprung zum Beginn des Zyklus, wenn i gerade ist. Ist i ungerade, so wird dieser Sprungbefehl nicht ausgeführt, sondern das Programm mit dem Befehl 103 bzw. 095 fortgesetzt.
[0]	91–100	[0]	91–092	
[3]	83–101	[3]	83–093	
[6]	73–102	[5]	72–094	
[RCL 7]	65–103	[RCL 7]	65–095	
	61–104		61–096	f
[×]	74–105			
[2]	82–106	[2]	82–097	2
[+]	84–107	[×]	74–098	$2f$
[RCL 4]	65–108	[RCL 4]	65–099	
	71–109		71–100	s

=	95–110	+	84–101	$s + 2f$
STO 4	64–111	STO 4	64–102	
	71–112		71–103	Speichern des neuen s
GOTO	14–113	GOTO	14–104	Rücksprung zum Beginn des Zyklus
0	91–114	0	91–105	
3	83–115	3	83–106	
6	73–116	5	72–107	
RCL 6	65–117	RCL 6	65–108	Beginn der Endauswertung
	73–118		73–109	x
F	24–119	F	24–110	
SBR	81–120	SBR	81–111	Aufruf des Unterprogramms zur Berechnung von $f(x)$
1	81–121	1	81–112	
5	72–122	5	72–113	
0	91–123	0	91–114	f
+	84–124			
RCL 4	65–125	RCL 4	65–115	
	71–126		71–116	s
×	74–127	+	84–117	$s + f$
RCL 5	65–128	RCL 5	65–118	
	72–129		72–119	h
÷	75–130	×	74–120	$h \cdot (s + f)$
3	83–131	3	83–121	3
=	95–132	÷	75–122	$\frac{h}{3} \cdot (s + f)$
R/S	13–133	R/S	13–123	HALT für Ausgabe des Integrals

GOTO	14–134	GOTO	14–124	Rücksprung zum Programmanfang
0	91–135	0	91–125	
0	91–136	0	91–126	
0	91–137	0	91–127	
				Unterprogramm zur Berechnung der Funktion $f(x) =$ $(\arcsin(1-x^2)+5)\cdot e^{-\sqrt{1-x^2}}$
RCL 6	65–150	RCL 6	65–150	
	73–151		73–151	x
F	24–152	F	24–152	
X^2	34–153	X^2	61–153	x^2
−	85–154			
1	81–155	1	81–154	1
X↔Y	54–156	X↔Y	54–155	
= [1]	95–157	−	85–156	$1-x^2$
STO 8	64–158	STO 8	64–157	
	62–159		62–158	Speichern von $u = 1-x^2$
F	24–160	F	24–159	
$\sin^{-1}$	31–161	$\sin^{-1}$	31–160	$\arcsin u$
+	84–162			
5	72–163	5	72–161	5
×	74–164	+	84–162	$\arcsin u + 5$
RCL 8	65–165	RCL 8	65–163	u
	62–166		62–164	
		F	24–165	

$\sqrt{}$	34–167	$\sqrt{}$	62–166	$\sqrt{u}$
+/–	94–168	+/–	94–167	$-\sqrt{u}$
e^x	41–169	e^x	41–168	$e^{-\sqrt{u}}$
= [1]	95–170	×	74–169	$(\arcsin u + 5) \cdot e^{-\sqrt{u}} = f$
F	24–171	F	24–170	
RTN	82–172	RTN	82–171	Rückkehr ins Hauptprogramm

[1]) Bei bestimmten Rechnertypen ist es nicht ratsam, in einem Unterprogramm die [=]-Taste zu verwenden, da mit der Betätigung der [=]-Taste alle bisherigen Rechnungen abgeschlossen werden.

Programmtest:

Als Testbeispiel wurde das Integral

$$J = \int_{-0{,}5}^{0{,}5} (\arcsin(1 - x^2) + 5) \cdot e^{-\sqrt{1 - x^3}} \cdot dx$$

berechnet.

Die Lösung lautet $J = 2{,}388$, wenn als Schrittweite $h = 0{,}1$ verwendet wird.

Programmdokumentation:

1. Gewünschte Rechengenauigkeit einstellen.
2. Unterprogramm für den Integranden ab Befehlsspeicher-Nr. 150 eingeben. — Als Arbeitsspeicher dürfen nur die Speicher 8 und 9 verwendet werden.
3. Programmstart. Startadresse 000
4. Eingabe der unteren Grenze des Integrals.
   [R/S]-Taste drücken.
5. Eingabe der oberen Grenze des Integrals.
   [R/S]-Taste drücken.
6. Eingabe der Anzahl der Streifen, in die das Intervall zerlegt werden soll. (m muß eine gerade Zahl sein.)
   [R/S]-Taste drücken.

7. Beim nächsten HALT erscheint in der Zahlenausgabe der Näherungswert des Integrals.
Ende des Programms.

3.5.2. Das NEWTONsche Verfahren zur näherungsweisen Lösung von Gleichungen

Steht für die Lösung einer Gleichung kein allgemeingültiger Lösungsalgorithmus zur Verfügung, der auf den betreffenden Gleichungstyp zugeschnitten ist, so kann man das NEWTONsche Näherungsverfahren anwenden. Es läßt sich wie folgt charakterisieren:

Man ermittelt zunächst eine *Näherungslösung* x_0 für die Gleichung und bestimmt mit Hilfe der Beziehung

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

eine verbesserte Näherungslösung x_1. Dabei symbolisiert $f(x)$ die zu lösende Gleichung in der Form

$$f(x) = 0,$$

und es wird die Funktion $y = f(x)$ als differenzierbar vorausgesetzt.
Ist die verbesserte Näherungslösung x_1 noch nicht genau genug, so kann man sie als neues x_0 verwenden und mit der obigen Formel ein neues x_1 berechnen. Diese Vorgehensweise wird so lange fortgesetzt, bis die ermittelte Lösung den geforderten Genauigkeitsansprüchen genügt (vgl. [3]).

Man erkennt daraus schon, daß wieder ein Zyklus auftritt, der sich aber von den bisher behandelten Zyklen in gewisser Weise unterscheidet. Während bei den bisherigen Aufgaben, bei denen Zyklen auftraten, eine bestimmte Anzahl von Durchläufen dieser Zyklen zu absolvieren war, kann bei dem hier auftretenden Zyklus *nicht von vornherein* gesagt werden, wie oft er durchlaufen werden muß. Der *Abbruch des Zyklus ist abhängig von dem Genauigkeitsgrad, mit dem die zuletzt gefundene Näherungslösung sich der tatsächlichen Lösung angenähert hat.* Will man erreichen, daß die Näherungslösung sich nicht mehr als um einen Wert ε von der tatsächlichen Lösung unterscheidet, so kann man die weitere Annäherung

abbrechen, wenn zwischen zwei aufeinanderfolgenden Näherungswerten die Beziehung

$$|x_1 - x_0| < \varepsilon$$

besteht.
Damit wir nun das Programm für die Anwendung des NEWTONschen Näherungsverfahrens für alle möglichen Gleichungstypen verwenden können und es nicht für jeden einzelnen Anwendungsfall neu schreiben müssen, wollen wir die beiden in der Näherungsformel auftretenden Ausdrücke $f(x)$ und $f'(x)$ wieder als Unterprogramm in unser Hauptprogramm aufnehmen, so daß in Zukunft nur noch diese beiden entsprechenden Funktionen neu programmiert werden müssen, das Rahmenprogramm aber beibehalten werden kann.
Der zugehörige *Programmablaufplan* ist in Bild 22 dargestellt.
Als Testbeispiel für unser Programm wurde die Gleichung

$$2x + \sin x - 2 = 0$$

aufgenommen.

Test des Programmablaufplans:

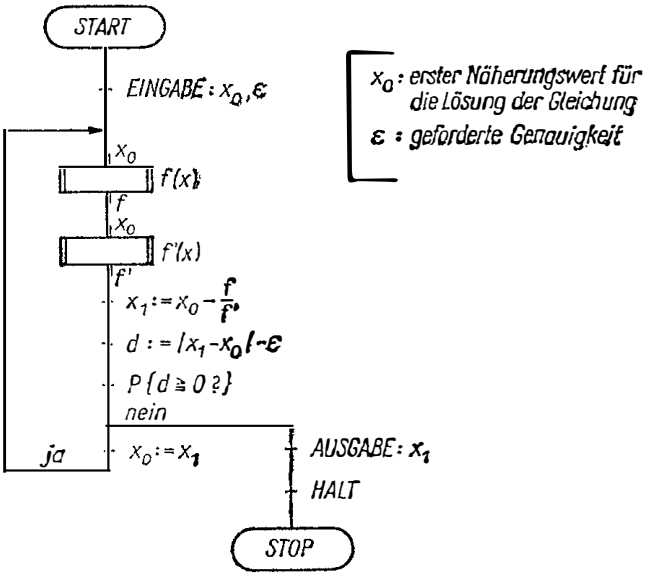
Für den Test des Programmablaufplans soll angenommen werden, daß

$$|x_1 - x_0| > \varepsilon,\ |x_2 - x_1| > \varepsilon \text{ und } |x_3 - x_2| < \varepsilon$$

ist.

Programmschritt	Ablauf		
Eingabe	x_0, ε		
f	y_0	$\rightarrow y_1$	$\rightarrow y_2$
f'	y_0'	y_1'	y_2'
x_1	$x_0 - \frac{y_0}{y_0'}$	$x_1 - \frac{y_1}{y_1'}$	$x_2 - \frac{y_2}{y_2'}$
d	$\lvert x_1 - x_0 \rvert - \varepsilon$	$\lvert x_2 - x_1 \rvert - \varepsilon$	$\lvert x_3 - x_2 \rvert - \varepsilon$
$d \geqq 0$?	ja	ja	nein
x_0	x_1	x_2	↓
Ausgabe			x_3

Der Programmablaufplan erweist sich als richtig.



Unterprogramme für die Gleichung $2x + \sin x - 2 = 0$

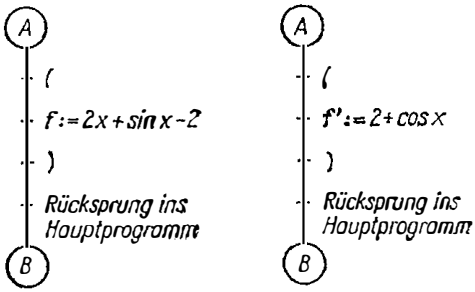


Bild 22

Speicherbelegungsplan:

Vor Beginn der Rechnung brauchen keine Speicher belegt zu werden.

	Speicher	
	Nummer	Inhalt
Arbeitsspeicher:	0	x_0
	1	x_1
	2	ε
	3	f
	ab 4	frei für Unterprogramme

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Poln. Notation		
Eingabe	Code	Eingabe	Code	Bemerkung
R/S	13–000	R/S	13–000	HALT für Eingabe von x_0
STO 0	64–001	STO 0	64–001	
	91–002		91–002	Speichern von x_0
R/S	13–003	R/S	13–003	HALT für Eingabe von ε
STO 2	64–004	STO 2	64–004	
	82–005		82–005	Speichern von ε
RCL 0	65–006	RCL 0	65–006	Beginn des Iterationszyklus
	91–007		91–007	x_0
F	24–008	F	24–008	
SBR	81–009	SBR	81–009	Aufruf des Unterprogramms zur Berechnung von $f(x)$
1	81–010	1	81–010	
0	91–011	0	91–011	
0	91–012	2	82–012	f
STO 3	64–013	STO 3	64–013	
	83–014		83–014	Speichern von $f(x_0)$
RCL 0	65–015	RCL 0	65–015	
	91–016		91–016	x_0
F	24–017	F	24–017	
SBR	81–018	SBR	81–018	Aufruf des Unterprogramms zur Berechnung von $f'(x)$
2	82–019	2	82–019	
0	91–020	0	91–020	

0	91–021	2	82–021	f'
F	24–022	F	24–022	
1/x	83–023	1/x	83–023	$1/f'$
×	74–024			
RCL 3	65–025	RCL 3	65–024	
	83–026		83–025	f
−	84–027	×	74–026	f/f'
RCL 0	65–028	RCL 0	65–027	
	91–029		91–028	x_0
=	95–030	−	85–029	$f/f' - x_0$
+/−	94–031	+/−	94–030	$x_0 - f/f' = x_1$
STO 1	64–032	STO 1	64–031	Speichern von x_1
−	85–033			
RCL 0	65–034	RCL 0	65–032	
	91–035		91–033	x_0
=	95–036	−	85–034	$x_1 - x_0$
F	24–037	F	24–035	
|x|	53–038	|x|	53–036	$\lvert x_1 - x_0 \rvert$
−	85–039			
RCL 2	65–040	RCL 2	65–037	
	82–041		82–038	ε
=	95–042	−	85–039	$\lvert x_1 - x_0 \rvert - \varepsilon$
X ≧ 0	15–043	X ≧ 0	15–040	Entscheidung, ob die gewünschte Genauigkeit erreicht ist ($\lvert x_1 - x_0 \rvert < \varepsilon$) oder nicht
GOTO	14–044	GOTO	14–041	Sprung zum Befehl 055 bzw. 052, von dem ab der nächste Iterationszyklus vorbereitet wird
0	91–045	0	91–042	

5	72–046	5	72–043	Dieser Sprungbefehl wird übergangen, wenn $\|x_1 - x_0\| < \varepsilon$.
5	72–047	2	82–044	In diesem Fall wird die Rechnung mit dem folgenden Befehl (Ergebnisausgabe) fortgesetzt.
RCL 1	65–048	RCL 1	65–045	
	81–049		81–046	x_1
R/S	13–050	R/S	13–047	HALT für Ausgabe des Ergebnisses x_1
GOTO	14–051	GOTO	14–048	
0	91–052	0	91–049	
0	91–053	0	91–050	
0	91–054	0	91–051	Rücksprung zum Programmanfang
RCL 1	65–055	RCL 1	65–052	Vorbereitung des nächsten Iterationszyklus
	81–056		81–053	x_1
STO 0	64–057	STO 0	64–054	
	91–058		91–055	Speichern von x_1 als neues x_0
GOTO	14–059	GOTO	14–056	Rücksprung zum Beginn des Iterationszyklus
0	91–060	0	91–057	
0	91–061	0	91–058	
6	73–062	6	73–059	
				Unterprogramm zur Berechnung von $f(x)$
F	24–100			
(	61–101			

2	82–102	2	82–102	2
×	74–103			
RCL 0	65–104	RCL 0	65–103	
	91–105		91–104	x_0
+	84–106	×	74–105	$2x_0$
RCL 0	65–107	RCL 0	65–106	
	91–108		91–107	x_0
sin	31–109	sin	31–108	$\sin x_0$
−	85–110	+	84–109	$2x_0 + \sin x_0$
2	82–111	2	82–110	2
=	95–112	−	85–111	$2x_0 + \sin x_0 - 2 = f$
F	24–113			
)	62–114			f
F	24–115	F	24–112	
RTN	82–116	RTN	82–113	Rücksprung ins Hauptprogramm
				Unterprogramm zur Berechnung von $f'(x)$
F	24–200			
(	61–201			
2	82–202	2	82–202	2
+	84–203			
RCL 0	65–204	RCL 0	65–203	
	91–205		91–204	x_0
cos	32–206	cos	32–205	$\cos x_0$
=	95–207	+	84–206	$2 + \cos x_0 = f'$
F	24–208			
)	62–209			f'

F	24–210	F	24–207	
RTN	82–211	RTN	82–208	Rücksprung ins Hauptprogramm

Programmtest:

Als Testbeispiel wurde die bereits erwähnte Gleichung

$$2x + \sin x - 2 = 0$$

gelöst:
Ausgehend vom Näherungswert $x_0 = 0{,}5$ ergibt sich die Lösung

$$x = 0{,}684\,0367$$

Geht man von anderen Näherungswerten am Anfang aus, beispielsweise von $x_0 = 10$ bzw. $x_0 = -10$, so erhält man selbstverständlich die gleiche Lösung. Es ist dabei interessant zu beobachten, wie schnell sich der Rechner trotz der unterschiedlichen Anfangswerte an die Lösung herantastet.

Programmdokumentation:

1. Gewünschte Rechengenauigkeit eingeben.
2. Unterprogramme für die Funktionen $f(x)$ und $f'(x)$ eingeben. Das Unterprogramm für $f(x)$ ist ab Speicherplatz 100, das für $f'(x)$ ab Speicherplatz 200 zu speichern.
3. Programm starten.
   Startadresse: 000
4. Ersten Näherungswert x_0 eingeben.
   R/S-Taste drücken.
5. Gewünschte Genauigkeit für das Ergebnis eingeben.
   R/S-Taste drücken.
6. Am Ende der Rechnung erscheint in der Zahlenanzeige die Lösung der Gleichung.

4. Rechner mit hohem Bedienkomfort

Bei den bisherigen Betrachtungen kam es darauf an, den Leser mit den *Grundsätzen des Programmierens* vertraut zu machen, ihn zu befähigen, einfache Programme selbst zu entwerfen und aufzustellen. Damit der Lernende nicht durch das Kennenlernen der vielen speziellen Funktionstasten eines komfortablen programmierbaren Taschenrechners vom eigentlichen Zweck dieses Buches abgelenkt wird, haben wir den bisherigen Programmierbeispielen die beiden fiktiven Rechnertypen KREULOTRON 5 und 6 zugrunde gelegt, die nur die wichtigsten für programmierbare Taschenrechner erforderlichen Operationen auszuführen in der Lage sind. Dieses bewußte Einschränken auf die unbedingt notwendigen Operationen läßt zwar die Grundsätze des Programmierens deutlicher hervortreten, bringt jedoch andererseits auch eine Reihe von Nachteilen mit sich, die auf modernen Taschenrechnern durch zusätzliche Operationstasten vermieden werden. In den folgenden Abschnitten sollen daher einige dieser bisher nicht erwähnten Erweiterungen näher beschrieben werden.

4.1. Kombinierte Codes

Der Aufbau der Befehlsworte für die Rechner KREULOTRON 5 und 6 bringt es mit sich, daß für die Speicherbefehle [STO] [n] und [RCL] [n] sowie für alle Befehle, die die Zweitfunktionstaste [F] benötigen, jeweils *zwei Befehlsworte* geschrieben werden müssen und demzufolge im Befehlsspeicher auch *zwei Speicherplätze* belegt werden. Beim Sprungbefehl [GOTO] [x] [y] [z] werden sogar *vier Speicherplätze* des Befehlsspeichers besetzt.
Bedenkt man, wie oft im Verlauf eines Programms Speicher-

plätze neu belegt, aus Speichern die jeweils dort befindlichen Zahlenwerte aufgerufen oder Sprungbefehle ausgeführt werden müssen, so werden allein durch diese immer wiederkehrenden Operationen sehr viele Speicherplätze des ohnehin meist nicht allzu umfangreichen Befehlsspeichers belegt. Aus diesem Grund werden bei zahlreichen Rechnertypen sogenannte *kombinierte Codes* verwendet, bei denen die genannten Operationen, die die Betätigung mehrerer Tasten erfordern, jeweils *zu einem einzigen Befehlswort zusammengefaßt* werden. Damit belegen natürlich diese Operationen auch jeweils nur *einen einzigen Speicherplatz* des Befehlsspeichers, womit die Kapazität des Befehlsspeichers wesentlich erhöht wird. Es wird dadurch möglich, umfangreichere Programme im Rechner zu speichern.

Am Ende eines jeden Programms haben wir immer den Befehl [GOTO] [0] [0] [0] angefügt, um damit die Möglichkeit einzuräumen, daß das Programm sofort nach Abschluß einer Rechnung mit veränderten Eingangsparametern wiederholt werden kann. Bei vielen Rechnertypen gibt es hierfür eine Taste, die mit [RST] gekennzeichnet ist [RST kommt von „reset" (engl.) zurücksetzen]. Diese Taste *veranlaßt automatisch den Rücksprung zum Programmanfang* und gestattet damit jederzeit die Wiederholung des Programms mit neuen Zahlenwerten.

4.2. Der PAUSE-Befehl

Um dem Menschen die Möglichkeit zu geben, an bestimmten Stellen in ein laufendes Programm einzugreifen, um beispielsweise Zwischenergebnisse ablesen oder neue Zahlwerte eingeben zu können, haben wir in unseren Programmen stets die Taste [R/S] verwendet. Der Rechner unterbricht dann an der vorgesehenen Stelle seine Rechnung, und der Mensch muß, wenn die Arbeit fortgesetzt werden soll, erneut die [R/S]-Taste drücken.

Bei Rechnern, die eine [PAUSE]-Taste besitzen, ist es vorteilhaft, an den Stellen, an denen die Rechnung unterbrochen werden soll, von dieser Taste Gebrauch zu machen. Der Rechner unterbricht dann an dieser Stelle für eine bestimmte Zeit,

meist sind es 45 Sekunden, die Rechnung und setzt sie danach *automatisch* fort, ohne daß der Mensch einzugreifen braucht. Benötigt man eine längere Zeit, um beispielsweise das Zwischenergebnis zu notieren und eine Zahl einzugeben, so kann man durch mehrfache Verwendung der PAUSE-Taste den Haltezustand des Rechners beliebig verlängern.

Durch die Verwendung der PAUSE-Taste ergibt sich zwar *keine Einsparung von Befehlsspeicherplätzen*, es wird jedoch die *Organisation des Rechenablaufs wesentlich erleichtert*, da der Mensch weniger in das Rechengeschehen eingreifen braucht als bei der Verwendung der R/S-Taste.

4.3. Speicherarithmetik

Um einen beliebigen Zahlenwert zum Inhalt des Speichers *n* zu addieren, war bisher die Tastenfolge

ZAHL | + | RCL | n | = | STO | n

erforderlich. Es werden also allein für diese häufig wiederkehrende Rechnung sieben Speicherplätze des Programmspeichers benötigt.

Mit Hilfe der Taste SUM kann die obige Befehlsfolge durch

ZAHL | SUM | n

ersetzt werden, die weniger als die Hälfte der oben benötigten Speicherplätze erfordert. Durch diese Tastenfolge wird jeweils die in der Anzeige und damit im X-Register befindliche ZAHL zum Inhalt des Speichers *n* addiert, und das Ergebnis dieser Addition wird nunmehr im Speicher *n* gespeichert.

Analog wird durch die Tastenfolge

ZAHL | INV | SUM | n

die Zahl, die sich im X-Register bzw. in der Anzeige befindet, vom Inhalt des Speichers *n* subtrahiert und die entstehende Differenz im Speicher *n* gespeichert.

Die Tastenfolge

ZAHL | PRD | n

bewirkt, daß im Speicher n nunmehr das Produkt aus dem vorher vorhandenen Speicherinhalt und dem Inhalt des X-Registers gespeichert wird, und bei der Tastenfolge

ZAHL INV PRD n

wird entsprechend der Quotient aus dem vorher im Speicher n vorhandenen Wert und dem Inhalt des X-Registers im Speicher n gespeichert.

4.4. Statistische Berechnungen

Für die Berechnung des Mittelwertes und der Streuung einer Meßreihe sowie für die Ermittlung der Regressionsfunktion gibt es auf vielen Rechnermodellen Sondertasten wie z.B. $\bar{x}$, σ usw.
Durch die Betätigung dieser Tasten wird die Ausführung von Programmen ausgelöst, wie sie in 3.4.3. und 3.4.4. beschrieben worden sind. Wie die Organisation der Zahleneingabe für den jeweils verwendeten Rechnertyp zu erfolgen hat, muß aus der Bedienanleitung des jeweiligen Rechners entnommen werden.
Baut man die erwähnten Befehle zur Berechnung des Mittelwertes, der Streuung, der Regressionsfunktion usw. in ein umfangreicheres Programm ein, so muß unbedingt beachtet werden, daß diese Teilprogramme eine ganze Reihe von Zahlenspeicherplätzen benötigen, die dann für große Teile des eigentlichen Hauptprogramms blockiert sind und *dort nicht verwendet* werden dürfen. Um welche Speicherplätze es sich dabei handelt, muß wiederum der Bedienanleitung des verwendeten Rechnermodells entnommen werden.

4.5. Marken und Sprungbefehle

Jeder programmierbare Rechner ist darauf eingerichtet, daß er die ihm eingegebenen Befehle *in der Reihenfolge* abarbeitet, *in der sie im Befehlsspeicher untergebracht worden sind*, d.h., in der sie im Programm niedergeschrieben wurden. Nun gibt es aber bei Programmverzweigungen, bei zyklischen Programmen, Stellen, an denen von dieser „natürlichen" Reihen-

folge der Befehlsabarbeitung *abgewichen* werden muß, von denen aus der Rechner zu einer anderen als der unmittelbar nachfolgenden Stelle des Programms übergehen muß.
In den behandelten Beispielen wurde dazu der Sprungbefehl

GOTO X Y Z

verwendet, wobei XYZ die Adresse desjenigen Befehls darstellen soll, mit dem die Rechnung fortzusetzen ist. Der Programmierer muß dabei stets genau wissen, welche Befehlsnummer der Befehl besitzt, mit dem die weitere Rechnung nach einem Sprung oder einer Verzweigung ihren Fortgang nehmen soll.
Wesentlich eleganter läßt sich dieses Problem lösen, wenn man einen Rechner besitzt, mit dem man sogenannte *Marken* oder *Labels* in ein Programm einfügen kann. Die zugehörige Taste ist auf diesen Rechnern meist mit LBL bezeichnet. Die *Label-Taste* ermöglicht die *Markierung bestimmter Befehle* eines Programms, beispielsweise den Beginn eines Zyklus, den Anfang eines Unterprogramms, den Beginn der Endauswertung einer umfangreicheren Rechnung u.a.m.
Man braucht dazu nur im LEARN-Zustand des Rechners vor demjenigen Befehl, der besonders gekennzeichnet werden soll, die Tastenfolge

LBL n

zu drücken, wobei *n* bei den meisten Rechnern eine beliebige Ziffer zwischen 0 und 9, bei einigen Rechnermodellen aber auch eine der Sondertasten A bis E bzw. A′ bis E′ sein darf. Es ist nur darauf zu achten, daß ein und dieselbe Labelnummer in einem Programm *nicht mehrfach* verwendet werden darf.
Soll dann im Verlauf der Rechnung an eine Stelle gesprungen werden, die durch eine derartige Marke, beispielsweise durch die Marke B, gekennzeichnet worden ist, so braucht man nur noch die Befehlsfolge

GOTO B

zu schreiben. (Bisher hätte man die Adresse XYZ des Befehls, mit dem die Rechnung weitergeführt werden soll, wis-

sen und [GOTO] [X] [Y] [Z] programmieren müssen; mit Hilfe der Marken stellt der Rechner diese „Ansprungadresse" selbständig fest.)
Es gibt also bei Rechnern, die eine Label-Taste besitzen, *zwei unterschiedliche Arten von Sprungbefehlen*: den *Sprung zu einer Marke*, der durch [GOTO] [n] einzutasten ist, sowie den *Sprung zu einer bestimmten Befehlsadresse*, der durch [GOTO] [X] [Y] [Z] eingegeben werden muß.
Mit Hilfe von Marken lassen sich natürlich auch die Anfänge von Unterprogrammen leicht kennzeichnen und bei der Programmierung demzufolge auch leicht erreichen. Markiert man beispielsweise den Anfang eines Unterprogramms durch das Label [LBL] [9], so braucht dieses Unterprogramm vom Hauptprogramm aus nur noch durch den Befehl [SBR] [9] aufgerufen zu werden.
Der Ablauf der Rechnung ist in Bild 23 schematisch dargestellt. Dabei bedeuten die leeren Felder die einzelnen Befehle des Haupt- und des Unterprogramms.
Es ist offensichtlich, daß die Labels die Programmierarbeit wesentlich erleichtern können. Der Rechner übernimmt es mit ihrer Hilfe für den Programmierer, sich die Adressen derjenigen Befehle zu merken, an denen die Rechnung nach Verzweigungen fortgesetzt werden soll.
Daß es bei komfortablen Rechnern neben der bei KREUL●-TRON 5 und 6 verwendeten Abfragemöglichkeit [X ≧ 0] auch noch weitere, wie z.B.

[X = 0] [X < 0] [X > 0]
[X ≦ 0] [X ≠ 0] [X > Y]

gibt, darauf wurde bereits hingewiesen. Auch diese Tasten bringen wesentliche Vereinfachungen für die Programmierarbeit mit sich. Ihre Wirkungsweise wird man ohne weiteres einsehen, wenn man die Wirkungsweise des Befehls [X ≧ 0] verstanden hat.
Eine weitere sehr vorteilhafte Möglichkeit, *bedingte Sprünge* zu programmieren, bieten die Rechnertypen, die sogenannte

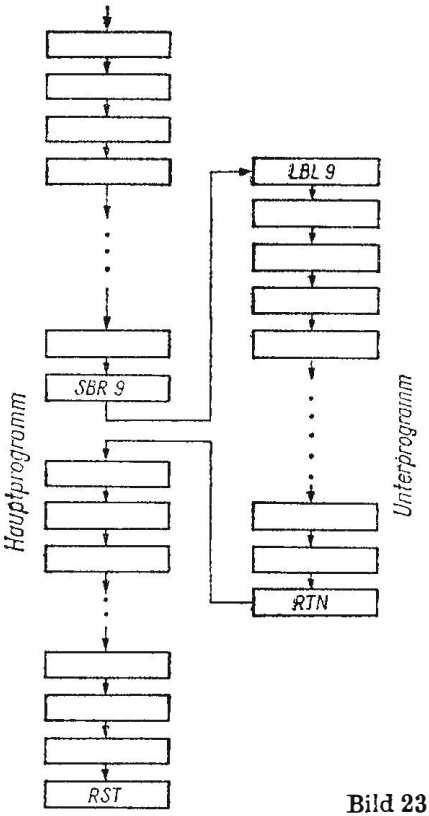


Bild 23

„*Flags*" (Flaggen) besitzen. Es stehen meist mehrere derartige Flags zur Verfügung, die mit den Ziffern 0, 1, ..., *n* gekennzeichnet werden, wobei *n* die Anzahl der vorhandenen Flags bedeutet.
Flags können *an beliebigen Stellen eines Programms gesetzt oder auch gelöscht* werden, und zwar wird durch die Befehlsfolge

ST FLG	n

das Flag Nr. *n* gesetzt, während durch die Befehlsfolge

INV	ST FLG	n

das Flag Nr. *n* gelöscht wird.

Mit der Befehlsfolge

[IF FLG] [n] [A]

wird der Rechner beispielsweise beauftragt,

> nachzuprüfen, ob das Flag Nr. *n* gesetzt ist. Ist dies der Fall, so soll das Programm mit dem Befehl fortgesetzt werden, der durch das Label A markiert worden ist. Ist das Flag Nr. *n* hingegen nicht gesetzt, so wird das Programm in der ursprünglichen Reihenfolge der Befehle fortgesetzt.

In 3.5.1. bereitete uns das Problem, bei der SIMPSONschen Regel die *y*-Werte der zu integrierenden Funktion abwechselnd mit 4 bzw. mit 2 zu multiplizieren, erhebliche Schwierigkeiten, und es mußte ein großer programmtechnischer Aufwand getrieben werden, um dieses Problem zu lösen (vgl. Seite 140). Mit Hilfe eines Flags läßt sich diese Aufgabe wesentlich einfacher bewältigen: Man setzt, bevor man in den Zyklus in Bild 20 eintritt, beispielsweise das Flag 1 und geht danach wie in Bild 24 dargestellt vor.
Wird vorausgesetzt, daß der Anfang des Unterprogramms für die Berechnung des Funktionswertes $f(x)$ mit der Marke D gekennzeichnet worden ist, so erhält dieser Programmteil etwa folgendes Aussehen:

```
···[ST FLG] [1] [LBL] [A] [RCL] [5]
[SUM] [6] [RCL] [6] [−] [RCL] [1]
[=] [X ≧ 0] [GOTO] [C] [SBR] [D]
[STO] [7] [IF FLG] [1] [B] [ST FLG]
[1] [RCL] [7] [·] [2] [=] [SUM] [4]
[GOTO] [A]
...
···[LBL] [B] [RCL] [7] [·] [4] [=]
[SUM] [4] [INV] [ST FLG]
[1] [GOTO] [A]
```

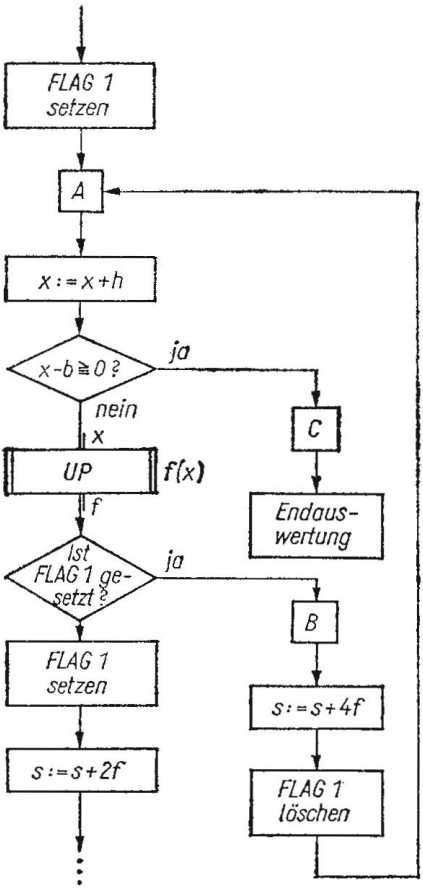


Bild 24

Der Vergleich mit dem in 3.5.1. erarbeiteten Programm zeigt deutlich, daß mit diesen zusätzlichen programmiertechnischen Hilfsmitteln wesentlich kürzere Programme geschrieben werden können.

4.6. Dekrement und Überspringen bei Null

Bei vielen Aufgaben, die auf zyklische Programme führen, weiß man von vornherein, daß der Zyklus genau n-mal durchlaufen werden muß, wobei n eine beliebige ganze positive

Zahl sein kann. Als Beispiel sei hier nur an das Programm für die Berechnung einer Summe mit n Summanden (Abschnitt 3.4.1.) erinnert.
Es mußte dort wie folgt vorgegangen werden: Es wurde eine Zählgröße i gewählt, die den Anfangswert $i = 1$ erhielt. Danach wurde die Befehlsfolge programmiert, die den Zyklus charakterisiert, und nach dem Durchlaufen des Zyklus mußte jedesmal die Frage angeschlossen werden, ob i bereits den Endwert n erreicht hat. War dies der Fall, so konnte zu den Befehlen übergegangen werden, die das Programm fortsetzten, war es hingegen nicht der Fall, so mußte der Zyklus wiederholt werden, nachdem vorher die Zählgröße i um 1 erhöht worden war.

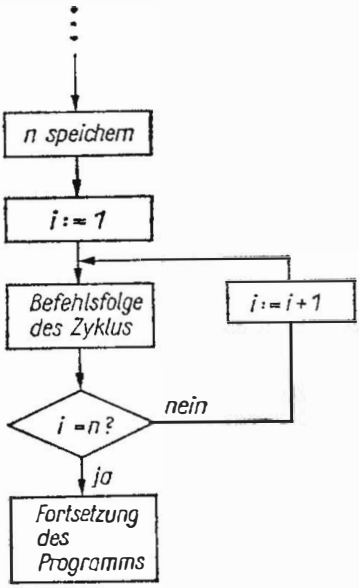


Bild 25

Dieser Ablauf ist in Bild 25 veranschaulicht worden, und das zugehörige Teilprogramm würde etwa wie folgt aussehen:

n	STO	5	1	STO	0	LBL
A	zum Zyklus gehörende Befehlsfolge					
RCL	5	–	RCL	0	=	X ≧ 0
GOTO	A	Fortsetzung des Programms				

Bei zahlreichen Rechnermodellen gibt es nun eine Taste, die mit

DSZ

bezeichnet ist. Die Wirkungsweise dieser Taste DSZ ist wie folgt: *Kommt der Rechner bei der Abarbeitung eines Programms an eine Stelle, an der der Befehl* DSZ *programmiert worden ist, so stellt der Rechner zunächst den Inhalt des Speichers Null fest und vermindert den gefundenen Wert um 1. Danach prüft der Rechner nach, ob der neue Inhalt des Speichers Null gleich Null ist. Ist dies der Fall, so überspringt der Rechner den Befehl, der nach dem Befehl* DSZ *steht und führt den übernächsten Befehl aus. Ist der Inhalt des Speichers Null nicht gleich Null, so wird der auf* DSZ *folgende Befehl ausgeführt.*

Aus dieser Beschreibung der Wirkungsweise des Befehles DSZ geht hervor, daß der Speicher Null in Programmen, die diesen Befehl verwenden, eine besondere Rolle spielt.

Mit dem Befehl DSZ ist es auf sehr einfache Weise möglich, zu erreichen, daß ein Zyklus genau n-mal durchlaufen wird. Man braucht dazu nur wie folgt vorzugehen:

- Die Anzahl der erforderlichen Durchläufe des Zyklus, also die Zahl n, wird im Speicher Null gespeichert.
- Am Anfang des Zyklus wird eine Marke gesetzt.
- Es folgt die Programmierung der zum Zyklus gehörenden Befehlsfolge, die durch den Befehl DSZ abgeschlossen wird. (Durch diesen Befehl wird die Zahl n nach jedem Durchlauf des Zyklus um 1 erniedrigt.)
- Nach dem Befehl DSZ wird ein Sprungbefehl zu der oben eingeführten Marke eingefügt, und im Anschluß daran werden die restlichen Befehle des Programms angefügt.
  (Erst wenn der Inhalt des Speichers Null bis zum Wert Null erniedrigt worden ist — dies ist genau nach n Durchläufen des Zyklus der Fall — wird der Sprungbefehl übergangen und die Rechnung entsprechend den nachfolgenden Befehlen fortgesetzt.) Der Ablauf dieses Programms ist in Bild 26 veranschaulicht.

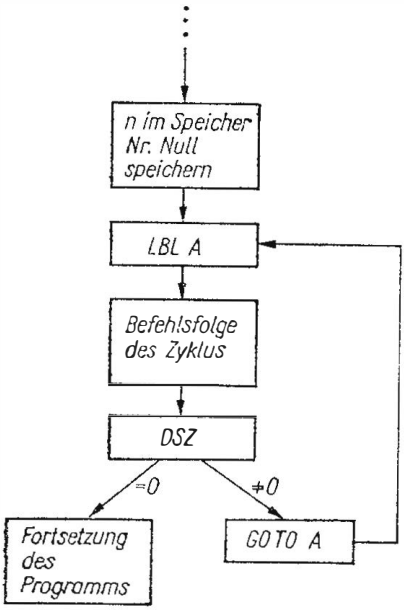


Bild 26

Die hierzu gehörende Befehlsfolge lautet wie folgt:

n	STO	0	LBL	A
zum Zyklus gehörende Befehlsfolge				DSZ
GOTO	A	Fortsetzung des Programms		

Der Vergleich mit der ursprünglichen Befehlsfolge läßt noch einmal die Vorteile der Labels sowie des DSZ-Befehles deutlich hervortreten. Es ist ersichtlich, daß sich viele Programmteile mit Hilfe dieser Befehle wesentlich eleganter und leichter formulieren lassen als bisher und daß damit auch eine spürbare Entlastung des Befehlsspeichers erreicht wird.

4.7. Unterprogramme

Unterprogramme wurden bereits in 3.5. behandelt, und in 4.5. wurde ergänzend dargestellt, wie man den Aufruf eines Unterprogramms durch die Verwendung von Marken vereinfachen kann.

Nun kann es ohne weiteres sein, daß *innerhalb eines Unterprogramms weitere Unterprogramme* aufgerufen werden müssen. Man spricht dann von Unterprogrammen in einer zweiten, dritten, ... Unterprogramm-Ebene.
Wenn ein Rechner in der Lage ist, Unterprogramme aus verschiedenen Unterprogramm-Ebenen zu verarbeiten, dann muß er die Möglichkeit besitzen, in einem Hilfsregister die unterschiedlichen Rücksprungadressen für die einzelnen übergeordneten Unterprogramme zu speichern. Dies ist nicht in unbeschränktem Maße möglich. Die Anzahl der bei einem bestimmten Rechnermodell möglichen Unterprogramm-Ebenen muß wiederum der Bedienanleitung des jeweiligen Rechners entnommen werden.

4.8. Korrektur fehlerhafter Programme

Auf die Notwendigkeit, jedes erarbeitete Programm genauestens auf seine Richtigkeit hin zu überprüfen, bevor man es anwendet, wurde bereits mehrfach hingewiesen. Bereits in Abschnitt 2. wurden Möglichkeiten beschrieben, wie man fehlerhafte Programme korrigieren kann. Dabei wurde für die Rechner KREULOTRON 5 und 6 angenommen, daß die Befehle im Befehlsspeicher fest auf den Speicherplätzen verbleiben, auf die sie eingegeben wurden, und daß man sie nur durch Löschen bzw. durch Überschreiben verändern kann.
Um eine Korrektur durchführen zu können, mußte man sich mit Hilfe der [SST]-Taste bzw. der [BST]-Taste schrittweise an die betreffende Stelle des Programms herantasten oder man konnte die Tastenfolge [GOTO] [X] [Y] [Z] im Betriebszustand RUN verwenden, um unmittelbar den gewünschten Befehl XYZ zu erreichen.
Konnte die Programmkorrektur nicht durch einfaches Abändern der fehlerhaften Befehle bewerkstelligt werden, so mußte man einen [GOTO]-Befehl einfügen, das korrigierte Programmstück am Ende des Programms anhängen und nach der Korrektur erneut mit einem [GOTO]-Befehl an die Stelle des ursprünglichen Programms zurückspringen, von der aus das Programm wieder fehlerfrei war.
Komfortablere Rechnermodelle besitzen noch einige zusätz-

liche Tasten, die die Korrektur von Programmen erleichtern.

Mit der Taste DEL [delete (engl.) auslöschen] kann man einen *Einzelbefehl eines Programms löschen.* Die nachfolgenden Befehle des Programms rücken dann selbständig *um eine Stelle nach vorn,* so daß die durch das Löschen des Befehls im Programmspeicher entstandene *Lücke automatisch geschlossen* wird.

Mit der Taste INS [insert (engl.) einfügen] kann man *zusätzliche Befehle in ein Programm einfügen.* Alle Befehle, die nachfolgen, werden selbständig bei jeder neuen Einfügung *um einen Platz nach hinten* gerückt, so daß durch das Einfügen neuer Befehle keiner der bereits vorhandenen Befehle verlorengeht.

Der Befehl NOP [no operation (engl.) keine Operation] *wird vom Rechner ignoriert.* Er ermöglicht die Bildung freier Befehlsspeicherplätze in einem Programm, die man später bei einer evtl. vorgesehenen Programmerweiterung noch besetzen möchte.

Natürlich kann man mit Hilfe des Befehles NOP auch fehlerhafte Befehle eines Programms überschreiben, so daß sie dann bei der Abarbeitung des Programms *nicht mehr wirksam* werden können.

5. Zusammenfassung und Ausblick

Die in den vorangegangenen Abschnitten behandelten Beispiele und Ergänzungen sollten zeigen, daß das *Programmieren für elektronische Taschenrechner kein Buch mit sieben Siegeln* darstellt, sondern daß es bei einigem Fleiß, mit Ausdauer und Energie durchaus erlernbar ist, daß aber für das Aufstellen von Rechenprogrammen einige Grundregeln zu beherzigen sind, die man konsequent einhalten sollte. Sie sollen abschließend noch einmal zusammengefaßt werden.

Als erste Grundvoraussetzung für ein erfolgreiches Arbeiten mit einem programmierbaren Taschenrechner ist zu nennen, daß man die *Rechenlogik* seines Rechners genauestens kennt. Bevor man an das Programmieren von Aufgaben herangeht, muß man sich mit seinem Rechner erst einmal als nichtprogrammierbares Gerät soweit vertraut gemacht haben, daß man alle anfallenden Rechnungen mit ihm ohne lange nachdenken zu müssen eintippen kann. Danach sollte man sich erst eingehend mit der Wirkungsweise solcher für die Programmierung wichtigen Tasten wie [X ≧ 0], [DSZ], [GOTO] usw. vertraut machen. Hat man die Möglichkeiten dieser Tasten für die Programmierung genau erfaßt, dann kann man darangehen, erste kleinere Programme zu entwickeln. Dabei sollten folgende Punkte beachtet werden:

- Ein Programm für die Lösung einer bestimmten Aufgabe sollte man dann erst aufstellen, nachdem man sich selbst *über alle Einzelheiten der Aufgabenstellung völlig* klar geworden ist. Rechenprogramme lassen sich nicht „aus dem Ärmel schütteln"!
- Ein wesentlicher Schritt für die Entwicklung eines effektiven Programms ist die Suche nach einem für programmierbare Rechner *geeigneten Rechenverfahren* zur Lösung der gestellten Aufgabe.
- Der *Lösungsweg* muß bis in alle Einzelheiten *gründlich durchdacht* worden sein, bevor man an die Programmierung

herangeht. Dabei ist auch festzulegen, welche *Daten* für die Rechnung benötigt werden und auf welchen *Speicherplätzen* man diese Daten und evtl. wichtige Zwischenergebnisse unterbringen will.

- Die Aufstellung eines ausführlichen *Programmablaufplans* lohnt sich immer. Der Programmablaufplan erleichtert das Schreiben des „Maschinenprogramms" wesentlich!
- Bevor der Programmablaufplan in ein Programm umgesetzt wird, sollte er einem gründlichen und sehr kritischen *Test auf seine Richtigkeit hin* unterzogen werden.
- Bei der Aufstellung des Rechenprogramms ist besondere Sorgfalt walten zu lassen, damit *keine Flüchtigkeitsfehler* auftreten. Sehr vorteilhaft ist es, wenn man sich bei jedem Programmschritt überlegt, welche Registeroperationen dabei im Rechner ablaufen.
- Man muß stets einen genauen Überblick darüber besitzen, welche Größen in den einzelnen *Speichern* untergebracht sind und durch welche *Befehlsnummern* bzw. *Marken* diejenigen Programmstellen gekennzeichnet sind, die für den weiteren Programmverlauf eine besondere Bedeutung haben (Beginn eines Zyklus, eines Unterprogramms usw.).
- Auch das aufgestellte Programm sollte auf Herz und Nieren *überprüft* werden, bevor man es für Nutzrechnungen freigibt.
- Schließlich sollte man zu jedem erarbeiteten Programm eine *ausführliche Programmdokumentation* anfertigen, damit man es im Bedarfsfall später ohne große Vorbereitungsarbeiten erneut nutzen kann.

Aus dem Englischen stammt der wahre Ausspruch: Nobody is perfect, zu deutsch: Kein Mensch ist vollkommen. Eingabefehler, logische Trugschlüsse und andere Irrtümer lassen sich nun einmal nicht vollkommen vermeiden. Daher sollte man nach der Fertigstellung einer Teilaufgabe im Gesamtkomplex der Programmierung sich *lieber einmal zu viel als einmal zu wenig* davon überzeugen, ob auch alles richtig erarbeitet worden ist.

Abschließend sollen noch einige Bemerkungen über die technische Weiterentwicklung der programmierbaren Taschenrechner gemacht werden.

Als wichtigstes Kriterium für die Leistungsfähigkeit eines

programmierbaren Taschenrechners sollte nicht die Vielzahl der auf ihm vorhandenen vorprogrammierten Funktionen gesehen werden, sondern in erster Linie die Anzahl der in ihm vorhandenen *Speicherplätze des Programm- und des Datenspeichers*. Viele der handelsüblichen programmierbaren Taschenrechner besitzen leider nur weniger als 100 Programmspeicherplätze. Wie aus den in dieser Einführung behandelten Beispielen hervorgeht, ist dadurch der Anwendungsbereich dieser Rechner wesentlich eingeschränkt. Man kann nur relativ kleine Aufgaben mit ihrer Hilfe programmieren und lösen. Dennoch stellen auch sie bereits eine spürbare Hilfe für den Nutzer dar.

Die Tendenz der Weiterentwicklung geht sinnvollerweise zu *Rechnern mit größerer Programm- und Datenspeicherkapazität*. Es gibt auch bereits programmierbare elektronische Taschenrechner, bei denen sich Programm- und Datenspeicher den Bedingungen *der jeweiligen Aufgabe anpassen* lassen. Für kurze Rechnungen, bei denen ein umfangreiches Datenmaterial zu verarbeiten ist, kann der Hauptteil des Speichers für die Unterbringung der Daten verwendet werden, während nur ein kleiner Befehlsspeicherraum genutzt wird. Hingegen können, insbesondere bei komplizierten wissenschaftlich-technischen Problemen Programme mit mehr als 900 Einzelbefehlen in solchen Taschenrechnern untergebracht werden. In diesem Fall ist jedoch dann die Zahlenspeicherkapazität stark eingeschränkt.

Ein weiterer Nachteil der derzeit erhältlichen programmierbaren Taschenrechner besteht darin, daß bei den meisten Rechnermodellen die gespeicherten Programme zerstört werden, sobald der Rechner ausgeschaltet wird. Das bedeutet, daß das benötigte Programm nach dem Wiedereinschalten des Gerätes zunächst einmal von neuem eingetastet werden muß. Es gibt jedoch neuerdings auch schon Rechnertypen mit einem sogenannten „*continuous memory*", einem bleibenden Gedächtnis, die ein einmal im Programmspeicher gespeichertes Programm so lange aufbewahren, bis es bewußt gelöscht wird. Dabei spielt es keine Rolle, wie oft der Rechner in der Zwischenzeit ein- und ausgeschaltet worden ist.

Eine weitere Möglichkeit, einmal aufgestellte Programme ohne wesentlichen Eingabeaufwand immer wieder verwenden zu können, besteht darin, diese Programme auf schmalen *Magnetstreifen* zu speichern und diese Magnetstreifen bei Be-

darf durch eine besondere Eingabevorrichtung in den Rechner einzugeben. Dabei geht der Service einzelner Herstellerfirmen so weit, daß sie vorprogrammierte Magnetstreifen für bestimmte, häufig auftretende Aufgabenklassen zur Verfügung stellen.
Schließlich gibt es auch Taschenrechner, an die *Magnetbandkassetten* angeschlossen werden können, so daß mit deren Hilfe auch Probleme abgearbeitet werden können, bei denen erhebliche Datenmengen anfallen. — Komfortable Taschenrechner besitzen auch Vorrichtungen zum Anschluß an kleine *Druckwerke*, die mit Hilfe eines Thermokopierverfahrens in der Lage sind, die eingegebenen Programme sowie wichtige Zwischenwerte und Ergebnisse der durchgeführten Rechnungen ausdrucken zu lassen.
Die neuesten Weiterentwicklungen in der Herstellung programmierbarer elektronischer Taschenrechner bestehen darin, daß Rechner gefertigt werden, bei denen es möglich ist, sogenannte „*Moduleinheiten*" anzuschließen, die nicht größer als ein Daumennagel sind. Diese Moduleinheiten enthalten etwa 25 *fertige Programme* mit insgesamt etwa 5000 Programmschritten für die unterschiedlichsten Anwendungsgebiete. Die Einzelprogramme eines derartigen Moduls können jederzeit aufgerufen und beispielsweise auch als Unterprogramme in selbst zu erarbeitende Hauptprogramme eingebaut werden. Damit wird die Leistungsfähigkeit der programmierbaren Taschenrechner natürlich außerordentlich erhöht.
Ob man für einen programmierbaren Taschenrechner oder für eine große Datenverarbeitungsanlage ein Programm aufstellt, immer muß man sich darüber im klaren sein, *daß jede Aufgabe auf unterschiedliche Weisen programmiert werden kann*. Es gilt, die effektivste Lösung für den jeweils verwendeten Rechnertyp herauszufinden. Dies kann man nur *durch ständiges Üben und durch klares, logisches Denken* erreichen. Dieses Buch sollte dafür einige Anregungen und nützliche Hinweise vermitteln. Einen perfekten Programmierer kann es jedoch aus dem Leser nicht machen. Dazu muß er selbst viel Mühe und Übung aufwenden, wenn er es werden will. Aber diese Mühe lohnt sich, denn die programmierbaren elektronischen Taschenrechner stellen ein wertvolles Arbeitsmittel für denjenigen dar, der sehr viele zeitaufwendige und ständig wiederkehrende Aufgabenstellungen zu lösen hat.

In den nächsten Jahrzehnten werden programmierbare Geräte zunehmend in alle Gebiete des täglichen Lebens eingreifen. Wer sich heute bereits eingehend mit den Grundlagen der Programmierung beschäftigt, wird dann auf den Einsatz derartiger Geräte wohl vorbereitet sein, und er wird sie dann wirkungsvoll für die Lösung seiner Aufgaben einsetzen können.

Programm und Programmdokumentation zur linearen Regression (Abschnitt 3.4.4.)

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Poln. Notation		
Eingabe	Code	Eingabe	Code	Bemerkungen
R/S	13–000	R/S	13–000	HALT für die Eingabe des Schlußkennzeichens *skz*
STO 0	64–001	STO 0	64–001	Speichern von *skz*
	91–002		91–002	
0	91–003	0	91–003	0
STO 1	64–004	STO 1	64–004	Speichern des Anfangswertes von i
	81–005		81–005	
STO 2	64–006	STO 2	64–006	Speichern des Anfangswertes von s_1
	82–007		82–007	
STO 3	64–008	STO 3	64–008	Speichern des Anfangswertes von s_2
	83–009		83–009	
STO 4	64–010	STO 4	64–010	Speichern des Anfangswertes von s_3
	71–011		71–011	
STO 5	64–012	STO 5	64–012	Speichern des Anfangswertes von s_4
	72–013		72–013	

R/S	13–014	R/S	13–014	HALT für die Eingabe eines x-Wertes. Beginn der Auswertung der Meßreihe
STO 6	64–015	STO 6	64–015	Speichern des x-Wertes
	73–016		73–016	x
–	85–017			
RCL 0	65–018	RCL 0	65–017	
	91–019		91–018	skz
=	95–020	–	85–019	$x - skz$
STO 8	64–021	ENTER	95–020	
	62–022			
×	74–023			
RCL 8	65–024			
	62–025			
=	95–026	×	74–021	$(x - skz)^2 = d$
–	85–027			
0	91–028	0	91–022	0
X↔Y	54–029	X↔Y	54–023	
=	95–030	–	85–024	$h = -d$
X ≧ 0	15–031	X ≧ 0	15–025	Entscheidung, ob das Ende der Meßreihe erreicht ist ($h = 0$) oder nicht ($h < 0$).
GOTO	14–032	GOTO	14–026	Sprung zum Befehl 088 bzw. 076, wenn $h = 0$, bei dem die Endauswertung beginnt. Bei $h < 0$ wird die Rechnung mit dem folgenden Befehl fortgesetzt.

0	91–033	0	91–027	
8	62–034	7	61–028	
8	62–035	6	73–029	
R/S	13–036	R/S	13–030	HALT für die Eingabe des zugehörigen y-Wertes
STO 7	64–037	STO 7	64–031	
	61–038		61–032	Speichern des y-Wertes
RCL 1	65–039	RCL 1	65–033	
	81–040		81–034	i
+	84–041			
1	91–042	1	91–035	1
=	95–043	+	84–036	$i + 1$
STO 1	64–044	STO 1	64–037	
	81–045		81–038	Speichern des neuen i
RCL 2	65–046	RCL 2	65–039	
	82–047		82–040	s_1
+	84–048			
RCL 6	65–049	RCL 6	65–041	
	73–050		73–042	x
=	95–051	+	84–043	$s_1 + x$
STO 2	64–052	STO 2	64–044	
	82–053		82–045	Speichern des neuen s_1
RCL 3	65–054	RCL 3	65–046	
	83–055		83–047	s_2
+	84–056			
RCL 7	65–057	RCL 7	65–048	
	61–058		61–049	y
=	95–059	+	84–050	$s_2 + y$

STO 3	64–060	STO 3	64–051	
	83–061		83–052	Speichern des neuen s_2
RCL 6	65–062	RCL 6	65–053	
	73–063		73–054	x
×	74–064			
RCL 7	65–065	RCL 7	65–055	
	61–066		61–056	y
+	84–067	×	74–057	xy
RCL 4	65–068	RCL 4	65–058	
	71–069		71–059	s_3
=	95–070	+	84–060	$s_3 + xy$
STO 4	64–071	STO 4	64–061	
	71–072		71–062	Speichern des neuen s_3
RCL 6	65–073	RCL 6	65–063	
	73–074		73–064	x
×	74–075	ENTER	95–065	
RCL 6	65–076			
	73–077			x
+	84–078	×	74–066	x^2
RCL 5	65–079	RCL 5	65–067	
	72–080		72–068	s_4
=	95–081	+	84–069	$s_4 + x^2$
STO 5	64–082	STO 5	64–070	
	72–083		72–071	Speichern des neuen s_4
GOTO	14–084	GOTO	14–072	Rücksprung zum Befehl 014, bei dem die Eingabe des nächsten x-Wertes vorbereitet wird.
0	91–085	0	91–073	

1	81–086	1	81–074	
4	71–087	4	71–075	
RCL 1	65–088	RCL 1	65–076	Beginn der Endauswertung der Meßreihe
	81–089		81–077	i
R/S	13–090 74–091	R/S	13–078	HALT, Ausgabe der Anzahl i der Wertepaare der Meßreihe
RCL 4	65–092	RCL 4	65–079	
×	71–093		71–080	s_3
–	85–094	×	74–081	$i \cdot s_3$
F	24–095			
(	61–096			
RCL 2	65–097	RCL 2	65–082	
	82–098		82–083	s_1
×	74–099			
RCL 3	65–100	RCL 3	65–084	
	83–101		83–085	s_2
F	24–102			
)	62–103	×	74–086	$s_1 \cdot s_2$
=	95–104	–	85–087	$i \cdot s_3 - s_1 \cdot s_2$
STO 8	64–105			
	62–106			
RCL 1	65–107	RCL 1	65–088	
	81–108		81–089	i
×	74–109			
RCL 5	65–110	RCL 5	65–090	
	72–111		72–091	s_4
–	85–112	×	74–092	$i \cdot s_4$

F	24–113			
(	61–114			
RCL 2	65–115	RCL 2	65–093	
	82–116		82–094	s_1
×	74–117	ENTER	95–095	
RCL 2	65–118			
	82–119			s_1
F	24–120			
)	62–121	×	74–096	s_1^2
=	95–122	−	85–097	$is_4 - s_1^2$
÷	75–123			
RCL 8	65–124			
	62–125			$i \cdot s_3 - s_1 \cdot s_2$
X↔Y	54–126			
=	95–127	÷	75–098	$b = (is_3 - s_1 s_2)/(is_4 - s_1^2)$
R/S	13–128	R/S	13–099	HALT, Ausgabe des Regressionskoeffizienten b
×	74–129			b
RCL 2	65–130	RCL 2	65–100	
	82–131		82–101	s_1
−	85–132	×	74–102	$b \cdot s_1$
RCL 3	65–133	RCL 3	65–103	
	83–134		83–104	s_2
X↔Y	54–135	X↔Y	54–105	
÷	75–136	−	85–106	$s_2 - b \cdot s_1$
RCL 1	65–137	RCL 1	65–107	
	81–138		81–108	i
=	95–139	÷	75–109	$a = (s_2 - b \cdot s_1)/i$

R/S	13–140	R/S	13–110	HALT, Ausgabe von a
GOTO	14–141	GOTO	14–111	Rücksprung zum Programmanfang
0	91–142	0	91–112	
0	91–143	0	91–113	
0	91–144	0	91–114	

Programmdokumentation:

1. Start des Programms
   Startadresse: 000
2. Eingabe des Schlußkennzeichens *skz*. R/S-Taste drücken.
3. Bei jedem nachfolgenden Halt ist jeweils ein x-Wert und beim nachfolgenden Halt der zugehörige y-Wert einzugeben. Nach jeder Eingabe ist die R/S-Taste zu drücken.
4. Am Ende der Meßreihe ist noch einmal das Schlußkennzeichen *skz* einzugeben und danach die R/S-Taste zu drücken.
5. Erste Ausgabe: Anzahl der Wertepaare der Meßreihe.
   Zweite Ausgabe: Regressionskoeffizient b.
   Dritte Ausgabe: Konstante a.
   Nach jeder Ausgabe ist die R/S-Taste zu drücken.
6. Die Regressionsfunktion lautet dann $y = a + b\,x$.
7. Das Schlußkennzeichen *skz* ist kleiner zu wählen als der kleinste x-Wert der Meßreihe.

Speicherbelegung, Programm und Programmdokumentation zur nichtlinearen Regression (Abschnitt 3.4.5.)

Speicherbelegungsplan:

Die Belegung der Speicher erfolgt wie beim Programm für die lineare Regression (s. S. 123).

Programm:

Rechner mit algebr. Logik		Rechner mit Umgek. Poln. Notation		Bemerkungen
Eingabe	Code	Eingabe	Code	
R/S	13–000	R/S	13–000	HALT für die Eingabe des Schlußkennzeichens *skz*
STO 0	64–001 91–002	STO 0	64–001 91–002	Speichern von *skz*
0	91–003	0	91–003	0
STO 1	64–004 81–005	STO 1	64–004 81–005	Speichern des Anfangwertes von i
STO 2	64–006 82–007	STO 2	64–006 82–007	Speichern des Anfangwertes von s_1
STO 3	64–008 83–009	STO 3	64–008 83–009	Speichern des Anfangwertes von s_2
STO 4	64–010 71–011	STO 4	64–010 71–011	Speichern des Anfangwertes von s_3
STO 5	64–012 72–013	STO 5	64–012 72–013	Speichern des Anfangwertes von s_4

R/S	13–014	R/S	13–014	HALT für Eingabe eines x-Wertes (bzw. des Schlußkennzeichens *skz* am Ende der Meßreihe) Beginn der Erfassung und Auswertung der Meßreihe
STO 6	64–015	STO 6	64–015	
	73–016		73–016	Speichern des x-Wertes bzw. des Schlußkennzeichens
–	85–017			x
RCL 0	65–018	RCL 0	65–017	
	91–019		91–018	*skz*
=	95–020	–	85–019	$x - skz$
F	24–021	ENTER	95–020	$x - skz$
x^2	34–022	×	74–021	$(x - skz)^2$
–	85–023			
0	91–024	0	91–022	0
X↔Y	54–025	X↔Y	54–023	
=	95–026	–	85–024	$h = 0 - (x - skz)^2$
X≧0	15–027	X≧0	15–025	Entscheidung, ob der zuletzt eingegebene Zahlenwert ein Meßwert ($h<0$) oder das Schlußkennzeichen *skz* ($h = 0$) ist. Im Fall $h<0$ wird der folgende Sprungbefehl übergangen.
GOTO	14–028	GOTO	14–026	
1	81–029	1	81–027	

0	91–030	0	91–028	
0	91–031	0	91–029	Sprung zur Endauswertung, deren Programmierung mit dem Befehl 100 beginnt.
R/S	13–032	R/S	13–030	HALT f.d. Eingabe des zugehörigen y-Wertes
STO 7	64–033	STO 7	64–031	
	61–034		61–032	Speichern des y-Wertes
ln	42–035	ln	42–033	ln y. In Abweichung vom Programmablaufplan wird hier als erstes die Anweisung $s_2 := s_2 + \ln y$ programmiert, da y sich bereits im X-Register befindet.
+	84–036			ln y
RCL 3	65–037	RCL 3	65–034	
	83–038		83–035	s_2
=	95–039	+	84–036	$s_2 + \ln y$
STO 3	64–040	STO 3	64–037	
	83–041		83–038	Speichern des neuen s_2
RCL 1	65–042	RCL 1	65–039	
	81–043		81–040	i
+	84–044			
1	81–045	1	81–041	1
=	95–046	+	84–042	$i + 1$
STO 1	64–047	STO 1	64–043	
	81–048		81–044	Speichern des neuen i

RCL 2	65–049	RCL 2	65–045	
	82–050		82–046	s_1
+	84–051			
RCL 6	65–052	RCL 6	65–047	
	73–053		73–048	x
=	95–054	+	84–049	$s_1 + x$
STO 2	64–055	STO 2	64–050	
	82–056		82–051	Speichern des neuen s_1
RCL 6	65–057	RCL 6	65–052	
	73–058		73–053	x
×	74–059			
RCL 7	65–060	RCL 7	65–054	
	61–061		61–055	y
ln	42–062	ln	42–056	$\ln y$
+	84–063	×	74–057	$x \cdot \ln y$
RCL 4	65–064	RCL 4	65–058	
	71–065		71–059	s_3
=	95–066	+	84–060	$s_3 + x \cdot \ln y$
STO 4	64–067	STO 4	64–061	
	71–068		71–062	Speichern des neuen s_3
RCL 6	65–069	RCL 6	65–063	
	73–070		73–064	x
F	24–071	ENTER	95–065	x
x^2	34–072	×	74–066	x^2
+	84–073			
RCL 5	65–074	RCL 5	65–067	
	72–075		72–068	s_4
=	95–076	+	84–069	$s_4 + x^2$
STO 5	64–077	STO 5	64–070	Speichern des neuen
	72–078		72–071	s_4

GOTO	14–079	GOTO	14–072	
0	91–080	0	91–073	
1	81–081	1	81–074	
4	71–082	4	71–075	Rücksprung zur Eingabe des nächsten x-Wertes bzw. des Schlußkennzeichens am Ende der Meßreihe
RCL 1	65–100 81–101	RCL 1	65–100 81–101	Beginn der Endauswertung der Meßreihen
R/S	13–102	R/S	13–102	HALT für die Ausgabe von i
×	74–103			i
RCL 4	65–104 71–105	RCL 4	65–103 71–104	s_3
−	85–106	×	74–105	$i \cdot s_3$
F	24–107			
(	61–108			
RCL 2	65–109 82–110	RCL 2	65–106 82–107	s_1
×	74–111			
RCL 3	65–112 83–113	RCL 3	65–108 83–109	s_2
F	24–114			
)	62–115	×	74–110	$s_1 \cdot s_2$
÷	75–116	−	85–111	$i \cdot s_3 - s_1 \cdot s_2$
F	24–117			
(	61–118			

RCL 1	65–119	RCL 1	65–112	
	81–120		81–113	i
×	74–121			
RCL 5	65–122	RCL 5	65–114	
	72–123		72–115	s_4
−	85–124	×	74–116	$i \cdot s_4$
RCL 2	65–125	RCL 2	65–117	
	82–126		82–118	s_1
F	24–127	ENTER	95–119	
x^2	34–128	×	74–120	${s_1}^2$
F	24–129			
)	62–130	−	85–121	$i \cdot s_4 - {s_1}^2$
=	95–131	÷	75–122	$b = (i \cdot s_3 - s_1 \cdot s_2)/(i \cdot s_4 - {s_1}^2)$
STO 8	64–132	STO 8	64–123	
	62–133		62–124	Speichern von b
×	74–134			
RCL 2	65–135	RCL 2	65–125	
	82–136		82–126	s_1
−	85–137	×	74–127	$b \cdot s_1$
RCL 3	65–138	RCL 3	65–128	
	83–139		83–129	s_2
X↔Y	54–140	X↔Y	54–130	
÷	75–141	−	85–131	$s_2 - b \cdot s_1$
RCL 1	65–142	RCL 1	65–132	i
	81–143		81–133	
=	95–144	÷	75–134	$(s_2 - b \cdot s_1)/i$
e^x	41–145	e^x	41–135	$a = e^{(s_2 - b \cdot s_1)/i}$
R/S	13–146	R/S	13–136	HALT für die Ausgabe von a

RCL 8	65–147	RCL 8	65–137	
	62–148		62–138	b
R/S	13–149	R/S	13–139	HALT für die Ausgabe von b
GOTO	14–150	GOTO	14–140	
0	91–151	0	91–141	
0	91–152	0	91–142	
0	91–153	0	91–143	Rücksprung zum Programmanfang

Programmdokumentation:

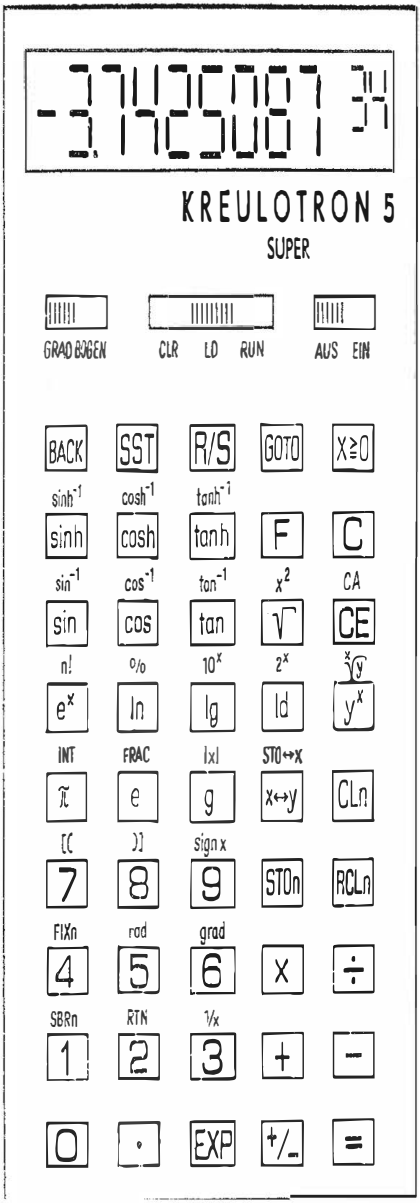
1. Gewünschte Rechengenauigkeit einstellen
2. Start des Programms. Startadresse: 000
3. Eingabe des Schlußkennzeichens *skz*. Das Schlußkennzeichen *skz* muß kleiner sein als der kleinste auftretende x-Wert der Meßreihe.
   R/S -Taste drücken.
4. Bei jedem nun folgenden Halt ist jeweils ein x-Wert und beim nachfolgenden Halt der zugehörige y-Wert einzugeben. Nach jeder Eingabe: R/S -Taste drücken.
5. Am Ende der Meßreihe ist beim folgenden Halt noch einmal das Schlußkennzeichen *skz* einzugeben und danach die R/S -Taste zu drücken.
6. Erste Ausgabe: Anzahl der Wertepaare.
   Zweite Ausgabe: Koeffizient a der Regressionsfunktion

   $$y = a \cdot e^{bx}.$$

   Dritte Ausgabe: Koeffizient b des Exponenten der Regressionsfunktion

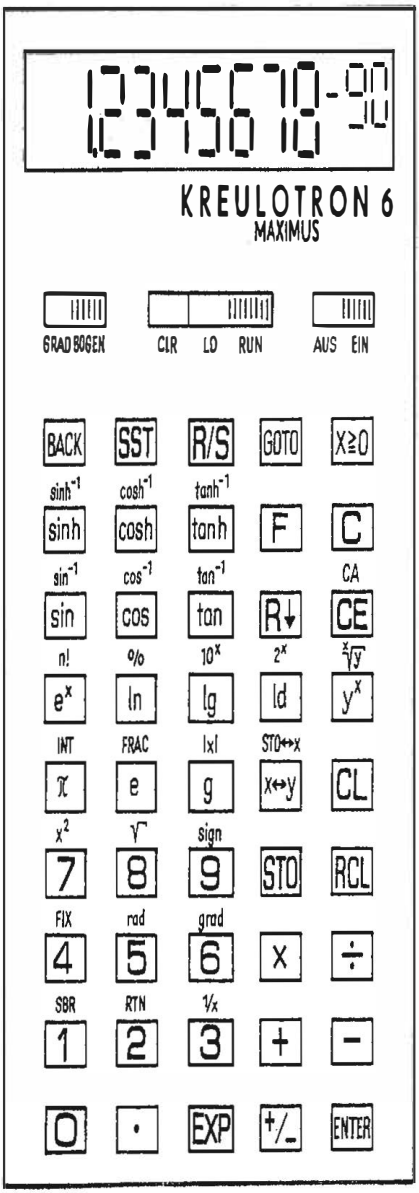
   $$y = a \cdot e^{bx}.$$

   Nach jeder Ausgabe muß die R/S -Taste gedrückt werden.



Codetabelle des Rechners Kreulotron 5

Taste	Code	Taste	Code	Taste	Code	Taste	Code	Taste	Code
BACK	–	SST	–	R/S	13	GOTO	14	X≧0	15
sinh	21	cosh	22	tanh	23	F	24	C	25
$\sinh^{-1}$	24 21	$\cosh^{-1}$	24 22	$\tanh^{-1}$	24 23				
sin	31	cos	32	tan	33	$\sqrt{}$	34	CE	35
$\sin^{-1}$	24 31	$\cos^{-1}$	24 32	$\tan^{-1}$	24 33	x^2	24 34	CA	24 35
e^x	41	ln	42	lg	43	ld	44	Y^x	45
n!	24 41	%	24 42	10^x	24 43	2^x	24 44	$\sqrt[x]{Y}$	24 45
π	51	e	52	g	53	Y↔X	54	CLn	55
INT	24 51	FRAC	24 52	|X|	24 53	STO↔X	24 54		
7	61	8	62	9	63	STOn	64	RCLn	65
(	24 61	)	24 62	sign x	24 63				
4	71	5	72	6	73	×	74	÷	75
FIX n	24 71	rad	24 72	grad	24 73				
1	81	2	82	3	83	+	84	−	85
SBR n	24 81	RTN	24 82	1/x	24 83				
0	91	.	92	EXP	93	+/−	94	=	95



Codetabelle für den Rechner Kreulotron 6

Taste	Code	Taste	Code	Taste	Code	Taste	Code	Taste	Code
BACK	–	SST	–	R/S	13	GOTO	14	X≧0	15
sinh	21	cosh	22	tanh	23	F	24	C	25
$\sinh^{-1}$	24 21	$\cosh^{-1}$	24 22	$\tanh^{-1}$	24 23				
sin	31	cos	32	tan	33	R↓	34	CE	35
$\sin^{-1}$	24 31	$\cos^{-1}$	24 32	$\tan^{-1}$	24 33			CA	24 35
e^x	41	ln	42	lg	43	ld	44	Y^x	45
n!	24 41	%	24 42	10^x	24 43	2^x	24 44	$\sqrt[x]{Y}$	24 45
π	51	e	52	g	53	X↔Y	54	CLn	55
INT	24 51	FRAC	24 52	|X|	24 53	X↔STO	24 54		
7	61	8	62	9	63	STOn	64	RCLn	65
X^2	24 61	$\sqrt{}$	24 62	sign	24 63				
4	71	5	72	6	73	×	74	÷	75
FIXn	24 71	rad	24 72	grad	24 73				
1	81	2	82	3	83	+	84	−	85
SBRn	24 81	RTN	24 82	1/x	24 83				
0	91	.	92	EXP	93	+/−	94	ENTER	95

Literatur- und Quellenverzeichnis

[1] Algebra und Geometrie für Ingenieure. 9. Aufl. Leipzig: VEB Fachbuchverlag 1974

[2] Ausgewählte Kapitel der Mathematik. 8. Aufl. Leipzig: VEB Fachbuchverlag 1974

[3] Mathematik für Ingenieur- und Fachschulen, Band I, 4. Aufl. Leipzig: VEB Fachbuchverlag 1979

[4] alpha. Mathematische Schülerzeitschrift 12. Jahrgang 1978, Heft 5. Berlin: Volk und Wissen Volkseigener Verlag

[5] Gilde, W., und S. Altrichter: Mehr Spaß mit dem Taschenrechner. Leipzig: VEB Fachbuchverlag 1978

[6] Kreul, H.: Was kann mein elektronischer Taschenrechner? 3. Aufl. Leipzig: VEB Fachbuchverlag 1980

[7] Schumny, H.: Taschenrechner Handbuch. Leipzig: BSB B. G. Teubner Verlagsgesellschaft 1979

Sachwortverzeichnis

Verzeichnis der verwendeten Sondertasten

DDR 9,80 M

PROGRAMMIEREN IST ERLERNBAR!

In diesem Buch wird gezeigt, wie man

- von der Aufgabenstellung
  über
- die Auswahl eines geeigneten Lösungsalgorithmus
  zum
- Programmablaufplan
  und damit zum
- Rechenprogramm

gelangen kann.

Das Buch wurde geschrieben für

- interessierte Oberschüler,
- Studenten,
- Techniker, Ingenieure und Ökonomen,
- Lehrer und Wissenschaftler,

die einen programmierbaren Taschenrechner effektiv nutzen wollen.

Es gibt darüber hinaus Hinweise, die auch für die Aufstellung von Rechenprogrammen für beliebige Rechenanlagen wertvoll sind.