Einführung in PL/1 für Wirtschaftswissenschaftler

Von
Dr. Klaus Werner Wirtz
Professor für Betriebswirtschaftslehre
insbesondere Datenverarbeitung

3., neubearbeitete Auflage

R. Oldenbourg Verlag München Wien

CIP-Titelaufnahme der Deutschen Bibliothek

Wirtz, Klaus Werner:
Einführung in PL 1 für Wirtschaftswissenschaftler / von Klaus Werner Wirtz. – 3., neubearb. Aufl. – München ; Wien : Oldenbourg, 1989
ISBN 3-486-21214-1

Gesamtherstellung: Rieder, Schrobenhausen

ISBN 3-486-21214-1

Inhaltsverzeichnis

Vorwort (zur ersten bis dritten Auflage)

Das vorliegende Buch ist aus Lehrveranstaltungen entstanden, die der Autor von 1977 bis 1986 an der Universität Essen - Gesamthochschule durchgeführt hat. Es setzt keine Programmierkenntnisse beim Leser voraus; nützlich sind jedoch Grundkenntnisse über automatisierte Datenverarbeitung, wie sie etwa in Veranstaltungen zur Einführung in die Datenverarbeitung oder aus entsprechenden Lehrbüchern (einige sind im Literaturverzeichnis genannt) erworben werden können.

Die Programmiersprache PL/1 existiert seit 1965. Sie wurde mit dem Ziel konzipiert, für möglichst viele Aufgabenstellungen einsetzbar zu sein; daher weist sie einen sehr großen Sprachumfang auf, der im Rahmen eines Einführungstexts nicht komplett behandelt werden kann. In diesem Buch wird eine auf kommerzielle Probleme zugeschnittene Untermenge der Sprache vermittelt.

Der Aufbau des Buchs ist so angelegt, daß es zum Selbststudium geeignet ist. Die einzelnen Kapitel beginnen durchweg mit einer Formulierung des Lernziels; auf den darstellenden Teil folgen Übungsaufgaben, in denen die behandelten Sprachelemente durch Anwendung vertieft werden. Zahlreiche Programmbeispiele im Text erleichtern das Verständnis des Inhalts.

Programmieren, gleich in welcher Programmiersprache, ist im wesentlichen eine praktische Tätigkeit und kann nicht durch Literaturstudium allein erlernt werden. Die Bearbeitung von Übungsaufgaben ist daher sehr wichtig für den Lernerfolg. Ein in PL/1 codierter Algorithmus kann in diesem Zusammenhang noch nicht als Lösung gelten; erst der Ablauf eines Programms auf einer Datenverarbeitungsanlage zeigt, ob das Programm das tut, was von ihm erwartet wird. Bei diesem "Testen" von Programmen wird von Fall zu Fall neben diesem Buch das PL/1-Handbuch des Übersetzerherstellers heranzuziehen sein, um Detailfragen zu klären. Vor allem ist im Text gelegentlich von "implementierungsabhängigen" Eigenschaften der Sprache PL/1 die Rede; das sind Dinge, die jeder Hersteller selbst festlegen kann.

Die in diesem Buch abgedruckten Programme wurden für die erste Auflage auf den Datenverarbeitungsanlagen PRIME 400/750 des Hochschulrechenzentrums Essen und TR445 des Hochschulrechenzentrums Düsseldorf gerechnet. Für die dritte Auflage wurden eini-

ge Programme auf den Siemens-Großrechner des Hochschulrechenzentrums Düsseldorf übertragen. Ein anderer Teil der Programme wurde außerdem mit dem Digital Research PL/1-Übersetzer auf einem Personal Computer implementiert. Dieser PL/1-Übersetzer macht eine Untermenge der PL/1-Sprache auch für PC-Benutzer verfügbar; er enthält allerdings nicht alle Sprachelemente, die in diesem Buch behandelt werden.

Während in der zweiten Auflage des Buchs verschiedene Errata der ersten Auflage vermerkt wurden, wurde für die dritte Auflage der gesamte Text inhaltlich überarbeitet und in eine lesefreundlichere Form gebracht.

Abschließend danke ich den Personen, ohne deren Hilfe und Unterstützung dieses Buch nicht oder nicht in dieser Form entstanden wäre: Herrn Prof. Dr. D. Seibt, in dessen Fachgebiet Betriebsinformatik/Operations Research im Fachbereich Wirtschaftswissenschaften der Universität Essen - Gesamthochschule ich von 1977 bis 1980 als Wissenschaftlicher Mitarbeiter beschäftigt war; Herrn Prof. Dipl.-Inf. R. Senger (Fachhochschule Karlsruhe), der während seiner Tätigkeit als Lehrbeauftragter in Essen den Aufbau der diesem Buch zugrundeliegenden Lehrveranstaltungen mitgeprägt hat; Herrn Dipl.-Kfm. T. Bauer (Universität Hohenheim), dem ich zahlreiche Anregungen verdanke; Frau K. Eggert, Frau B. Emmerich, Frau I. Siebert und Frau E. Vorholt, die verschiedene Entwicklungsstufen des Manuskripts der ersten Auflage geschrieben haben; und nicht zuletzt Herrn Dipl.-Volksw. M. Weigert vom Oldenbourg Verlag, der die Entwicklung und Herstellung des Buchs unterstützt hat.

Klaus Werner Wirtz

1. Einführung

Lernziel:
Ziel dieses einführenden Kapitels ist, den Prozeß der Entwicklung von Programmen kurz darzustellen. Es zeigt sich, daß der Teil dieses Prozesses, in dem die Programmiersprache PL/1 (oder eine andere Sprache) zur Anwendung gelangt, nur einer von mehreren abgrenzbaren Arbeitsvorgängen ist, die teils nacheinander, teils parallel ablaufen. Deshalb wird diesen anderen Entwicklungsphasen, die in den folgenden Kapiteln nicht mehr behandelt werden, hier besondere Aufmerksamkeit gewidmet.

1.1. Der Prozeß der Entwicklung eines Programms

Unter einem Programm für eine Datenverarbeitungsanlage versteht man eine Arbeitsvorschrift, mit der ein Problem oder eine Klasse von irgendwie ähnlichen Problemen gelöst wird.

Der Prozeß der Programmentwicklung beginnt also mit einer Problemerkennung. Ein Problem kann z.B. sein, die Lösungen der quadratischen Gleichung

$$4x^2 + 8x + 3 = 0$$

zu finden; eine Klasse von ähnlichen Problemen stellt die Menge der quadratischen Gleichungen

$$ax^2 + bx + c = 0$$

dar, die alle nach einem einheitlichen Schema gelöst werden können.

Ein anderes Problem ist, aus früheren Umsatzwerten eines Betriebs zukünftig erwartete Umsatzwerte zu prognostizieren. Dieses Problem ist offenbar wesentlich ungenauer formuliert als das erste. Unklar ist vor allem,

- welcher Art die vorliegenden Vergangenheitswerte sind (z.B. Monats-, Jahreswerte),
- wieviele Werte vorliegen,
- wieviele Werte vorausgesagt werden sollen, und
- welches statistische Verfahren anzuwenden ist.

Solange diese Fragen nicht beantwortet sind, ist an eine Bearbeitung des Problems - ob manuell oder mit Hilfe einer Datenverarbeitungsanlage - nicht zu denken. An die Problemerkennung schließt sich also eine Problemanalyse an, die zum Ziel hat, möglichst alle zur

Bearbeitung des Problems erforderlichen Informationen zusammenzustellen und zu entscheiden, ob das Problem grundsätzlich lösbar ist.

Auf die Problemanalyse folgt - manchmal ohne erkennbaren Übergang - die Suche nach einem Lösungsweg. Ein solches Verfahren zur Lösung kann sehr einfach zu finden sein; für das Problem der quadratischen Gleichungen z.B. entdeckt man in einer Formelsammlung folgendes Verfahren:

1) a muß ungleich Null sein (sonst liegt keine quadratische Gleichung vor).
2) $b^2 - 4ac$ muß größer oder gleich Null sein (sonst gibt es komplexe Lösungen, die hier nicht interessieren).
3) Liegen diese Voraussetzungen vor, dann ergeben sich die Lösungen x_1 und x_2 nach der Formel

$$x_{1/2} = -\frac{b}{2a} \pm \frac{1}{2a} * \text{Wurzel aus } (b^2 - 4ac)$$

Damit ist der Lösungsweg bereits beschrieben. Ein solches Verfahren bezeichnet man als Algorithmus. An einen Algorithmus werden folgende Anforderungen gestellt:

- Er muß einen Anfang und mindestens ein Ende haben. Unser Algorithmus hat drei mögliche Enden: nach Schritt 1 und 2 im Fehlerfall, nach Schritt 3 im "Normalfall".
- Er muß nach einer endlichen Zahl von Schritten zu einem Ende kommen. Oben sind es entweder ein, zwei oder drei Schritte.
- Er muß ein oder mehrere Ergebnisse haben. Ergebnisse des obigen Algorithmus sind: "Es liegt keine quadratische Gleichung vor" oder "Die Lösungen sind komplex" oder "Die Lösungen sind reell und betragen x_1 und x_2".
- Er muß - wenn nötig - Eingabewerte zulassen. Unser Algorithmus braucht im konkreten Fall Zahlenwerte für a, b und c.
- Er muß eindeutig sein, d.h. mehrmaliges Ausführen des Algorithmus mit den gleichen Eingabewerten muß stets gleiche Ergebnisse liefern. Dies ist bei dem obigen Algorithmus gewährleistet.

Nicht immer kann der komplette Lösungsweg aus einer Formelsammlung einfach abgeschrieben werden. Sehr oft stellt das Finden eines geeigneten Algorithmus einige Anforderungen an den Einfallsreichtum und die Kreativität des Bearbeiters. Große und komplizierte Algorithmen werden der Übersichtlichkeit wegen oft in

Form von grafischen Darstellungen anschaulich gemacht. In diesem Buch werden dazu die sogenannten Struktogramme verwendet, deren Aufbau und Gebrauch im nächsten Abschnitt beschrieben wird.

Soll ein Algorithmus automatisch in einer Datenverarbeitungsanlage ablaufen, muß er in einer dieser Anlage verständlichen Sprache formuliert werden. Den Vorgang der Umsetzung eines normalsprachlich oder in einer Grafik formulierten Algorithmus nennt man Codierung; eine für Datenverarbeitungsanlagen geeignete Sprache heißt Programmiersprache. Es gibt zahlreiche Programmiersprachen; eine davon ist die Sprache PL/1, die Gegenstand dieses Buches ist. Der Vorgang der Codierung ist - im Gegensatz zu dem eher kreativen Prozeß der Lösungsfindung - ein mehr formaler Prozeß, da schwerpunktmäßig auf die Regeln der Programmiersprache zu achten ist.

Nach Beendigung der Codierung liegt ein Programm vor. Der Prozeß der Programmentwicklung ist damit jedoch noch nicht abgeschlossen, da sich an verschiedenen Stellen Fehler eingeschlichen haben können:

- Im Programm können Verstöße gegen die Syntax der Programmiersprache enthalten sein.
- Im Verlauf der Problemanalyse, der Lösungsfindung oder der Codierung können Irrtümer aufgetreten sein, die zur Folge haben, daß das Programm nicht die nach der Problemstellung zu erwartenden Ergebnisse liefert. Ein Programm, das u.a. eine Mehrwertsteuer von 14 % errechnen soll und zu einem Betrag von 100 als Mehrwertsteuer den Wert 1,40 ausgibt, mag syntaktisch richtig sein, inhaltlich ist es jedoch falsch.

Solche Fehler müssen gefunden und behoben werden, bevor ein Programm als fertig gelten kann. Verstöße gegen die Syntax der Programmiersprache werden automatisch vom Übersetzer erkannt. Der Übersetzer ist ein - üblicherweise vom Hersteller der Datenverarbeitungsanlage zur Verfügung gestelltes - Programm, das die Anweisungen des Programms auf syntaktische Richtigkeit prüft und in Befehle der Maschinensprache überträgt. Inhaltliche Fehler können nicht automatisch erkannt werden, da der Übersetzer keine Kenntnis von dem zu lösenden Problem hat; er kann nur formale Prüfungen vornehmen. Nach inhaltlichen Fehlern sucht man, indem man das Programm mit Eingabewerten ablaufen läßt, für die die Ergebnisse bekannt sind. Abweichungen zwischen diesen Ergebnissen und den Ausgabewerten des Programms deuten auf inhaltliche Fehler hin.

Den Vorgang der Fehlersuche und -behebung nennt man Test; die Daten, die zum Ausprobieren eines Programms benutzt werden, heißen Testdaten. Testen ist ein Prozeß, der üblicherweise mehrmals durchlaufen wird: Findet man einen Fehler, sucht man seine Ursache und behebt sie; danach wird das Programm erneut ausprobiert. Zeigt sich wieder ein Fehler, wird dessen Ursache gesucht und behoben; ein neuer Testlauf schließt sich an, usw. Die Problematik des Programmtestens besteht darin, daß man stets nur eine kleine Menge der insgesamt möglichen Eingabedaten im Test ausprobieren kann; die Richtigkeit eines Programms kann also nicht endgültig bewiesen werden. In der Praxis kommt es denn auch häufig vor, daß nach Jahren der Benutzung eines Programms immer noch Fehler gefunden werden, die durch vorher nie aufgetretene, aber zulässige Konstellationen von Eingabedaten ans Tageslicht kommen.

Zwei Empfehlungen für die Suche nach inhaltlichen Fehlern lauten:

1) Möglichst viele verschiedene Kombinationen von Eingabedaten, die auch Sonderfälle und Grenzwerte des Algorithmus abdecken, ausprobieren. Für den Algorithmus zur Lösung der quadratischen Gleichung bedeutet das, auch Eingabedaten, die keine quadratische Gleichung beschreiben, und solche, die zu komplexen Lösungen führen, vorzusehen.
2) Mit der inneren Einstellung, Fehler finden zu wollen, an den Test herangehen. Dies klingt trivial; hat man ein Programm selbst entwickelt, ist man jedoch oft von seiner Richtigkeit völlig überzeugt und hat deshalb Schwierigkeiten, Fehler als Fehler zu erkennen. (Aus diesem Grund überträgt man in der Praxis häufig das Testen eines Programms nicht dem Ersteller, sondern einer anderen Person, da das Aufdecken von Fehlern, die ein anderer begangen hat, leichter fällt!)

Nach Abschluß der Testphase ist das Programm fertig. Eine sehr wichtige Aktivität wurde jedoch bisher noch nicht erwähnt, da sie parallel zu allen anderen Tätigkeiten ausgeführt wird: die Dokumentation. Ziel der Dokumentation ist, alle Unterlagen bereitzustellen, die zur Nutzung des Programms, zum Verständnis der Programmergebnisse und für spätere Programmänderungen notwendig sind. Dazu reicht der Text des Programms oft nicht aus. Bestandteile einer Dokumentation können sein:

- Die Formulierung des Problems, das das Programm löst,
- Die Darstellung des Lösungswegs,
- Der kommentierte Programmtext,

- Testdaten und Ergebnisse von Programmläufen.

Die ersten beiden und der letzte Punkt sind wohl einsichtig; wie ein Programmtext verständlich aufgebaut und kommentiert wird, wird in den folgenden Kapiteln anhand von Beispielen erläutert. Die Erfahrung zeigt, daß sogar der Autor eines Programms Schwierigkeiten hat, sich nach einiger Zeit zu erinnern, was das Programm tut und wie es das tut, wenn ihm nur der nackte Programmtext vorliegt.

Zusammenfassung:
In Abbildung 1.1. ist der Prozeß der Entwicklung eines Programms noch einmal dargestellt.

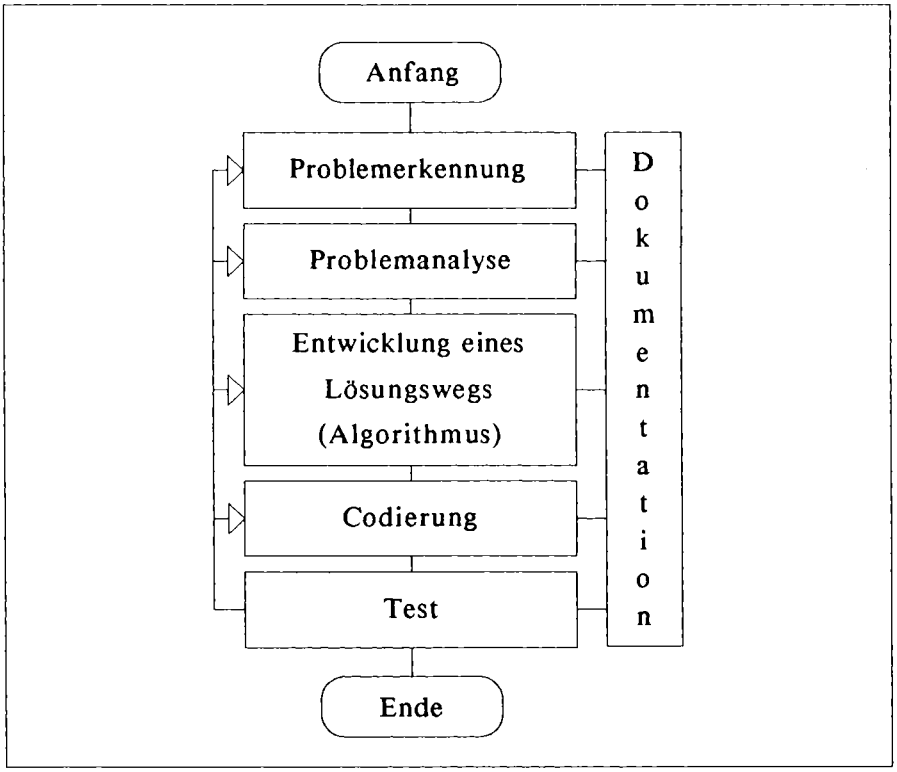

Abb. 1.1. Der Prozeß der Programmentwicklung

Die Phasen
- Problemerkennung,

- Problemanalyse,
- Entwicklung eines Algorithmus,
- Codierung, und
- Test

laufen also nacheinander ab. Alle Phasen tragen etwas zur Dokumentation bei. Fehler, die im Test auftreten, werden auf ihren Ursprung hin untersucht: falsches Problemverständnis, Irrtümer bei der Entwicklung des Algorithmus, Ungenauigkeiten bei der Codierung können Fehlerursachen sein. In Abhängigkeit von der Phase, in der der Fehler wurzelt, müssen eventuell mehrere Arbeitsvorgänge wiederholt werden.

Dieser Ablauf ist zugeschnitten auf die Zwecke des vorliegenden Buchs: Ein einzelner Leser soll lernen, relativ kleine, überschaubare Programmierprobleme zu bearbeiten. Die daraus entstehenden Programme werden im allgemeinen nicht unmittelbar praktisch einsetzbar sein; sie haben bloßen Übungscharakter. In der betrieblichen Praxis laufen große Programmentwicklungsprojekte in vieler Hinsicht anders ab; Unterschiede ergeben sich vor allem aus:

- Der Trennung zwischen Entwicklern und Anwendern eines Programms. Programme werden meist in der Datenverarbeitungsabteilung entwickelt; angeregt und angewendet werden sie meist von einer Fachabteilung. Die Phasen der Problemerkennung und der Problemanalyse werden daher oft von Fachspezialisten aus der Fachabteilung und von Datenverarbeitungsspezialisten gemeinsam durchgeführt. Damit die Fachabteilung das entwickelte Programm selbständig nutzen kann, sind auch an die Dokumentation besondere Anforderungen zu stellen.
- Der Größe eines betrieblichen Projekts. Reale Projekte sind oft so umfangreich, daß ein Einzelner mit ihrer Bearbeitung überfordert wäre. Folglich werden solche Projekte von Arbeitsgruppen abgewickelt, wobei sich Probleme der Arbeitsteilung und der Projektorganisation und -lenkung ergeben.
- Der Notwendigkeit, parallel zur Entwicklung der Programme organisatorische Konsequenzen der Einführung der Programme zu erkennen und diese Umstellungen vorzubereiten. Hier sind Fragen der Änderung von Arbeitsabläufen, der Neufestlegung von Arbeitsinhalten und der Schulung von späteren Benutzern der Programme angesprochen.
- Der "Lebensdauer" von Programmen. Einmal entwickelte Programme haben eine durchschnittliche Einsatzdauer von sieben bis zehn Jahren. In diesem Zeitraum ändern sich häufig die Anforderungen an die Programme, so daß die Änderung und Weiter-

entwicklung bestehender Programme zu einem wesentlichen Aufwandsfaktor werden.

Abschließend sei noch einmal betont, daß der Schwerpunkt des vorliegenden Buchs eindeutig in der Codierphase liegt. Fragen der Problemanalyse und des Testens von Programmen werden mehr am Rande im Rahmen der vorgestellten Programmbeispiele behandelt; eine gewisse Fertigkeit in diesen Arbeiten erlangt der Leser, wenn er möglichst viele der jedem Kapitel angefügten Übungsaufgaben programmiert und auf einer Datenverarbeitungsanlage testet.

1.2. Darstellung von Algorithmen durch Struktogramme

Ein Algorithmus, wie der im vorigen Abschnitt zur Lösung von quadratischen Gleichungen angeführte, besteht aus einzelnen Schritten, die in einer bestimmten Reihenfolge hintereinander ausgeführt werden. Um diese Reihenfolge deutlich zu machen, wird dieser Beispielalgorithmus noch einmal in leicht veränderter Form dargestellt:

1) Nimm Werte für a, b und c.
2) Wenn a ungleich Null ist, fahre bei 3) fort; sonst beende das Verfahren, weil keine quadratische Gleichung vorliegt.
3) Berechne $b^2 - 4ac$.
4) Wenn $b^2 - 4ac$ größer oder gleich Null ist, fahre bei 5) fort; sonst beende das Verfahren, weil komplexe Lösungen auftreten.
5) Berechne

$$x_1 = -\frac{b}{2a} + \frac{1}{2a} * \text{Wurzel aus } (b^2 - 4ac)$$

6) Berechne

$$x_2 = -\frac{b}{2a} - \frac{1}{2a} * \text{Wurzel aus } (b^2 - 4ac)$$

7) Gib x_1 und x_2 aus.

Durch diesen Algorithmus werden drei verschiedene Reihenfolgen (Wege) beschrieben, von denen im konkreten Fall jeweils nur eine zur Ausführung kommt:

- Ist a gleich Null, werden die Schritte 1 und 2 ausgeführt.
- Ist a ungleich Null und $b^2 - 4ac$ kleiner Null, werden die Schritte 1 bis 4 ausgeführt.
- Ist a ungleich Null und $b^2 - 4ac$ größer oder gleich Null, werden die Schritte 1 bis 6 ausgeführt.

Die Schritte 2 und 4 enthalten sogenannte Entscheidungen, die die weitere Reihenfolge steuern. Die anderen Schritte laufen ohne weiteres hintereinander ab, genauer:
- Auf Schritt 1 folgt auf jeden Fall Schritt 2.
- Auf Schritt 3 folgt auf jeden Fall Schritt 4.
- Auf Schritt 5 folgt unbedingt Schritt 6.

Solche Teile eines Algorithmus, die linear hintereinander ausgeführt werden, nennt man Sequenzen.

Ein einzelner Arbeitsschritt, der keine Entscheidung enthält, wird grafisch in einem Rechteck abgebildet:

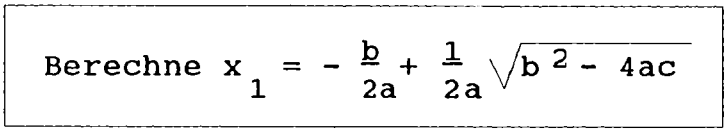

Abb. 1.2. Anweisungssymbol

Eine Sequenz von Schritten wird durch mehrere untereinander stehende Rechtecke gleicher Breite dargestellt:

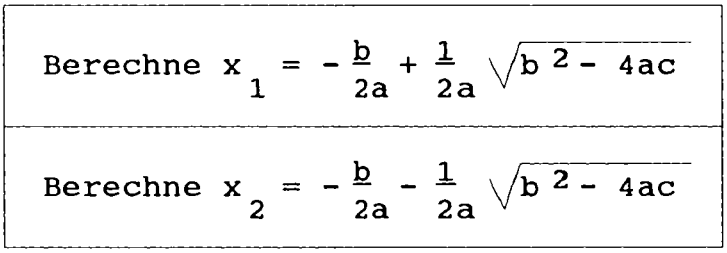

Abb. 1.3. Sequenz

Ein Arbeitsschritt, der eine Entscheidung enthält, wird in folgendem Symbol abgebildet:

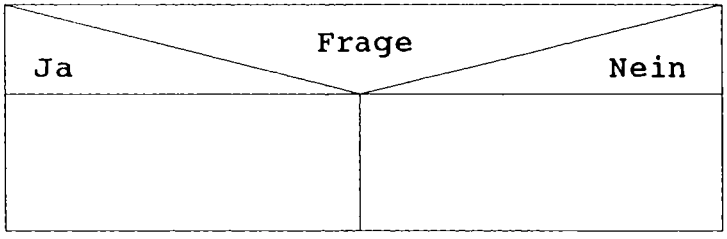

Abb. 1.4. Entscheidungssymbol

Das auf der Spitze stehende Dreieck enthält die Frage, die den weiteren Ablauf steuert. Diese Frage lautet bei Schritt 2: "Ist a ungleich Null?". Bei Schritt 4 heißt sie: "Ist $b^2 - 4ac$ größer oder gleich Null?".

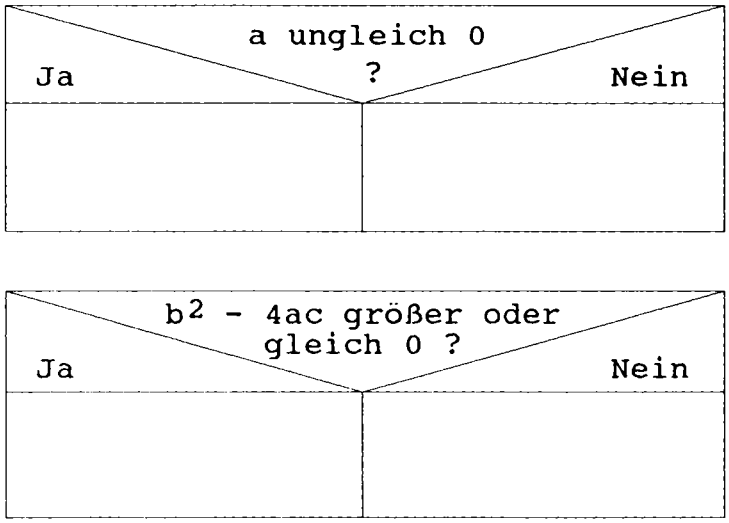

Abb. 1.5. Beispiele zum Entscheidungssymbol

In die unterhalb der Worte "Ja" und "Nein" stehenden Rechtecke werden dann die Schritte eingetragen, die ausgeführt werden, wenn die Frage bejaht bzw. verneint wird. Innerhalb eines solchen Ja- oder Nein-Zweigs kann also eine Sequenz von Schritten, aber auch eine weitere Entscheidung stehen.

Mehrere Symbole können zu größeren Einheiten zusammengesetzt werden. Alle sechs Schritte des obigen Algorithmus sind in Abbildung 1.6. zu einer Gesamtdarstellung, einem sogenannten Struktogramm, zusammengefaßt.

Die drei alternativen Wege, die in dem Algorithmus stecken, werden in der Abbildung gut sichtbar durch die drei Ausgänge, die das Struktogramm nach unten hat.

Festzuhalten ist:
- Weder die absolute Größe der einzelnen Symbole, noch das Verhältnis der Seitenlängen zueinander liegen fest.
- Die Symbole können ineinander verschachtelt werden.

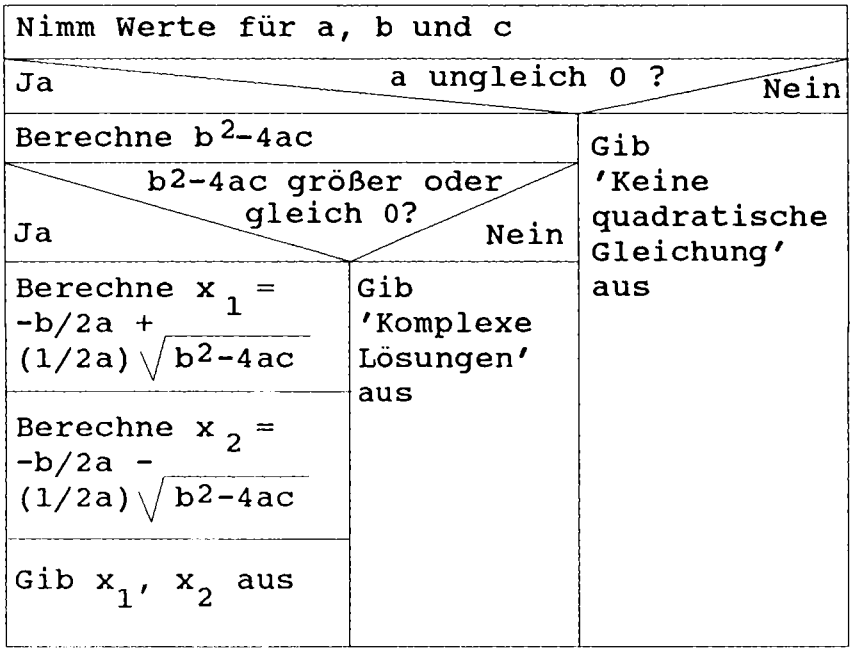

Abb. 1.6. Struktogramm "Lösung einer quadratischen Gleichung"

Ein drittes Konstruktionselement für Struktogramme wird benötigt, wenn der Algorithmus für quadratische Gleichungen so erweitert wird, daß er für beliebig viele Wertetripel (a,b,c) die Lösungen der zugehörigen quadratischen Gleichungen berechnet. Dies geschieht, indem der oben formulierte Algorithmus um einen Anfangsschritt 0 ergänzt wird, der lautet:

0) Wiederhole die Schritte 1 bis 7, solange Werte für a, b und c vorliegen.

Das dritte Konstruktionselement bezeichnet also Wiederholungen. Die Anzahl der Wiederholungen wird dadurch begrenzt, daß jede Wiederholung der Schritte 1 bis 7 zur Bedingung hat, daß noch Werte für a, b und c vorliegen. Sobald das Ende der Eingabewerte erreicht ist, bricht die Wiederholung ab.

Das Symbol für eine Wiederholung hat folgendes Aussehen:

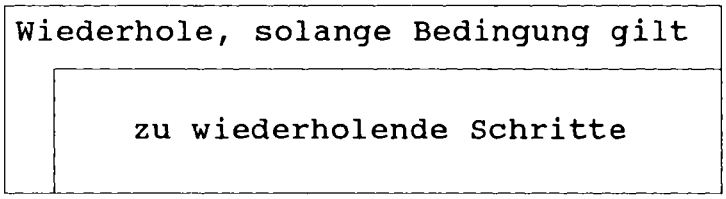

Abb. 1.7. Wiederholungssymbol

Der wiederholt ablaufende Algorithmus für quadratische Gleichungen stellt sich also so dar:

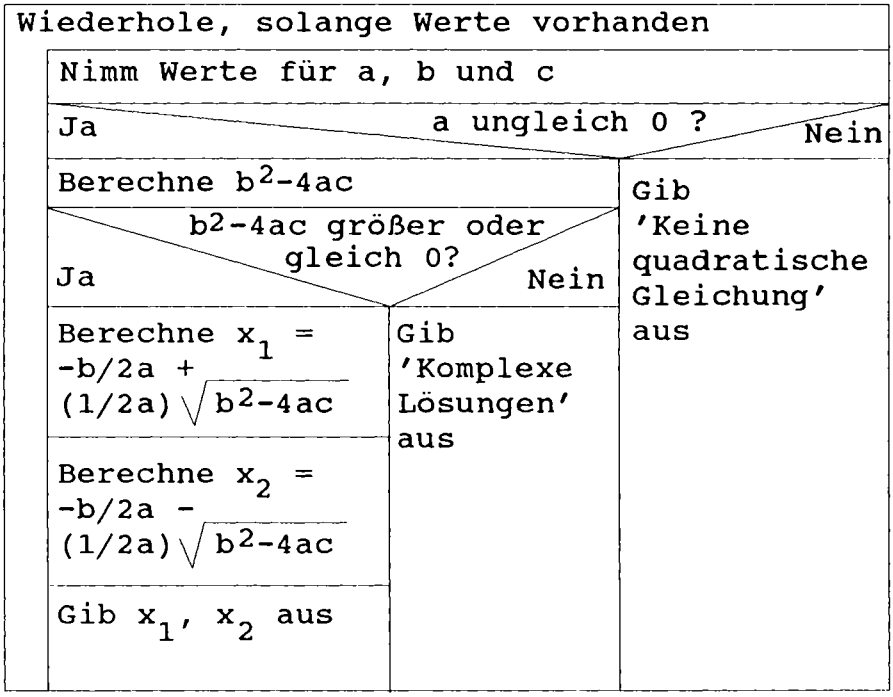

Abb. 1.8. Struktogramm "Lösung von quadratischen Gleichungen"

Über diese drei grafischen Konstrukte hinaus werden keine weiteren Darstellungselemente benötigt. Diese Darstellungsform wurde 1973 von I. Nassi und B. Shneiderman vorgestellt (Nassi/Shneiderman (1973)); Struktogramme werden daher auch als Nassi-Shneiderman-Diagramme bezeichnet.