Pascal für Wirtschafts-wissenschaftler

Einführung in die strukturierte Programmierung

Von
Dr. Judith Gebauer
und
Prof. Dr. Marcus Vögtle

3., überarbeitete Auflage

R. Oldenbourg Verlag München Wien

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.ddb.de> abrufbar.

Rosenheimer Straße 145, D-81671 München
Telefon: (089) 45051-0
oldenbourg.de

Gedruckt auf säure- und chlorfreiem Papier
Gesamtherstellung: Druckhaus „Thomas Müntzer" GmbH, Bad Langensalza

ISBN 3-486-57577-5
ISBN 978-3-486-57577-4

Geleitwort zur 1. Auflage

Das vorliegende Buch entstand aus den Erfahrungen mit der Durchführung eines Programmierpraktikums zu Pascal an der Universität Freiburg. Das Praktikum wird nunmehr seit sechs Jahren regelmäßig im Grundstudium des Diplomstudiengangs Volkswirtschaftslehre angeboten.

Die Frage, ob das Erlernen einer allgemeinen höheren Programmiersprache wie Pascal im wirtschaftswissenschaftlichen Studium Sinn mache, ist bei den Fachvertretern und Studierenden gleichermaßen umstritten. Gibt es doch für die meisten methodischen Probleme in der Zwischenzeit geeignete, benutzerfreundliche Standardsoftware. Auch im späteren Berufsleben wird der Studierende der Wirtschaftswissenschaften kaum noch selbständige Programmieraufgaben zu lösen haben.

Die Beherrschung von Pascal ist für uns deshalb nicht das eigentliche Ziel des Kurses, sondern nur Mittel zum Zweck. Hauptsächlich wollen wir den Studierenden die Möglichkeit eröffnen, wirtschaftswissenschaftliche Methoden und Konzepte am Rechner nachzuvollziehen und deren Wirkungsweise experimentell zu erforschen. Das Potential des Computers zur konzeptionellen Durchdringung einer Disziplin wird in den Wirtschaftswissenschaften im Gegensatz zu den Naturwissenschaften und zur Mathematik noch zu wenig gewürdigt. Dies kann aber auch kein Programmpaket leisten, bei dem der methodische Lösungsweg in einer „black box" verborgen bleibt. Die black box-Sicht ist für die praktische Anwendung von immenser Bedeutung, wenn man einmal den Mechanismus der Methoden verstanden hat, nicht aber für die Methodenlehre. Aus diesem Grund ist die Vermittlung einer allgemeinen Programmiersprache wichtig und sollte auch stets fachbezogen erfolgen.

Ich bin meinen beiden Mitarbeitern und langjährigen Betreuern des Pascal-Praktikums, Frau Diplom-Volkswirtin Judith Gebauer und Herrn Diplom-Volkswirt Marcus Vögtle, dankbar, daß sie aus den bisherigen „fliegenden Blättern" zum Praktikum ein „strukturiertes" Einführungsbuch haben entstehen lassen. Ebenso dankbar bin ich dem R. Oldenbourg Verlag und insbesondere Herrn Diplom-Volkswirt Martin M. Weigert für den spontanen Entschluß, das Buch im Verlagsprogramm zu den Wirtschafts- und Sozialwissenschaften aufzunehmen. Ich bin sicher, daß es die Lehre in Freiburg bereichern wird, und erhoffe mir das auch andernorts.

Freiburg, im Februar 1995 Professor Dr. Franz Schober

Danksagung

Wir bedanken uns ganz besonders bei **Professor Dr. Franz Schober** für sein Engagement in allen Phasen des Projekts. Sowohl die organisatorische Unterstützung als auch die wertvollen inhaltlichen Anregungen haben uns sehr geholfen.
Herzlichen Dank auch an **Anton Behringer** für seine Unterstützung bei der Vorbereitung des Manuskripts für die 3. Auflage.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Ziele dieses Buches

Das vorliegende Buch beruht auf dem Konzept einer Veranstaltung „Grundlagen der Programmierung in Pascal", die wir seit mehreren Jahren als wissenschaftliche Mitarbeiter am Lehrstuhl für Wirtschaftsinformatik (Prof. Dr. Franz Schober) für Studierende der Volkswirtschaftslehre an der Universität Freiburg anbieten.

Zur Programmiersprache Pascal gibt es bereits eine Vielzahl von Büchern. Diese sind jedoch zum Teil eher für den mathematisch-naturwissenschaftlich orientierten Leser geschrieben. Zudem ist ihr Schwierigkeitsgrad für den Anfänger oft zu hoch. Daher fiel es uns in der Vergangenheit schwer, den Studierenden der Wirtschaftswissenschaften ein geeignetes Lehrbuch zu empfehlen. Deshalb haben wir uns entschlossen, auf der Basis der von uns für die Studierenden zusammengestellten Unterlagen ein Lehrbuch zu verfassen, das mit adäquatem Schwierigkeitsgrad und wirtschaftswissenschaftlichen Beispielen und Aufgaben in das Thema ein- und auch weiterführt.

Mit diesem Buch möchten wir nicht zuletzt die Motivation der Studierenden fördern, eine Programmiersprache zu erlernen, da uns dazu immer wieder bohrende Fragen der folgenden Art gestellt werden:

- Wozu braucht ein Wirtschaftswissenschaftler Programmierkenntnisse?
- Gibt es nicht schon für jedes Problem Softwarelösungen zu kaufen, wozu also noch selbst programmieren?
- Warum werden statt der Programmierung nicht Einführungskurse in allgemein verbreitete Anwendungssoftwarepakete angeboten?
- Ist Pascal nicht eine veraltete Programmiersprache?

Im folgenden möchten wir Antworten auf diese Fragen geben und gleichzeitig die Ziele erläutern, die wir mit diesem Buch verfolgen.

1. **Ziel:** Anleitung zum Umgang mit der Computertechnologie und Vermittlung grundlegender Kenntnisse über die Funktionsweise eines PC.

   Nahezu jeder Wirtschaftswissenschaftler wird heute in seiner beruflichen Tätigkeit von der Computertechnologie unterstützt. Dies fängt beim Schreiben von Briefen mit Hilfe eines Textverarbeitungsprogramms auf dem PC an und hört beim Erstellen von Spreadsheets mit entsprechenden Kalkulationsprogrammen noch lange nicht auf. Electronic Mail, Electronic Commerce, Customer Relationship Management und Supply Chain Management sind Stichworte, die in Zukunft weiterhin an Bedeutung gewinnen werden.

   Dieses Buch soll ein Grundverständnis für die Arbeit mit dem Computer vermitteln und so den Einstieg in die Welt der Informationstechnologie erleichtern. Denn wie so oft im Leben ist auch hier aller Anfang schwer. Eine solide Grundlage, wie sie das Erlernen einer Programmiersprache darstellt, sollte als flexibel

ausbaubares Fundament angesehen werden. Es erleichtert die weiteren Schritte wie z.B. die Einarbeitung in Anwendungssoftware, die auf das jeweilige Aufgabenumfeld zugeschnitten ist.

2. **Ziel:** Allgemeine Anleitung zu analytischem und strukturiertem Denken.

   Das vorliegende Buch soll als Teil einer Ausbildung im wirtschaftswissenschaftlichen Bereich gesehen werden. Der Programmierung eines Computers, sei es nun mit Pascal oder einer anderen strukturierten Programmiersprache, liegen nämlich ganz spezielle Denkmuster zugrunde. Denkmuster, die auch außerhalb des Bereichs der Programmierung zur Analyse und strukturierten Lösung von Problemstellungen verwandt werden können, wie sie im wirtschaftswissenschaftlichen Bereich häufig vorkommen. Wir betrachten die Programmiersprache Pascal aufgrund ihrer guten Verständlichkeit und Strukturiertheit als besonders geeignetes Hilfsmittel zum Training solcher Denkmuster und fragen erst in zweiter Linie nach der aktuellen Bedeutung im Bereich professioneller Programmierung.

   Selbst im Zeitalter graphisch orientierter „Window"-Programme ist es nicht nur für Programmierfreaks nützlich, Programmierkenntnisse zu besitzen. Auch bei der Arbeit mit Standardsoftwaresystemen im Bereich Textverarbeitung, Tabellenkalkulation oder Datenbankabfragen lassen sich einfache Programmstrukturen an mancher Stelle gewinnbringend einsetzen.

3. **Ziel:** Förderung der Kreativität und Motivation, auch eigene Probleme mittels einfacher Programme selbständig zu lösen.

   Die in diesem Buch angeführten Programmbeispiele und -aufgaben dienen zwar in erster Linie der Demonstration und als Hilfe beim Erlernen der Programmiersprache. Sie können auf der anderen Seite aber auch zur Lösung ganz konkreter Fragestellungen in der wirtschaftswissenschaftlichen Praxis eingesetzt werden, der sie entnommen sind.

   Die Programme zeigen zugleich, daß der Lösung solcher Fragestellungen nicht immer hochkomplizierte oder/und teure Programme zugrunde liegen müssen, sondern daß ein „do it yourself" durchaus machbar ist. Wir wollen unsere Leser motivieren, die mitgelieferten Programme für die eigenen Belange zu modifizieren oder zu erweitern sowie nicht zuletzt dazu, individuelle Fragestellungen mit Hilfe selbstgeschriebener Programme zu lösen.

   Unbestritten existieren Softwareprodukte zur computergestützten Lösung aller erdenklichen Problem- und Fragestellungen. Aber wer hat die zeitlichen und finanziellen Mittel, sich ständig Informationen über die Produkte beziehungsweise die Produkte selbst zu beschaffen? Ganz abgesehen vom Aufwand, der für die Installation der Software und die Einarbeitung in ihre Spezialitäten notwendig ist.

Wir hoffen, daß uns mit diesem Buch ein Beitrag zum Erreichen der gesteckten Ziele gelungen ist und möchten Sie als Leser ermuntern, uns Ihre Kritik und Anmerkungen zukommen zu lassen.

1.2 Aufbau

Das folgende **Kapitel 2** dient zur Orientierung und soll die Grundlagen für den Umgang mit dem PC schaffen. Wir geben außerdem einen kurzen Überblick über die Geschichte der Programmiersprachen unter besonderer Berücksichtigung von Pascal. Allgemeine Fragen des Softwareentwurfs werden anschließend in **Kapitel 3** behandelt.

Danach beschäftigen wir uns konkret mit der Programmierung in Pascal. **Kapitel 4** gibt eine Einführung in den Programmaufbau und einen Überblick über die wichtigsten Elemente der Programmiersprache, bevor wir in den **Kapiteln 5 und 6** auf die verschiedenen Arten von Anweisungen im Detail eingehen. In den folgenden Kapiteln behandeln wir speziellere Themen wie den Umgang mit selbstdefinierten und strukturierten Datentypen (**Kapitel 7 und 8**) sowie das Arbeiten mit Unterprogrammen (**Kapitel 9**).

Zu jedem „Programmierkapitel" bieten wir einen umfangreichen Fragen- und Aufgabenteil an, der dem Leser helfen soll, das Gelernte zu rekapitulieren und anzuwenden, um es so besser aufnehmen zu können. Die Lösungen zu den Programmieraufgaben, die in **Kapitel 10** zusammengestellt sind, lassen sich außerdem an mancher Stelle in der Praxis oder während des Studiums einsetzen. Darüber hinaus sollen sie auch zu weitergehender eigenständiger Programmierung motivieren.

Zur Programmierung verwendeten wir Turbo Pascal, ein eingetragenes Markenzeichen der Firma Borland Software Corp.

2 Grundlagen

2.1 Oft gehörte Bedenken

Vor dem ersten direkten Kontakt mit einem Computer steht oft Scheu oder sogar Abneigung gegenüber der Maschine. Die Akzeptanz stellt jedoch eine wichtige Voraussetzung für ein effektives Arbeiten mit der verwendeten Technik dar. Ihre Bedeutung sollte nicht unterschätzt werden. Dies gilt generell und nicht nur für den Bereich der Computertechnologie, erlangt hier aber eine besondere Bedeutung: Erstens kommt die Berührung mit der Technik für viele Menschen recht plötzlich und zweitens gibt es momentan wohl keinen Bereich, in dem ein rasanterer technischer Fortschritt zu beobachten wäre.

Umso wichtiger ist es, Akzeptanzprobleme zu erkennen, ernst zu nehmen und zu versuchen, sie zu lösen. Im folgenden gehen wir zunächst auf einige der Befürchtungen und Abneigungen ein, die in diesem Zusammenhang oft geäußert werden, bevor wir uns den wichtigsten Eigenschaften der Computer selbst zuwenden.

- „Da ich mich nicht auskenne, kann ich bestimmt leicht etwas kaputt machen, ohne in der Lage zu sein, dies wieder zu korrigieren."

  Ein Computersystem setzt sich grundsätzlich aus zwei Arten von Komponenten zusammen: Alles, was man anfassen kann, bezeichnet man als **Hardware**. Hierzu gehören beispielsweise der Bildschirm, die Tastatur, die Maus, möglicherweise ein Drucker und die Kiste selbst, welche die Geheimnisse birgt. Beim Umgang mit einem Computer sind selbstverständlich eine gewisse Vorsicht und die Befolgung einiger Regeln geboten. Der Umfang dieser Regeln entspricht jedoch dem für die Handhabung anderer elektrischer und elektronischer Geräte, wie etwa eines Fernseh- oder Videogeräts oder einer elektrischen Schreibmaschine.

  Meist bezieht sich die Angst vor der unbeabsichtigten Zerstörung auch nicht auf die Hardware, sondern auf die **Software**. Damit sind die Informationen gemeint, die auf den verschiedenen Speichermedien des Geräts für den Benutzer in der Regel unsichtbar abgelegt sind. Da die Informationen physisch kaum greifbar sind, braucht man Hilfsmittel, um sie sichtbar zu machen, Programme (ebenfalls Informationen in Form von Software) und Ausgabegeräte wie etwa den Bildschirm oder einen Drucker. Die Arbeit mit diesen Hilfsmitteln erscheint uns zunächst ungewohnt. Vor allem können wir nicht abschätzen, wann den unsichtbar abgelegten Informationen mittelbar oder unmittelbar Gefahr droht. Anders ist dies beispielsweise bei der Information, welche auf einem Blatt Papier gespeichert ist. Hier erkennen wir unmittelbar die Gefahr, wenn sich jemand anschickt, etwa einen Eimer Farbe oder eine Kanne Kaffee über dem Papier auszuleeren und wissen aus Erfahrung, was wir dagegen unternehmen können.

Da Informationen in Form von Software jedoch für jedermann in gleichem Maße unsichtbar sind und die Anbieter dieser Informationsprodukte normalerweise ein Interesse daran haben, daß diese nicht jedem Fehlgriff eines Benutzers zum Opfer fallen, steht vor der kompletten Zerstörung der Informationen meist eine ganze Reihe von optischen oder auch akustischen Warnungen. Solange ein Benutzer in der Lage ist, diese Meldungen zu registrieren, die über den Bildschirm oder den eingebauten Lautsprecher ausgegeben werden, dürfte es für ihn kaum möglich sein, unbeabsichtigt wesentliche Teile der gespeicherten Information zu zerstören.

- „Wie kann ich den Computer dazu bringen zu tun, was ich von ihm will?"

  Wer beginnt, sich mit der Computertechnologie auseinanderzusetzen, hat zumeist das Gefühl, sich in eine völlig neue und für ihn fremde Welt zu stürzen. Der Technik werden jede Menge wunderbarer Fähigkeiten zugeschrieben, was zur Folge hat, daß sich der Anfänger überfordert fühlt und sich selbst nicht zutraut, diese Fähigkeiten auszunutzen. Man sieht „den Wald vor lauter Bäumen" nicht mehr.

  Wir möchten Ihnen versichern: Mit einem Computer zu arbeiten, ist kein Hexenwerk, und Sie müssen auch nicht alle internen Abläufe verstehen, um ihn nutzen zu können. Sehr oft gilt, daß derjenige schon gewonnen hat, der die Meldungen aufmerksam lesen kann, die ihm auf dem Bildschirm ausgegeben werden. Da die meisten Software-Anbieter ein Interesse daran haben, daß ihre Kunden mit den Produkten korrekt umzugehen vermögen, bauen sie vielfältige Hilfefunktionen in ihre Programme ein. Diese klären den aufmerksamen Benutzer etwa über aktuelle Aktionsmöglichkeiten auf, über den Bearbeitungszustand, in dem sich eine Aktion gerade befindet oder möglicherweise auch über Fehler, die er gemacht hat und auf welche Weise er sie beheben kann.

  In der Regel werden Software-Produkte außerdem mit einer Fülle an Dokumentationsmaterialien ausgeliefert, die entweder in Form von Handbüchern vorliegen oder als direkt am Bildschirm aufrufbare online-Hilfe Bestandteil des Produkts selbst sind. Sie erlauben dem Benutzer eine Einarbeitung in das Produkt und Weiterbildung in Bereichen, in denen er ein besonderes Interesse oder mit denen er besondere Schwierigkeiten hat.

  Als Einsteiger sollten Sie sich anfangs nicht zuviel vornehmen, auf einer einfachen Stufe anfangen und sich dann langsam steigern. Außerdem sollten Sie nie vergessen, daß Sie als Computerbenutzer ein Kunde sind, dem das Gerät Dienste leisten soll und nicht umgekehrt.

- „Ich ziehe die Arbeit mit Papier und Bleistift und die direkte Kommunikation mit Menschen der Arbeit und Kommunikation mit einer Maschine vor."

  Die Angst vor dem Unbekannten beziehungsweise vor der Überforderung durch die Maschine kann eine Ursache für die generelle Ablehnung der Auseinandersetzung mit einem Computer sein.

Grundsätzlich sollten wir und als Menschen unserer herausragenden Fähigkeiten ständig bewußt bleiben, uns so der Technik immer überlegen fühlen und vergegenwärtigen, daß wir die Technik benutzen und nicht umgekehrt. Computer sind nämlich durch einige Eigenschaften gekennzeichnet, die unsere menschlichen in hervorragender Weise ergänzen können und die wir „gnadenlos" ausnutzen sollten.

Im folgenden Abschnitt werden einige der speziellen Eigenschaften der Computer beschrieben.

2.2 Allgemeine Charakteristika von Computern

Provozierend kann man sagen: **„Computer sind schnell, dumm und penibel."** Diese Charakteristika machen den Einsatz der Computertechnologie lohnenswert, solange sie in der richtigen Weise genutzt werden.

Was ist damit nun genau gemeint?

- Computer sind schnell.

  Diese Aussage gilt vor allem für die Ausführung numerischer Aufgaben. Tatsächlich kommt das Wort Computer vom lateinischen computare, was berechnen bedeutet. Gerade in der Ausführung von Berechnungen wie Addition, Multiplikation und so fort lagen deshalb lange Zeit die Hauptanwendungsgebiete der Computertechnologie. Da sie außerdem sehr genau arbeiten und ihnen in technisch einwandfreiem Zustand in der Regel keine Rechenfehler unterlaufen, benutzte man sie vor allem als „Rechenknechte" oder „Number-Cruncher".

  Beispielsweise braucht ein Pascal-Programm, das von eins bis eine Million zählt, abhängig von der verwendeten Hardware nur den Bruchteil einer Sekunde. Wir brauchen im Vergleich dazu ungefähr elfeinhalb Tage, wenn wir pro Sekunde nur eine Zahl zählen und uns nicht verzählen.

- Computer sind dumm.

  Ohne genaue Anweisungen „können" Computer nichts. Hat ein Hersteller alle mechanischen und elektronischen Bauteile eines Computers (die Hardware) gefertigt und zusammengesetzt, so ist das Gerät noch nicht betriebsbereit. Es weiß noch nicht, aus welchen Teilen es besteht, woher es Eingaben zu erwarten hat, wie und in welcher Reihenfolge es Informationen verarbeiten soll, wohin es Ergebnisse ausgeben oder speichern soll und so weiter. Es braucht vom Start weg ständig Informationen darüber, was es „zu machen" hat. Die Software enthält diese Informationen, mit deren Hilfe einem Computer Ausführungsbefehle gegeben werden können.

  Informationen, die einem Computer beispielsweise direkt nach dem Einschalten Anweisungen geben, stehen teilweise in einem Festwertspeicher, auch

ROM (Read-Only-Memory) genannt. Sie werden direkt bei der Herstellung festgelegt und können dann später nicht mehr verändert werden. Vom ROM werden Aufgaben übernommen wie etwa:

* Anmeldung der Einzelteile wie Bildschirm, Tastatur oder Laufwerke bei der Zentraleinheit des Rechners,
* Durchführung eines Funktionstests der Komponenten,
* Einladen weiterer Informationen wie z.B. des Betriebssystems in den Hauptspeicher.

- Computer sind penibel bzw. buchstabengläubig.

  Jede Anweisung wird wörtlich genommen, und Rechtschreibfehler beim Programmieren werden nicht verziehen. Denn ein Computer führt genau das aus, was Sie ihm befehlen, nicht mehr und nicht weniger. Vor allem sollten Sie sich merken: **„Ein Computer tut das, was man ihm sagt, nicht das, was man von ihm will."**

  Gerade als Anfänger wundert man sich oft, warum der Computer seltsam reagiert. Dies liegt jedoch zumeist nicht am Gerät, sondern an den (falschen) Anweisungen, die man ihm gegeben hat.

  Erteilen Sie dem Computer mit Hilfe eines Programms beispielsweise die Anweisung „Zähle von null bis eine Million, dabei soll zwischen zwei Werten eine Schrittweite von null liegen", so zählt der Computer bis in alle Ewigkeit: `0,0,0,....`

  Sie müssen dem Computer mittels der Software also alles sagen:

  * Was er genau tun soll.
  * Auf welche Weise er es tun soll.
  * In welcher Reihenfolge er die einzelnen Aktionen ausführen soll.

  Die meisten Anwender können hierzu auf schon bestehende Software-Produkte zurückgreifen, was die Sache insgesamt doch sehr erleichtert.

2.3 Programmiersprachen

Eine Programmiersprache ist eine Sprache zur Formulierung von Anweisungen, die von einem Computer ausgeführt werden können. Damit bildet sie eine wichtige Schnittstelle zwischen Benutzer und Computer. Um Mehrdeutigkeiten zu vermeiden, ist es notwendig, die Strukturelemente einer Programmiersprache eindeutig zu definieren. Dies geschieht mit Hilfe von Syntax und Semantik. Vergleichen Sie dazu Kapitel 4.1. Man teilt die Programmiersprachen nach dem Grad ihrer Hardwareabhängigkeit in Generationen ein. Diese Klassifikation fällt in etwa mit den historischen Entwicklungsschritten zusammen, ohne daß jedoch eine Generation die ande-

re völlig abgelöst hätte. Vielmehr existieren heute alle Generationen mit mehr oder weniger großer Bedeutung nebeneinander.

2.3.1 Erste Generation: Maschinensprache der „Computersteinzeit"

Als elektronisches Gerät arbeitet ein Computer intern nur mit zwei Zuständen, die man sich als „Strom fließt/Strom fließt nicht", „Schalter offen/Schalter geschlossen" oder einfach als 0 und 1 vorstellen kann. Dieses Grundprinzip gilt seit der Erfindung der elektronischen Rechenmaschinen bis heute.

Vor allem am Anfang der Programmierung (in den 50er und 60er Jahren) befaßten sich die Programmierer direkt mit diesen zwei intern darstellbaren Zuständen. Die Programmierung der Rechner erfolgte durch das Kippen von Schaltern, wodurch dann Verbindungen innerhalb des Computers hergestellt wurden. Das heißt, daß alle Operationen, Zahlen und Buchstaben usw. nur durch Ketten der beiden Zeichen 0 und 1 dargestellt werden mußten. Diese Art der Darstellung nennt man Binärcode.

Eine Programmierung für eine Addition der Zahlen 3 und 4 hätte z.B. folgende Form gehabt:

- Umrechnung beider Operanden in den Binärcode:
  3: `0011`, 4: `0100`
- Additionsbefehl für den Inhalt der Speicherplätze für die beiden Zahlen ebenfalls in Binärcode:
  Addition: `00011010`
- Nach der Verarbeitung des Befehls konnte man das Ergebnis je nach Konfiguration der Rechenanlage beispielsweise als blinkende Lichterreihe in Binärcode ablesen und anschließend eventuell wieder in eine Dezimalzahl zurücktransferieren.
  `0111`: Binärcode für die Dezimalzahl 7.

Die Nachteile, in dieser Art zu programmieren, liegen auf der Hand: Programme in Maschinensprache sind wegen des auf zwei Zeichen beschränkten Zeichenvorrats für den Menschen kaum verständlich, extrem unübersichtlich, und die Programmierung ist deshalb sehr fehleranfällig.

2.3.2 Zweite Generation: Programmierkurzschrift Assembler

Die frustrierten Programmierer fanden im Laufe der Zeit Wege, den Computer Anweisungen ausführen zu lassen, die nicht direkt im Binärcode eingegeben werden. Statt dessen werden die Befehle und Operanden durch leichter verständliche mnemonische Abkürzungen dargestellt, durch Abkürzungen also, die etwas über die Bedeutung der von ihnen dargestellten Objekte aussagen. Diese Kurzschrift nennt man Assemblersprache. Durch ihre Verwendung werden die Programme für den Men-

schen verständlicher, leichter handhabbar und insgesamt etwas abgekürzt. Der maschinenorientierte Aufbau der Befehle wurde dabei jedoch beibehalten.

So könnte die oben angesprochene Aufgabe der Addition von 3 und 4 in Assembler folgendermaßen gelöst werden:

- Einladen der Zahl 3 in den Speicherplatz A:
  `LOAD A,3`
- Addition der Zahl 4 zum Inhalt des Speicherplatzes A:
  `ADD A,4`

Bevor ein Assembler-Programm vom Computer ausgeführt werden kann, muß die Kurzschrift weiterhin in Maschinensprache übersetzt werden, weil dies immer noch die einzige Sprache ist, die der Computer versteht. Das geschieht mit Hilfe entsprechender Übersetzungsprogramme (Compiler), die in diesem Fall, wie die zu übersetzende Sprache auch, Assembler genannt wird.

Assembler ist immer noch maschinenorientiert, d.h. die Programmierung in Assemblersprache verlangt immer noch genaue Hardwarekenntnisse. Für jeden Prozessortyp existieren spezielle, auf seinen Befehlsvorrat zugeschnittene Assembler-Sprachen. Dadurch erlaubt die Sprache ein optimales Programmieren, was Speicherbedarf und Verarbeitungsgeschwindigkeit der Programme betrifft, bietet dafür aber nur wenig Komfort sowohl bei der Programmierung als auch bei der späteren Wartung der Programme. Ihr Hauptanwendungsgebiet liegt in der Erstellung zeitkritischer oder häufig benutzter Programmteile.

Wegen der engen Hardware-Orientierung ist eine Übertragung (Portierung) der Programme problematisch. Das bedeutet, bei Herstellerwechsel oder Wechsel der Rechnerfamilie ist praktisch eine Neuprogrammierung der Problemlösungen erforderlich. Außerdem ist die Erstellung komplexerer Anwendungsprogramme kaum möglich, da Assembler-Programme mit zunehmender Größe schwer verständlich und unübersichtlich werden. Die Programmierung bleibt infolgedessen fehleranfällig.

2.3.3 Dritte Generation: Höhere Programmiersprachen

Die Entwicklung ging weiter, und es wurden auf der nächsten Stufe die höheren Programmiersprachen, die sogenannten Sprachen der dritten Generation entworfen.

Diese Programmiersprachen ermöglichen eine von der Hardware weitgehend unabhängige Programmierung. Das bedeutet, daß ein Programmierer keine Kenntnisse der technischen Details der Maschine, auf der das Programm laufen soll, mehr benötigt. Er muß nur noch die Sprache kennen. Gleichzeitig zeichnen sich höhere Programmiersprachen dadurch aus, daß sie sich an den zu bearbeitenden Problemfeldern ausrichten, weshalb man sie auch problemorientiert nennt. Aus diesem Grund ist das Programmieren mit Hilfe einer höheren Programmiersprache wesentlich be-

quemer als mit einer maschinenorientierten. Gleichzeitig sind die Programme leichter lesbar.

Der Befehl zur Addition der Zahlen 3 und 4 lautet in Pascal beispielweise:

```
summe := 3 + 4
```

Wie bei der Assemblersprache muß auch ein Programm in einer höheren Programmiersprache in die Maschinensprache des Computers übersetzt werden, bevor er mit der Arbeit beginnen kann. Während bei Assemblerprogrammen eine Programmanweisung in jeweils einen Maschinenbefehl übersetzt wird, entspricht bei Programmen in höheren Programmiersprachen eine Programmanweisung jeweils einer ganzen Reihe von Befehlen im Maschinencode. Dies vereinfacht die Programmierung erheblich. Als Faustregel gilt: Ein einziger Programmierer erledigt in einer höheren Programmiersprache die Arbeit von zehn Assemblerprogrammierern. Durch die notwendige Übersetzung sind die resultierenden ablauffähigen Programme andererseits nicht so schnell wie direkt in Maschinensprache programmierte und brauchen gleichzeitig mehr Speicherplatz.

Das **Programm der höheren Programmiersprache** wird als **Quelltext**, Quellprogramm oder auch Quellcode bezeichnet, die **in Maschinensprache übersetzte Version** als **Objektcode**, Objektprogramm oder auch Maschinencode. Das **Übersetzungsprogramm** heißt **Compiler**, der Übersetzungsvorgang compilieren.

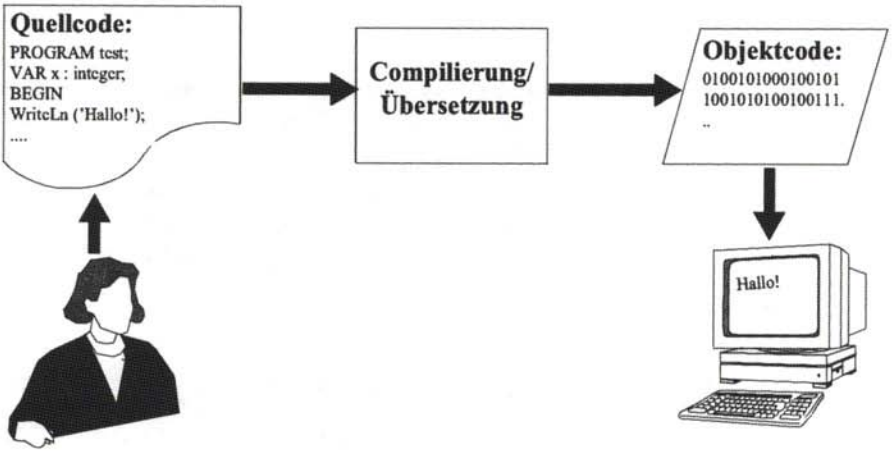

Abb. 1: Arbeit mit einer höheren Programmiersprache

Beispiele für höhere Programmiersprachen, die zur Umwandlung des Quellcodes Compiler verwenden, sind Pascal, Cobol, Fortran oder C.

Daneben existiert noch eine zweite Art der höheren Programmiersprachen: Hier wird das Quellprogramm mit Hilfe eines Interpreters analysiert. Jede Anweisung wird einzeln in Maschinensprache umgewandelt und anschließend sofort ausge-

führt. Dadurch wird die Programmausführung insgesamt flexibler als bei der Verwendung von Compilern, gleichzeitig aber auch langsamer. Beispiele für Interpretersprachen sind Basic oder APL.

Auf jeden Fall ist es wichtig, daß der Compiler bzw. Interpreter einen Objektcode erzeugt, der von der Maschine, welche die Anweisungen ausführen soll, verstanden wird. Deshalb muß bei der Verwendung einer geänderten Rechnerarchitektur nur der Übersetzer als Schnittstelle, nicht aber die einzelnen Quellprogramme angepaßt werden. Die Programme sind also übertragbar (portierbar). Außerdem ermöglichen höhere Programmiersprachen die sogenannte strukturierte Programmierung und damit ein ingenieurmäßiges Vorgehen bei der Softwareerstellung. Hierauf werden wir in Kapitel 3 noch ausführlicher eingehen.

2.3.4 Weiterentwicklungen

Die Geschichte der Programmiersprachen endet nicht in der dritten Generation. Man spricht inzwischen von Programmiersprachen der vierten und sogar der fünften Generation. Die Abgrenzungen sind hier im Gegensatz zu den vorhergehenden alles andere als eindeutig. Das kommt vor allem daher, daß die Entwicklungen auf diesem Gebiet noch in vollem Gange sind.

Eine mögliche Einteilung ist die folgende: Programmiersprachen und Softwaresysteme der **vierten Generation** werden unter der Zielsetzung „effiziente Anwendungsentwicklung" auf einen Nenner gebracht. Hierzu gehören beispielsweise Programmiersprachen, die eine deskriptive Programmierung erlauben. Dabei muß der Programmierer im Quellcode nicht mehr angeben, **auf welche Weise** ein Problem gelöst werden soll, sondern er legt in beschreibender Form fest, **was** geschehen soll. SQL ist ein weit verbreitetes Beispiel einer deskriptiven Programmiersprache zur Arbeit mit Datenbanken. Programmiersprachen der vierten Generation sind in der Regel auf ein enges Problemlösungsgebiet wie z.B. Datenbankanwendungen spezialisiert und nicht universell einsetzbar wie etwa eine strukturierte Programmiersprache der dritten Generation.

Hauptanwendungsbereich von Systemen der **fünften Generation** ist die künstliche Intelligenz. Hierunter versteht man den Versuch, mit Hilfe von Computertechnologie bisher dem Menschen vorbehaltene Intelligenzleistungen nachzubilden. Auf diesem Gebiet werden unter anderem logische Programmiersprachen wie etwa Prolog eingesetzt. Bisher konnten die hochgesteckten Erwartungen, die man in den 80er Jahren insbesondere von japanischer Seite aus an die Systeme der fünften Generation stellte, jedoch noch nicht erfüllt werden.

Im folgenden Abschnitt geben wir einen Überblick über die in diesem Buch behandelte Programmiersprache Pascal, eine Sprache der dritten Generation.

2.4 Die Programmiersprache Pascal

2.4.1 Entstehungsgeschichte

Die Programmiersprache Pascal ist nach dem französischen Mathematiker Blaise Pascal (1623-1662) benannt, der schon 1641 eine mechanische Addiermaschine für sechsstellige Dezimalzahlen konstruierte. Pascal wurde Anfang der 70er Jahre vom Computerwissenschaftler Niklaus Wirth an der ETH Zürich entwickelt. Wirth wollte seinen Studenten mit dieser Sprache vor allem ein effektives Erlernen des Programmierens ermöglichen.

Die konsequente Ausrichtung der Sprache auf die strukturierte Programmierung ist der Hauptgrund dafür, daß Pascal auch heute noch in der Datenverarbeitungs-Grundausbildung im Hochschulbereich weit verbreitet ist. Die Programme lassen sich leicht in übersichtliche Blöcke unterteilen, und Regeln bestimmen, welche Programmteile an welchen Stellen des Programms zu stehen haben. Insgesamt ist Pascal, verglichen mit anderen Programmiersprachen, leicht zu erlernen. Da die zugrunde liegenden Elemente außerdem zum großen Teil selbsterklärend sind, bleiben auch komplexe Programme relativ gut lesbar.

Seit Mitte der 70er Jahre wurde versucht, Pascal international zu normen. Dies führte 1982 zum Normenentwurf ISO 7185 (International Organization for Standardization), der von vielen nationalen Normungsinstitutionen übernommen worden ist. In Deutschland ist Standard-Pascal 1983 als Normentwurf DIN 66256 in deutscher Sprache veröffentlicht worden.

2.4.2 Bestandteile des Software-Pakets Turbo-Pascal

Wir behandeln in diesem Buch die strukturierte Programmierung am Beispiel von Pascal unter der Verwendung von Turbo-Pascal, einem recht weit verbreiteten Produkt der Firma Borland, dem das normierte Standard-Pascal zugrunde liegt. Die Programmelemente, die in diesem Buch vorrangig beschrieben werden, können sowohl in der letzten Version 7.0 als auch in älteren Versionen ohne Probleme angewendet werden.

Die für uns wichtigen Bestandteile der verwendeten Software Turbo-Pascal sind folgende:

1. **Compiler**

   Der Compiler ist das „Herzstück" von Turbo-Pascal. Wie oben schon gesagt: Egal welche Programmiersprache man verwendet, immer müssen die für uns verständlichen Worte des Quellcodes in die für den Computer verständliche Maschinensprache übersetzt werden, damit ein Programm „ablauffähig" wird. Dies ist die Aufgabe des Compilers.

2. **Entwicklungsumgebung**

   Der zweite Bestandteil des Software-Pakets Turbo-Pascal ist die integrierte Entwicklungsumgebung. Sie soll dem Programmierer Hilfestellung bei der Programmerstellung bieten und somit die Arbeit mit dem eigentlichen Compiler erleichtern. Zu ihr gehören:

   * Ein **Texteditor** nach der Art eines Textverarbeitungsprogramms unterstützt das Schreiben und Ändern der Quelltexte.
   * Dem Benutzer werden Mechanismen zur **Dateiverwaltung** wie etwa zum Laden oder Speichern von Programmen zur Verfügung gestellt.
   * Ein weiterer Bestandteil wird **Debugger** genannt. Er bietet Mechanismen zur Fehlersuche in den Quelltexten.
   * Schließlich gibt es noch einen **Linker**, der nach der Compilierung den Objektcode mit zusätzlich benötigten Bibliotheksroutinen verbindet und so ein lauffähiges Programm erzeugt.

Bevor wir uns den Bestandteilen von Pascal zuwenden, soll im folgenden Kapitel ein allgemeiner Überblick über das Vorgehen bei der Programmierung gegeben werden.

3 Programmierung und Programmiertechniken

3.1 Ablauf einer Programmentwicklung

Damit der Computer eine Aufgabe lösen oder ausführen kann, müssen ihm genaue Anweisungen gegeben werden, was er wie und in welcher Reihenfolge zu tun hat. Mehrere Anweisungen zur Lösung eines Problems werden zu einem Programm zusammengefaßt. Ein Programm wird auch als Software bezeichnet.

Je nach Art der zu lösenden Aufgabe besitzen die Programme eine unterschiedliche Komplexität. Insbesondere bei der Entwicklung großer und komplexer Programme ist es notwendig, daß der Programmentwickler eine **ingenieurmäßige Vorgehensweise** wählt. Dies erleichtert das Erstellen eines anforderungsgerechten Programms.

Ein ingenieurmäßiges Vorgehen zeichnet sich durch folgende Merkmale aus:

1. Es existiert ein **Entwicklungsprozeß**, der strikt verfolgt wird.
2. **Termin- und Kostenkontrollen** werden regelmäßig durchgeführt.
3. Die **Qualität** der bearbeiteten (Teil-) Aufgaben wird systematisch **kontrolliert**.
4. Bei Programmentwicklungen durch mehrere Personen werden Methoden zur **Teamentwicklung und -organisation** eingesetzt.
5. Es werden **Methoden und Werkzeuge** zur Unterstützung der Phasen des Entwicklungsprozesses (CASE-Tools) eingesetzt.
6. Die problemadäquate **Einbeziehung** des späteren **Benutzers** wird gewährleistet.

Dieses ingenieurmäßige Vorgehen soll sicherstellen, daß die mit der Programmentwicklung verbundenen Ziele erreicht werden. Wichtige Ziele, die das Programm als Endprodukt dieses Prozesses erreichen soll, sind dabei:

1. **Funktionalität**: Das Problem wird adäquat gelöst.
2. **Zuverlässigkeit**: Das Programm ist technisch fehlerfrei.
3. **Wartbarkeit**: Das Programm ist leicht änderbar und damit an veränderte Anforderungen anpassungsfähig.
4. **Effizienz**: Die Systemressourcen werden sparsam eingesetzt.
5. **Benutzerfreundlichkeit**: Das Programm ist leicht bedienbar.
6. **Kostentreue**: Die Entwicklungskosten bleiben im geplanten Rahmen.
7. **Termintreue**: Der geplante Zeitbedarf für die Systementwicklung wird nicht überschritten.

Obwohl das ingenieurmäßige Vorgehen eher für die Entwicklung umfangreicher Programme gedacht ist, sollten Sie es auch für die im Rahmen dieses Buches gestellten Aufgaben im Hinterkopf behalten. Schließlich wollen auch Sie gute Programme im Sinne der oben genannten Ziele schreiben.

Für die Entwicklung einfacher (Pascal-) Programme schlagen wir folgende Vorgehensweise vor:

1. **Definition der Aufgabe**:

   Die mit dem Programm zu lösende Aufgabe ist auf ihre Anforderungen hin zu analysieren und möglichst genau zu definieren.

2. Festlegen der **Programmstruktur** und ihrer Inhalte:

   Ein Programm sollte nach dem **EVA-Prinzip** aufgebaut werden. Das heißt es erfolgt zuerst das **E**inlesen bzw. Bereitstellen aller benötigten Daten. Dann werden diese Daten durch geeignete Lösungsalgorithmen **v**erarbeitet und schließlich die Lösung **a**usgegeben.

   In diesem zweiten Schritt sind somit auch die **Datenstrukturen** festzulegen und geeignete **Lösungsalgorithmen** zu finden.

3. Formulieren des Programms in einer **Beschreibungssprache**:

   Zu den Beschreibungssprachen gehören graphikorientierte Hilfsmittel, wie zum Beispiel Flußdiagramme und Struktogramme. Sie dienen der übersichtlichen Festlegung des Programmablaufs und erleichtern somit einen ersten Programmentwurf und dessen Überprüfung.

4. Formulieren des Programms in einer höheren **Programmiersprache**:

   Auf Grundlage der Beschreibungssprache wird das Programm im Quellcode geschrieben.

5. **Übersetzen** des Programms:

   Damit der Quellcode vom Computer verstanden wird, ist er in die Maschinensprache zu übersetzen. Dies erledigt der Compiler.

6. **Programmtest**:

   Das Durchführen dieses Schritts ist sehr wichtig. Denn ein Programm, auf dessen Ergebnisse man sich nicht verlassen kann, ist wertlos. Daher sollte ein Programm mehrmals mit unterschiedlichen geeigneten Werten getestet werden.

   Es lassen sich drei Fehlerarten unterscheiden:

   a) **Fehler zur Übersetzungszeit**: Beim Übersetzen des Quellcodes in die Maschinensprache werden die syntaktischen und die semantischen Fehler vom Compiler erkannt. Das sind in anderen Worten die Grammatik- und Aussagefehler im Quellcode (vgl. Kapitel 4.1.1).

   b) **Laufzeitfehler**: Diese Fehler treten beim Ablauf des Programms auf und führen zum Programmabbruch durch das Betriebssystem. Das sind z.B. eine Division durch 0 oder Wertebereichsüberschreitungen.

   c) **Logische Fehler**: Wird die Aufgabenstellung durch das Programm nicht gelöst, weil z.B. ein unbrauchbarer Lösungsalgorithmus gewählt wurde, handelt es sich um einen logischen Fehler. Dieser wird vom Compiler nicht erkannt.