

Skriptum Informatik

- eine konventionelle Einführung

Prof. Dr. Hans-Jürgen Appelrath
Universität Oldenburg

Prof. Dr. Jochen Ludewig
Universität Stuttgart



B. G. Teubner Stuttgart Hochschulverlag AG an der ETH Zürich

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Appelrath, Hans-Jürgen:

Skriptum Informatik : eine konventionelle Einführung / Hans-Jürgen Appelrath ; Jochen Ludewig. 3., überarb. und erw. Aufl. – Stuttgart : Teubner ; – Zürich : Hochschulverl. an der ETH, 1995

ISBN 978-3-519-22153-1

ISBN 978-3-322-91824-6 (eBook)

DOI 10.1007/978-3-322-91824-6

NE: Ludewig, Jochen:

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung der Verlage unzulässig und strafbar. Das gilt besonders für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

1. Auflage 1991

2., durchgesehene Auflage 1992

3., überarbeitete und erweiterte Auflage 1995

© 1995 B. G. Teubner Stuttgart

und vdf Hochschulverlag AG an der ETH Zürich

Umschlaggestaltung: Fred Gächter, Oberegg, Schweiz

Vorwort zur 1. Auflage

Der Titel dieses Buches ist Beschreibung und Abgrenzung: Wir legen ein Skriptum einer zweisemestrigen Einführungsvorlesung Informatik vor, also einen Text, der im wesentlichen anlässlich von Lehrveranstaltungen der Verfasser an der ETH Zürich (1985-88), der Universität Stuttgart (1988-90) und der Universität Oldenburg (1989-90) aufgeschrieben wurde. Und diese Vorlesungen für Studenten¹ der Hauptfach-, einer Nebenfach- oder einer sogenannten Bindestrich-Informatik waren und sind *konventionell* in dem Sinne, daß das Erlernen einer modernen, aber imperativen Programmiersprache (Modula-2) ein wichtiges Teilziel darstellt. Auch das Vorgehen dabei, bottom-up, vom kleinen zum großen, ist durchaus konventionell.

Die Wahl der Sprache ist umstritten; andere Ansätze legen eine nicht-imperative Sprache zugrunde oder ziehen es vor, die Theorie besonders zu betonen. Jeder kann für sein Konzept gute Gründe vorbringen; unsere sind die folgenden:

- In der Praxis – und dort werden fast alle nach erfolgreichem oder auch abgebrochenem Studium landen – haben die nicht-konventionellen Programmiersprachen bis heute keinen nennenswerten Anteil erreicht; daran wird sich auch in der absehbaren Zukunft nicht viel ändern. Nur durch die Ausbildung mit einer imperativen Sprache können wir also einen Beitrag zu besserem *Software Engineering* leisten. Die Vermittlung der *Konzepte* genügt nicht, denn Programmieren hat auch eine starke handwerkliche Komponente, die detaillierte Hinweise und Übung erfordert.
- Kaum ein Student beginnt das Informatik-Studium ohne Programmiererfahrung. Leider sind diese Erfahrungen zum großen Teil nicht nützlich; noch immer geistert BASIC durch private und leider auch schulische Rechner, und auch was in Pascal gefaßt wurde, ist oft kein sauberer, systematisch entwickelter Code. Vor diesem Hintergrund mag dieses Skriptum auch als Orientierungshilfe für den Informatik-Unterricht in der Oberstufe dienen. Unterrichten wir die Erstsemester nur in Prolog oder in einer anderen nicht-konventionellen Sprache, so setzen wir dieser Fehlentwicklung nichts entgegen, sondern bauen eine entrückte *Zweitwelt* auf. Die erste im Grundstudium eingesetzte Sprache sollte praxistauglich und hinreichend verfügbar sein. Modula-2 erfüllt beide Anforderungen.
- Eine Ausrichtung der Vorlesung auf den Objektbegriff erschien uns durchaus attraktiv, nur sehen wir keinen Weg, mit Objekten zu arbeiten, bevor die *elementaren* Begriffe systematisch eingeführt sind. Darum schätzen wir Modula-2 als eine Sprache, die die *Vorbereitung* des Objektbegriffs unterstützt. Da man in der Pro-

¹Wir verzichten in diesem Buch auf jeden Versuch, geschlechtsneutrale Formulierungen zu erreichen, denn alle uns bisher bekannten Lösungsansätze dieses Problems erscheinen unbefriedigend. Wir stellen allerdings ausdrücklich fest, daß dieses Problem besteht und einer Lösung bedarf.

grammierung das Detail beherrschen muß, bevor man zum „Programmieren im Großen“ übergeht, sehen wir auch keine Alternative zum bottom-up-Konzept.

Nicht zuletzt: Wir haben mit Modula-2 viel Erfahrung, so daß wir in der Vorlesung aus dem Vollen schöpfen können – und wir kennen die Unzulänglichkeiten gut genug, um den Unterschied zwischen abstraktem Konzept und konkretem Konstrukt der Sprache deutlich zu machen und damit allgemein auf eine kritische Einstellung gegenüber *jeder* Programmiersprache hinzuwirken.

„Konventionell“ und „praxisnah“ bedeuten freilich nicht, daß wir uns am Durchschnitt der heutigen Programmierpraxis orientieren. Im Gegenteil bemühen wir uns um begriffliche Klarheit und um eine angemessene Berücksichtigung der Theorie, beispielsweise durch die Behandlung der Themen Turing-Maschinen, Berechenbarkeit, Grammatiken, Semantik, Programmverifikation und Komplexität, auch wenn wir damit eine spätere Grundvorlesung Theoretische Informatik nur vorbereiten können.

Der Titel „Skriptum Informatik – eine konventionelle Einführung“ soll auch den Werkzeug-Charakter betonen. Offensichtlich sind die Konflikte zwischen Systematik und Didaktik, zwischen erwünschter Vollständigkeit und notwendiger Beschränkung, nicht generell lösbar. Wir haben im Zuge der Vorbereitung immer wieder darüber gestritten, wie die Abschnitte zu ordnen seien, was wichtig und was entbehrlich ist. So liegt zwischen den Buchdeckeln nun ein Kompromiß, aus dem jeder Dozent, jeder Leser *seinen* Stoff in *seiner* Reihenfolge herauschälen möge.

In Vorbereitung befindet sich ein umfangreicher Übungsband mit zahlreichen Aufgaben und Musterlösungen, der ebenfalls bei vdF und Teubner erscheinen wird.

Lehrbuchautoren sind *Compiler* im ursprünglichen Wortsinn, sie tragen zusammen, übersetzen und verbinden, was andere geschaffen haben. Darum geht unser Dank an die Autoren aller Quellen, die wir – sicher nur in beschränktem Maße bewußt – verwendet haben. Ganz bewußt allerdings haben wir Quellen von Prof. Dr. V. Claus und Dr. A. Schwill (beide Oldenburg), insbesondere den Informatik-Duden (Dudenverlag, 1989), genutzt.

Viele Studierende und Mitarbeiter haben in den letzten Jahren durch Kritik und Korrekturen dabei geholfen, Rohfassungen in ein Buch zu verwandeln. Herr Dipl.-Inform. Rainer Götze hatte in der letzten Phase die anspruchsvolle Funktion des Redakteurs und Koordinators und damit wesentlichen Anteil am Zustandekommen. Frau Claudia Martsfeld hat mit Geduld und Einfühlungsvermögen große Teile des Manuskripts geschrieben. Ihnen allen gilt unser Dank.

Oldenburg/ Stuttgart, im August 1991

Hans-Jürgen Appelrath und Jochen Ludewig

Vorwort zur 3. Auflage

Die weiterhin erfreuliche Nachfrage gestattet uns, einige der Unzulänglichkeiten, die erst in der 2. Auflage sichtbar geworden sind, zu beheben und eine verbesserte 3. vorzulegen. Überarbeitet wurden nur der Abschnitt 4.3 zum Thema „Testen“ und Kapitel 5, in dem wir verschiedene Programmierstile zeigen. In der alten Fassung hatten wir uns damit begnügt, die verschiedenen Lösungen eines Beispiel-Problems zu skizzieren, jetzt sind sie codiert und wurden mit einer Ausnahme auch übersetzt und ausgeführt.

Wie bisher gibt es in diesem Buch keine Übungsaufgaben; sie sind dem Übungsband von Spiegel, Ludewig und Appelrath zu entnehmen, der im Frühjahr 1992 ebenfalls als Gemeinschaftsproduktion der Verlage Teubner und vdf erschienen und seit 1994 in der 2. Auflage verfügbar ist. In diesem inhaltlich und strukturell auf das vorliegende Skriptum abgestimmten Buch sind viele Aufgaben mit Tips und Lösungen zu finden, die die „Einführung in die Informatik“ unterstützen und wesentlich erleichtern. Der Code aller Programme, die im Skriptum und im Übungsband abgedruckt sind, ist auch elektronisch verfügbar.

Wie schon vor zwei Jahren haben wir Anlaß, uns bei aufmerksamen Lesern – hier möchten wir beispielhaft Herrn Kollegen Bruno Müller-Clostermann nennen – zu bedanken, die auf Mängel des Skriptums hingewiesen haben, und wir hoffen weiterhin auf Ihre Kommentare. Es wäre für uns darüber hinaus interessant, von den Erfahrungen weiterer Kolleginnen und Kollegen zu wissen, die das Skriptum in ihren Lehrveranstaltungen einsetzen. Wir freuen uns über jede Mitteilung und bieten unsere Hilfe an, wenn es um technische Unterstützung geht, z.B. auch die elektronische Bereitstellung der Abbildungen des Skriptums.

Rainer Götze und Gerhard Möller haben uns wie bei den beiden ersten Auflagen wieder als „Qualitätssicherer“ kritisch und konstruktiv zur Seite gestanden.

Freundliche Unterstützung bei der Überarbeitung des Kapitels 5 haben Angela Georgescu, Marcus Deininger, Kurt Schneider und Christian Rathke geleistet.

Claudia Martsfeld hat sorgfältig wie bei den ersten beiden Auflagen die Schreib- und Zeichenarbeit übernommen. Dabei wurden zusätzlich die Abbildungen, die bislang oft, wenn auch kaum merklich durch den Transfer vom einem ins andere System gelitten hatten, nach Postscript umgesetzt und sollten nun so im Buch stehen, wie sie ursprünglich erstellt worden sind.

Oldenburg/Stuttgart, im April 1995

Hans-Jürgen Appelrath und Jochen Ludewig

Inhaltsverzeichnis

1. Grundlagen	11
1.1 Algorithmus und Berechenbarkeit	11
1.1.1 Algorithmus	11
1.1.2 Turing-Maschine	13
1.1.3 Berechenbarkeit	18
1.2 Sprache und Grammatik	21
1.2.1 Sprache	21
1.2.2 Grammatik	25
1.3 Rechner	31
1.3.1 Von-Neumann-Rechnerarchitektur	32
1.3.2 Rechnersysteme	38
1.4 Informatik als Wissenschaft	46
2. Imperative Programmierung – die Sprache Modula-2	53
2.1 Syntaxdarstellungen	53
2.2 Elementare funktionale Modula-2-Programme	57
2.2.1 Eine Modula-2-Teilsprache	57
2.2.2 Programmverzweigungen	69
2.2.3 Funktionen und Prozeduren	72
2.2.4 Elementare Datentypen, Aufzählungs- und Bereichstypen	74
2.2.5 Eingabevariablen	79
2.2.6 Rekursive Funktionen und Prozeduren	80
2.2.7 Nachteile funktional-rekursiver Programme	85
2.3 Iterative Programme	87
2.3.1 Wertzuweisungen und Referenzparameter	87
2.3.2 Gültigkeitsbereich und Lebensdauer	89
2.3.3 Anweisungen zur Iteration	96
2.3.4 Vergleich iterativer und rekursiver Lösungen	99
2.3.5 Sprunganweisungen	102
2.3.6 Prozedurtypen	104
2.4 Komplexe Datentypen	109
2.4.1 Mengen (Sets)	109
2.4.1.1 Darstellung und Manipulation von Mengen	110
2.4.1.2 Ein Beispiel für Sets	111
2.4.2 Arrays (Felder)	113
2.4.3 Records (Verbunde)	119
2.4.3.1 Einfache Records	120

2.4.3.2	Records mit Varianten	126
2.4.4	Zeiger (Pointer) und dynamische Variablen	129
2.4.4.1	Die Speicherung auf der Halde	130
2.4.4.2	Operationen auf Zeigern	132
2.4.4.3	Verkettete Listen	136
2.4.4.4	Anwendungen und Probleme dynamischer Variablen	145
2.4.5	Dateien (Files)	148
2.4.5.1	Eigenschaften und formale Beschreibung	148
2.4.5.2	Dateien in Pascal	150
2.4.5.3	Dateien in Modula-2	153
3.	Abstraktion	157
3.1	Abstraktionskonzepte in Programmiersprachen	157
3.2	Abstraktion in Modula-2	161
3.2.1	Das Prinzip der separaten Übersetzung	161
3.2.2	Modularisierung eines Programms	163
3.2.3	Datenkapselung	167
3.2.4	Abstrakte Datentypen	174
3.2.4.1	Das Prinzip des Abstrakten Datentyps	174
3.2.4.2	Abstrakte Datentypen Schlange und Stack	176
3.2.4.3	Abstrakter Datentyp für große Zahlen	183
3.2.4.4	Abstrakter Datentyp für komplexe Zahlen	188
4.	Semantik, Verifikation und Test	193
4.1	Konzepte für eine Semantikdefinition	194
4.1.1	Semantik: Begriff und Motivation	194
4.1.2	Grundprinzipien von Semantiknotationen	195
4.1.3	Ein Beispiel für die operationale Semantik	198
4.2	Spezifikation und Verifikation von Programmen	204
4.2.1	Vor- und Nachbedingungen	204
4.2.2	Schwächste Vorbedingungen	207
4.2.3	Die Verifikation	209
4.2.4	Beschreibung einer Schleife durch eine Invariante	215
4.2.5	Konstruktion iterativer Programme	217
4.2.6	Zusammenfassung	222
4.3	Test	224
4.3.1	Begriffsbildung und Prinzipien	224
4.3.1.1	Begriffliche Abgrenzung	224
4.3.1.2	Aufgabenteilung und Zielsetzung	225
4.3.1.3	Material und Resultate des Tests	226

4.3.2	Grenzen des Testens	227
4.3.3	Die Konstruktion von Testdaten	228
4.3.4	Zusammenfassung	237
5.	Programmierparadigmen und -sprachen	239
5.1	Programmierparadigmen	239
5.1.1	Imperatives Programmieren	243
5.1.2	Funktionales Programmieren	249
5.1.3	Logik-basiertes Programmieren	252
5.1.4	Objektorientiertes Programmieren	257
5.1.5	Regel-basiertes Programmieren	265
5.1.6	Programmierung von Mehrprozessor-Systemen	267
5.2	Übersicht über Programmiersprachen	269
6.	Datenstrukturen und Algorithmen	273
6.1	Komplexität und Effizienz	273
6.1.1	Motivation und Begriffsbildung	273
6.1.2	Effizienz und Komplexität von Algorithmen	274
6.1.3	Komplexität von Funktionen und Sprachen	279
6.2	Graphen und Bäume	283
6.2.1	Graphen	283
6.2.2	Bäume	293
6.3	Suchen in gegebenen Datenstrukturen	305
6.3.1	Suchen in Tabellen	305
6.3.2	Suchen von Zeichenketten	309
6.4	Datenorganisationen für effizientes Suchen	320
6.4.1	Suchverfahren auf Bäumen	320
6.4.1.1	Binäre Suchbäume	320
6.4.1.2	AVL-Bäume	328
6.4.1.3	Optimale Suchbäume	344
6.4.1.4	B-Bäume	349
6.4.1.5	Weitere balancierte Suchbäume	357
6.4.2	Hashing	360
6.4.2.1	Begriffsbildung und Anforderungen	360
6.4.2.2	Perfektes Hashing	362
6.4.2.3	Kollisionsbehandlung	363
6.4.2.4	Löschen in Hash-Tabellen	368
6.4.2.5	Aufwandsabschätzung	369
6.4.2.6	Implementierung von Kollisionsbehandlungen	371

6.5	Sortieren	377
6.5.1	Klassifizierung und allgemeine Betrachtungen	377
6.5.2	Interne Sortierverfahren	385
6.5.2.1	Einfache Sortierverfahren	385
6.5.2.2	Schnelle Sortierverfahren	389
6.5.2.3	Implementierung ausgewählter Sortierverfahren	399
6.5.2.4	Aufwandsvergleich der Sortierverfahren	404
6.5.2.5	Sortieren durch Streuen und Sammeln	405
6.5.3	Externe Sortierverfahren	409
6.5.3.1	Direktes Mischen	410
6.5.3.2	Natürliches Mischen	411
6.5.3.3	Mehrwege-Mischen	411
6.6	Speicherverwaltung	413
6.6.1	Algorithmische Konzepte	414
6.6.2	Implementierung von Stacks	422
Anhang A: Mathematische Grundbegriffe und Formeln		429
Anhang B: Syntaxdiagramme für Modula-2.....		432
Literatur		449
Abkürzungsverzeichnis		451
Modula-2-Index		452
Index		453