Andreas Solymosi

Objektorientiertes Plug and Play

Die Reihe „Lehrbuch“, orientiert an den Lehrinhalten des Studiums an Fachhochschulen und Universitäten, bietet didaktisch gut ausgearbeitetes Know-how nach dem State-of-the-Art des Faches für Studenten und Dozenten gleichermaßen.

In der Reihe sind u.a. erschienen:

Formalisieren und Beweisen

von Dirk Siefkes

Semantik

Theorie sequentieller und paralleler Programmierung
von Eike Best

Parallele Programmierung

von Thomas Bräunl

Konzepte und Praxis des Compilerbaus

von Volker Penner

VLSI-Entwurf eines RISC-Prozessors

Eine Einführung in das Design großer Chips
und die Hardware-Beschreibungssprache VERILOG HDL
von Ulrich Golze

Rechnerarchitektur

Analyse, Entwurf, Implementierung, Bewertung
von John L. Hennessy und David A. Patterson

Modernes Software Engineering

Eine Einführung
von Reiner Dumke

Interaktive Systeme

Software-Entwicklung und Software-Ergonomie
von Christian Stary

Objektorientiertes Plug and Play

von Andreas Solymosi

Andreas Solymosi

Objektorientiertes Plug and Play

Ein Programmierlehrbuch für Wiederverwendbarkeit
und Softwarequalität in C++

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Solymosi, Andreas:

Objektorientiertes plug and play: ein Programmierlehrbuch
für Wiederverwendbarkeit und Softwarequalität in C++ /

Andreas Solymosi. – Braunschweig; Wiesbaden: Vieweg, 1997

ISBN 978-3-528-05569-1

ISBN 978-3-528-05569-1

ISBN 978-3-322-87246-3 (eBook)

DOI 10.1007/978-3-322-87246-3

Das in diesem Buch enthaltene Programm-Material ist mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Der Autor und der Verlag übernehmen infolgedessen keine Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programm-Materials oder Teilen davon entsteht.

Alle Rechte vorbehalten

© Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1997

Der Verlag Vieweg ist ein Unternehmen der Bertelsmann Fachinformation GmbH.

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

<http://www.vieweg.de>

Druck und buchbinderische Verarbeitung: Druckerei Hubert & Co., Göttingen
Gedruckt auf säurefreiem Papier

ISBN 978-3-528-05569-1

Inhaltsverzeichnis

Vorwort	X
Danksagungen	XIV
Die Begleiddiskette	XIV
1. Informelle Einführung	1
1.1. Dualität des Programmierens	1
1.2. Passive und aktive Programmelemente	2
1.3. Das Programmierparadigma	4
1.4. Algorithmen	6
1.5. Programmiersprachen	7
1.6. Das Schichtenmodell	9
1.7. Werkzeuge des Programmierens	10
1.8. Struktur einer Programmiersprache	17
1.9. Fehlerarten	21
2. Objekte und Klassen	23
2.1. Der leere Algorithmus	23
2.2. Programmbausteine	25
2.3. Operationen	27
2.4. Algorithmen	29
2.4.1. Elementare Algorithmen	30
2.4.2. Sequentielle Algorithmen	31
2.4.3. Statisches und dynamisches Ende	32
2.5. Importprozeduren	32
2.6. Ausnahmebehandlung	34
2.7. Abstrakte Objekte	36
2.8. Datentypen	37
2.8.1. Konkrete Datenobjekte	37
2.8.2. Abstrakte Datentypen und Datenabstraktionsmodule	39
2.8.3. Anlegen von Objekten	40
2.8.4. Passende Datentypen	40
2.8.5. Duplikat eines Datentyps	41
2.9. Klassenobjekte	41
2.10. Prozeduren	42
2.10.1. Makros	42
2.10.2. Subroutinen	43
2.10.3. Inline-Subroutinen	44
2.10.4. Lokale Prozeduren	45
2.11. Schnittstellen	47
2.11.1. Modulschnittstellen	47
2.11.2. Klassenschnittstellen	50

2.12. Modulrumpfe	50
2.12.1. Die Entwicklung eines Moduls	51
2.12.2. Fremdleistung	51
2.12.3. Klassenrumpfe	53
2.13. Zusammenfassung	54
3. Werte	59
3.1. Parametrisierte Operationsaufrufe	59
3.2. Duplizieren von Inhalten	63
3.3. Prozeduren mit mehreren Parametern	65
3.4. Schachtelungen	67
3.4.1. Lokale und globale Objekte	67
3.4.2. Statische Schachtelung	68
3.4.3. Blöcke	69
3.4.4. Dynamische Schachtelung	70
3.5. Zusammenfassung	71
4. Ereignissteuerung	75
4.1. Menüs für Programmsteuerung	76
4.2. Meldungsfenster	77
4.3. Hierarchie von Bausteinen	78
4.4. Rückruf	80
4.5. Werkzeuge für Bedienerkommunikation	83
4.6. Objektwahl	87
4.7. Zusammenfassung	91
5. Aufzählungen	94
5.1. Importierte Aufzählungstypen	94
5.2. Aufzählungsobjekte	95
5.3. Definition von Aufzählungstypen	95
5.4. Logische Datentypen	98
5.5. Operationen für Aufzählungen	100
5.5.1. Mutatoren für Aufzählungsobjekte	101
5.5.2. Berechnungsfunktionen	101
5.5.3. Monadische Operatoren	102
5.5.4. Vergleichsoperationen	102
5.5.5. Vergleichsfunktionen	103
5.5.6. Vergleichsoperatoren	103
5.5.7. Diadische Operatoren	104
5.5.8. Überschreiben von Operatoren	105
5.5.9. Seiteneffekt	106
5.5.10. Ordnungsoperatoren	107
5.5.11. Logische Operatoren	107
5.6. Zeichen	109
5.7. Zusammenfassung	114

6. Verwendung von Funktionen	116
6.1. Lokale Funktionen	116
6.2. Exportierte Funktionen	118
6.3. Aufzählungsklassen	119
6.4. Implementierung von Modulen und Klassen	122
6.5. Schablonen	127
6.6. Ausdrücke	130
6.7. Textorientierte Ein- und Ausgabe	134
6.8. Zusammenfassung	136
7. Abstrakte Multibehälter	139
7.1. Mengen	140
7.2. Säcke	148
7.3. Überladen einer Methode	149
7.4. Abstrakte Multibehälter	149
7.5. Folgen	150
7.5.1. Zeichenfolgen	150
7.5.2. Arten von Folgen	151
7.5.3. Warteschlangen	152
7.5.4. Anwendung einer Klassenschablone	153
7.5.5. Privates Erben	155
7.5.6. Stapel	155
7.5.7. Positionierbare Listen	156
7.5.8. Sequentielle Dateien	157
7.5.9. Persistente Dateien	158
7.5.10. Sortierkanäle	159
7.6. Assoziativspeicher	161
7.6.1. Beispiel	161
7.6.2. Schablone mit mehreren Parametern	161
7.6.3. Verwendung von Assoziativspeichern	162
7.6.4. Direkte Dateien	163
7.6.5. Mehrfaches Erben	163
7.7. Zusammenfassung	164
8. Konglomerate	167
8.1. Felder	167
8.2. Feld als Implementierungswerkzeug	173
8.3. Verbunde	180
8.4. Verbund als Implementierungswerkzeug	184
8.5. Dateien	189
8.6. Zusammenfassung	192

9. Dynamische Datentypen	194
9.1. Zeigerobjekte und -typen	194
9.2. Lebensdauer von Objekten	196
9.3. Verwendung von Zeigern	196
9.4. Verkettete Listen als Implementierungswerkzeug	201
9.5. Zusammenfassung	206
10. Arithmetik	208
10.1. Ganzzahlbehälter	208
10.2. Ganzzahlliterale	209
10.3. Ganzzahlobjekte	210
10.4. Ganzzahloperationen	210
10.5. Weitere Ganzzahltypen	211
10.6. Ein- und Ausgabe von ganzen Zahlen	213
10.7. Zusammenfassung	214
11. Numerik	216
11.1. Abstrakte Bruchtypen	216
11.1.1. Rationale Zahlen	216
11.1.2. Gleitkomma-Dezimalbrüche	218
11.1.3. Festkomma-Dezimalbrüche	219
11.1.4. Binärbrüche	221
11.1.5. Modellzahlen	222
11.2. Konkrete Brüche	223
11.2.1. Bruchliterale	223
11.2.2. Standard-Bruchtypen in C	224
11.2.3. Operationen für Bruchtypen	224
11.2.4. Ein- und Ausgabe von Bruchwerten	224
11.3. Vektoren und Matrizen	225
11.4. Zusammenfassung	229
12. Steuerstrukturen	232
12.1. Fallunterscheidungen	232
12.1.1. Mehrweg-Alternativen	232
12.1.2. Einweg-Alternativen	233
12.1.3. Zweiweg-Alternativen	234
12.1.4. Schachtelung von Fallunterscheidungen	234
12.1.5. Lineare Algorithmen	235
12.1.6. Algorithmisierung	235
12.1.7. Kombination von Bedingungen	235
12.1.8. Erkennen einer Ausnahmesituation	236
12.1.9. Vorbeugende Ausnahmebehandlung	238
12.2. Wiederholungen	239
12.2.1. Festschleifen	240
12.2.2. Zählschleifen	241
12.2.3. Iteratorklassen	242
12.2.4. Verwendung von Zählschleifen	243

12.2.5. Endlosschleifen	245
12.2.6. Rumpfgesteuerte Schleifen	246
12.2.7. Kopfgesteuerte Schleifen	248
12.2.8. Beispiele	249
12.2.9. Dateiverarbeitung	250
12.2.10. Fußgesteuerte Schleifen	252
12.2.11. Gleichwertigkeit von Wiederholungen	252
12.3. Struktogramme	254
12.4. Sprünge	257
12.4.1. Unstrukturierte Algorithmen	257
12.4.2. Steuerstrukturfreie Programmiersprachen	259
12.5. Rekursion	261
12.6. Zusammenfassung	267
13. Vererbung und Polymorphie	270
13.1. Der Weg zur Vererbung	272
13.1.1. Die Urlösung	272
13.1.2. Mit neuer Methode ergänzen	272
13.1.3. Mit neuer Komponente ergänzen	273
13.1.4. Einkaufen	274
13.1.5. Modulvererbung	276
13.1.6. Typvererbung	277
13.1.7. Das Geheimnisprinzip	278
13.1.8. Vererbung ohne Datenkapselung	279
13.2. Polymorphie	280
13.3. Der polymorphe Stapel	288
13.4. Klassenbibliotheken	297
13.5. Zusammenfassung	299
Die Bibliothek	302
Literatur	303
Glossar	305
Sachwortverzeichnis	328

Vorwort

Die Zeiten des Programmierens sind vorbei. Ebenso wie die Zeiten des Lötens.

Während vor einem Jahrzehnt Informatiker mit Lötkolben und Spannungsmeßgeräten ausgerüstet werden mußten, um zwei Rechner miteinander zu verbinden, ermöglicht heute die „plug-and-play“¹-Technologie das Verbinden standardisierter Hardwarekomponenten im Handumdrehen. Vereinheitlichte Schnittstellen und Stecker machen die Kenntnisse über die internen Vorgänge überflüssig.

Vor 10 Jahren hatten auch Informatiker mit intensiven Programmierkenntnissen durchaus gute Berufschancen; heute sind solche viel weniger gefragt. Nur ein kleiner Teil der eingestellten Computerspezialisten beschäftigt sich mit der Formulierung neuer Algorithmen. Die überwiegende Anzahl wird eingesetzt, um vorhandene Hard- und Softwarebausteine² zusammenzustecken und den Bedürfnissen des Arbeitgebers anzupassen. Sie müssen das Innenleben dieser Bausteine nicht kennen - sie könnten es gar nicht in Anbetracht deren großer Anzahl und hoher Komplexität. Sie betrachten nur ihr Äußeres, eine Art Bedienungsanleitung, ihre Schnittstelle zur Umgebung. Sie müssen verstehen, welche Teile dieser Schnittstelle für die gegebene Aufgabe relevant sind und wie sie mit anderen Bausteinen zusammengefügt werden kann. Das traditionelle Programmieren von Verzweigungen und Schleifen, Bits und Bytes, Integers und Characters kommt nur am Rande vor.

Diese Erfahrung ist die Quelle des Konzepts, das diesem Lehrbuch zugrunde liegt, Programmiersprachen anders als bisher zu unterrichten: Im Mittelpunkt des *Objekt-orientierten „plug and play“* sollen nicht mehr die sprachlichen Strukturen, sondern das Zusammenfügen vorhandener Programmbausteine stehen. Dies bedeutet einerseits, daß der Lehrling zu Beginn mit den „höheren“ Sprachkonzepten (wie *Module*, *Klassen*, *generische Einheiten*, usw.) konfrontiert wird. Also lernt man die Sprache „von oben“. Während man konventionell überwiegend mit Hilfe der „niedrigen“ Sprachelemente (wie *Bits*, *Bytes*, *Ganzzahlen*, *Verzweigungen*, *Schleifen*) programmiert, fördert das *Programmieren von oben* das abstrakte Denken durch die Benutzung der Schnittstellen der Bausteine, ohne ihre Interna zu kennen.

Auf der niedrigen Ebene sollte man also nur dann programmieren, wenn keine geeigneten Bausteine vorhanden sind. Somit entspricht es durchaus dem gängigen Trend der modernen Softwareentwicklung: Das *Programmieren im Kleinen* sollte demnach reduziert und das *Programmieren im Großen* betont werden. Das projektübergreifende Programmieren - man könnte es *Programmieren im sehr Großen* nennen - steckt heute noch in Kinderschuhen, auch wenn die Entwicklung in diese Richtung (z.B. *OLE*, *Frameworks*, usw.) absehbar ist.

¹ auf deutsch etwa: *einstecken und loslegen*

² sog. Standardlösungen

Als Basis für diese Vorgehensweise wurde die Sprache C++ ausgewählt. Die Ursache hierfür ist, daß ihre Vorgängerin C eine der populärsten Programmiersprachen ist. Ein Grund für ihre Verbreitung mag ihre Flexibilität sein. Der Programmierer hat mehr Möglichkeiten, seine kreativen Ideen zum Ausdruck zu bringen, als bei den meisten anderen Sprachen. Die in C geschriebenen Programme sind oft fast genauso schnell, als ob sie auf einer maschinennahen Sprache geschrieben worden wären. Weil C eine zeichenorientierte Sprache ist, schreibt der erfahrene Entwickler seine Programme sehr schnell. Die Sprache legt fast keine Lasten auf den Programmierer, daher gibt es viele Erfolgserlebnisse; es ist sehr angenehm und macht Spaß, in C Programme zu schreiben.

Dies hat allerdings auch unangenehme Nebeneffekte. Weil der Kreativität des Programmierers kaum Grenzen gesetzt werden, ist es auch sehr leicht, in C Fehler zu machen. Leistungsfähige und erfahrene Systementwickler kommen damit relativ gut zurecht, aber die Qualität der erstellten Software leidet oft daran, daß die Sprache zu wenig Einschränkungen beinhaltet.

Diese Erfahrungen haben zu Entwicklungen anderer Sprachen wie *Ada* geführt, die mehr Programmierdisziplin erzwingen. Weil es aber so unbequem ist, diszipliniert zu programmieren, haben sie sich kaum durchgesetzt. Wenn überhaupt, dann dort, wo die Bedeutung der Disziplin bekannt ist: beim Militär oder in sicherheitsrelevanten Aufgabenbereichen wie der Flug- oder Atomindustrie.

C++, die Weiterentwicklung von C, stellt eine Art Zwischenlösung dar. Einerseits behält sie, als Obermenge von C, ihre ganze Flexibilität. Andererseits ermöglicht sie denen, die qualitative Software erstellen möchten, nur die zuverlässigen Sprachelemente zu benutzen. Vermutlich nimmt C++ auf der Popularitätsskala deswegen nach und nach den Platz von C ein.

Leider stellt so die Sprache das theoretisch mögliche Qualitätsminimum immer noch nicht sicher; es liegt nach wie vor im Ermessen des Programmierers, wie weit er die neuen Möglichkeiten für Einschränkungen nutzt oder sich die vollständige Kontrolle über seine eigene Freiheit vorbehält. Somit bleibt für zukünftige C++-Programmierer die Erziehung zur Selbstdisziplin im Rahmen ihrer Ausbildung die einzige Chance.

In diesem Lehrbuch wird der Versuch unternommen, C++ auf unkonventionelle Weise zu unterrichten. Es werden weniger die Möglichkeiten der Sprache vorgestellt, vielmehr eine - nach Meinung des Autors vernünftige - Art von Programmierstil und -denken, sowie der Weg, wie dies in C++ ausgedrückt werden kann. So können dem Studenten schon zu Anfang Konzepte offensichtlich werden, die sich andere erst als Fortgeschrittene oder gar nicht aneignen. Die Sprachelemente, die traditionell am Anfang des Unterrichts eingeführt werden, werden erst in den hinteren Kapiteln behandelt. Manche, die - nach Meinung des Autors - der Programmierdisziplin schaden, werden übergangen.

Daher ist dies kein Lehrbuch für die Sprache C++. Es ist vielmehr ein Lehrbuch für Programmieren - und zwar nach dem Prinzip „*plug and play*“: das Programmieren wird „*von oben*“ gelernt.

Dieses Lehrbuch wird in Informatikstudiengängen von Hochschulen¹, an denen das Fach *Softwaretechnologie* einen Schwerpunkt bildet, zunehmend mit Erfolg eingesetzt. Die übliche Reihenfolge wird also umgekehrt, und zuerst werden die „höheren Konzepte“ (ohne die einfacheren Sprachelemente) unterrichtet. Die Einführung von Fallunterscheidungen, Wiederholungen und sogar der Ganzzahltypen wird so weit wie möglich hinausgeschoben. Dafür werden Module, abstrakte Datentypen, Klassen und Schablonen gleich zu Anfang benutzt, ohne dabei zu erläutern, was sich hinter ihnen versteckt. Hierdurch soll das *abstrakte Denken* gefördert werden: Der Gebrauch von vorhandenen Programmbausteinen ist auch ohne die Kenntnisse ihrer Interna möglich. Professionelle Werkzeugkästen (wie etwa eine Unix-Toolbox) unterstützen diese Philosophie.

Der Ansatz zur *Vererbungsprogrammierung* ist dadurch naturgemäß. Die in C++ neu eingeführten Konzepte für das *objektorientierte Programmieren* werden dabei genutzt; diese führen zum Konzept der *Wiederverwendbarkeit* beim Programmieren. Es wird angestrebt, daß sich die Studenten - im Gegensatz zur üblichen Praxis - daran gewöhnen, zuerst nach vorhandenen Bausteinen zu suchen, und erst wenn sie nichts finden, selber zu programmieren.

Das vorgestellte Unterrichtskonzept und das Programmierparadigma sind an sich sprachunabhängig. Ein ähnliches Lehrbuch existiert für die Sprache *Ada*² und *Pascal*; andere Sprachen wie *Oberon* und *Java* sind in Vorbereitung. Die *Objektorientierung* ist dabei eine Voraussetzung, auch wenn die Chancen der *Vererbung* und *Polymorphie* erst zum Schluß als Höhepunkt wirklich ausgenutzt werden. Nichtsdestotrotz wird der Hauptgedanke des objektorientierten Programmierens, nämlich die *Wiederverwendbarkeit*, von Anfang an stark betont.

Das Lehrbuch vermittelt keine Fähigkeiten zur Algorithmisierung; komplexe Ablaufvorgänge müssen in anderen Rahmen erlernt werden. Ebenfalls werden kaum Kenntnisse zum Entwurf einer Modul- oder Klassenhierarchie unterrichtet: Die Schnittstellen werden als gegeben vorausgesetzt. Es wird nur das Handwerkszeug vorgestellt, wie vorhandene Bausteine zusammengefügt werden können.

Eine Voraussetzung für die erfolgreiche Arbeit mit dem Lehrbuch ist das Vorhandensein eines Rechners mit einer C++-Entwicklungsumgebung, auf dem die Übungen durchgeführt werden können. Der C++-Compiler soll die *Ausnahmebehandlung* nach der ANSI-Spezifikation (*try*, *catch*, *throw*) übersetzen können.

Zu diesem Buch gehört eine *Begleitdiskette*. Sie enthält einige im Text aufgeführte Beispielprogramme und Musterlösungen für die Übungsaufgaben. Darüber hinaus enthält sie den Satz von Programmbausteinen sowie Dienstprogramme, die die vorgestellten Konzepte implementieren. Anweisungen für ihre Anwendung wurden in der Datei INFO.TXT aufgelistet.

¹ so auch an der *Technischen Fachhochschule Berlin*

² s. [Sol] im Literaturverzeichnis

Die im Lehrbuch aufgeführten Beispielprogramme sind mit dem Compiler *Borland C++* Version 4.5 ausgetestet worden.

Abgesehen von eventuell auftretenden Installationsproblemen des Compilers und der Bibliothek auf der Begleitdiskette ist das Buch auch fürs Selbststudium geeignet. Kenntnisse in *C++* oder in anderen Programmiersprachen stellen zwar Hindernisse dar, sie können aber bewußt überwunden werden.

Die Terminologie des Lehrbuches ist selbstgewählt; sie entspricht nur stellenweise dem üblichen Sprachgebrauch. Auf die Abweichungen wird jedoch immer hingewiesen. In den Erläuterungen wird von der Sprache *C* bei Konzepten gesprochen, die sowohl hier wie auch in *C++* Gültigkeit haben. Hinweise auf die Sprache *C++* gelten jedoch für *C* nicht.

Danksagungen

Für die Hilfe, die ich während des Verfassens dieses Lehrbuchs erhalten habe, bin ich meiner Kollegin *Prof. Dr. D. Weber-Wulff* sowie meinen Kollegen *Prof. Dr. W. Brecht*, *Prof. Dr. U. Grude* und *Prof. Chr. Knabe* von der *Technischen Fachhochschule Berlin*, sowie meinem ehemaligen Promotionsbetreuer *Prof. Dr. H.-J. Schneider* von der *Universität Erlangen-Nürnberg* sehr dankbar, die Teile meiner Entwürfe durchgelesen und durch ihre zahlreichen Bemerkungen zur inhaltlichen Qualität wesentlich beigetragen haben. Meine Diplomanden *O. Castro-Sandoval* und *Y. Azami* haben einen Teil der Arbeit übernommen, indem sie meine Beispielprogramme ergänzt und ausgetestet, sowie die Begleitdiskette zusammengestellt haben. Mein Bruder *P. Solymosi* hat wichtige fachliche und sprachliche Verbesserungsvorschläge gemacht. Nicht zuletzt danke ich meiner Frau *Dr. I. Solymosi*, die nicht nur die Belastungen meiner Arbeit neben 4 kleinen Kindern mitgetragen, sondern viele meiner Rechtschreib- und Deutschfehler beseitigt, d.h. meinen Entwurf Korrektur gelesen hat. Unser gemeinsamer Glaube an *Jesus Christus* gab ihr die Kraft dazu.

Der Autor

Die Begleitdiskette

Die für die Arbeit mit diesem Lehrbuch notwendige 3½"-Begleitdiskette kann über folgende Adresse bestellt werden:

AP SIS GmbH, D-16727 Schwante

Der Preis von DM / sFr / US\$ 10,- bzw. ÖSch 100,- soll als Geldschein oder als Scheck eines deutschen Bankinstituts beigelegt werden. Alternativ reicht eine Bankgutschrift von DM 10,- mit Verwendungszweck „Begleitdiskette“ und Postadresse an das Konto:

AP SIS GmbH, Kto-Nr. 6650177604 bei der Hypobank Weilheim (BLZ 70320305)

Einfacher ist es aber, den Inhalt der Begleitdiskette kostenlos über das Internet an der folgenden Adresse abzuholen:

<http://www.tfh-berlin.de/~oo-plug>

Hier sind aktuelle Änderungen, Korrekturen und Programmversionen in anderen Sprachen erreichbar.

Für Hinweise, Kritik und gefundene Fehler ist der Autor über die elektronische Postadresse

oo-plug@tfh-berlin.de oder solymosi@tfh-berlin.de

dankbar. Hier können auch Fragen zum Lehrbuch gestellt werden.