Vorwort

In den vergangenen Jahren wurde die *Programmierung von Rechenanlagen* als diejenige Disziplin erkannt, deren Beherrschung grundlegend und entscheidend für den Erfolg vieler Entwicklungsprojekte ist und die wissenschaftlicher Behandlung und Darlegung zugänglich ist. Vom Handwerk stieg sie zur akademischen Disziplin auf. Die ersten hervorragenden Beiträge zu dieser Entwicklung wurden von E.W. Dijkstra und C.A.R. Hoare geliefert. Dijkstra's *Notes on Structured Programming* [6] führten zu einer neuen Betrachtung der Programmierung als wissenschaftliches Thema und als intellektuelle Herausforderung. Sie bahnten den Weg zu einer "Revolution" in der Programmierung [35]. Hoare's *An Axiomatic Basis for Computer Programming* [10] zeigte deutlich, dass Progamme einer exakten Analyse nach mathematischen Grundsätzen zugänglich sind. Beide Artikel argumentieren überzeugend, dass viele Programmierfehler vermieden werden können, wenn man den Programmierern die Methoden und Techniken, die sie bisher intuitiv und oft unbewusst verwendeten, zur Kenntnis bringt. Diese Artikel konzentrierten sich auf die Aspekte des Aufbauens und Analysierens von Programmen oder, genauer ausgedrückt, auf die Struktur der durch die Progammtexte dargestellten *Algorithmen*. Es ist jedoch völlig klar, dass ein systematisches und wissenschaftliches Angehen der Konstruktion von Programmen seine grösste Wirkung im Fall grosser komplexer Programme hat, die komplizierte Datenmengen bearbeiten. Folglich muss eine Methodik des Programmierens auch alle Aspekte der Datenstrukturierung behandeln. *Programme* sind letztlich konkrete Formulierungen abstrakter Algorithmen, die sich auf bestimmte Darstellungen und *Datenstrukturen* stützen. Einen wesentlichen Beitrag, Ordnung in die verwirrende Vielfalt der Terminologie und Konzepte von Datenstrukturen zu bringen, leistete Hoare durch seine *Notes on Data Structuring* [13]. Es wurde klar, dass Entscheidungen über die Strukturierung der Daten nicht ohne Kenntnis der auf die Daten anzuwendenden Algorithmen getroffen werden können, und dass umgekehrt die Struktur und Wahl der Algorithmen oft stark von den zugrunde liegenden Daten abhängt. Kurz gesagt: Programmerstellung und Datenstrukturierung sind untrennbar ineinandergreifende Themen.

Dennoch beginnt dieses Buch mit einem Kapitel über Datenstrukuren. Dafür

gibt es zwei Gründe. Erstens hat man das intuitive Gefühl, dass Daten den Algorithmen vorangehen; man muss Objekte haben, bevor man Operationen auf sie anwenden kann. Zweitens - und das ist der unmittelbare Grund - geht dieses Buch davon aus, dass der Leser mit den grundlegenden Begriffen des Programmierens von Rechenanlagen vertraut ist. Einführende Programmierkurse konzentrieren sich jedoch traditionell und vernünftigerweise auf Algorithmen, die auf einfachen Datenelementen operieren. Somit scheint ein einführendes Kapitel über Datenstrukturen angebracht.

Im ganzen Buch und besonders im ersten Kapitel folgen wir der von Hoare [13] dargelegten Theorie und Terminologie, die in der Programmiersprache *Pascal* realisiert wurde [13]. Das Wesentliche dieser Theorie ist, dass Daten zuerst Abstraktionen realer Phänomene darstellen und vorzugsweise als abstrakte, in üblichen Programmiersprachen nicht notwendigerweise vorhandene Strukturen ausgedrückt werden. Im Prozess der Programmentwicklung wird die Darstellung der Daten schrittweise verfeinert - parallel zur Verfeinerung der Algorithmen - um sich mehr und mehr den durch ein verfügbares Programmiersystem gegebenen Möglichkeiten anzupassen [32]. Wir setzen daher eine Anzahl grundlegender Prinzipien zum Aufbau von Datenstrukturen voraus, die sogenannten *fundamentalen Strukturen*. Es ist sehr wichtig, dass diese Konstruktionen auf wirklichen Rechenanlagen leicht zu implementieren sind. Denn nur in diesem Fall können sie als wirkliche Elemente einer Datendarstellung betrachtet werden, als die aus dem letzten Vefeinerungschritt hervorgehenden "Moleküle" der Datenbeschreibung. Es sind dies der Verbund (record), das Feld (array) und die Menge (set). Es überrascht nicht, dass diese grundlegenden Aufbauprinzipien fundamentalen mathematischen Begriffen entsprechen.

Ein Grundprinzip dieser Theorie der Datenstrukturen ist die Unterscheidung zwischen fundamentalen und "höheren" Strukturen. Erstere sind die Moleküle selbst aus Atomen aufgebaut - und dienen als Komponenten für die letzteren. Variablen einer fundamentalen Struktur ändern nur ihren Wert, aber niemals die Menge der Werte, die sie annehmen können. Folglich bleibt die Grösse des Speichers, den sie belegen, konstant. Variablen höherer Strukturen sind hingegen charakterisiert durch die Veränderung ihres Wertes *und* ihrer Struktur während der Ausführung eines Progamms. Für ihre Implementation werden daher aufwendigere Techniken benötigt.

Die sequentielle Datei - einfach File oder Sequenz genannt - erscheint in dieser Klassifikation als ein Zwitter. Sie ändert zwar ihre Länge; diese Änderung der Struktur ist aber trivialer Natur. Da das File eine wirklich grundlegende Rolle in praktisch allen Rechenanlagen spielt, wird es mit den fundamentalen Strukturen in Kapitel 1 behandelt.

Das zweite Kapitel behandelt *Sortier-Algorithmen*. Es zeigt eine Vielfalt verschiedener Methoden, die alle dem gleichen Zweck dienen. Mathematische

Analysen einiger dieser Algorithmen zeigen die Vor- und Nachteile und bringen dem Programmierer die Wichtigkeit der Analyse bei der Wahl einer guten Lösung für ein gegebenes Problem zum Bewusstsein. Die Aufteilung in Methoden zum Sortieren von Arrays und Methoden zum Sortieren von Files (oft internes, resp. externes Sortieren genannt) zeigt den entscheidenden Einfluss der Darstellung der Daten auf die Wahl anwendbarer Algorithmen und auf ihre Komplexität. Dem Sortieren würde nicht soviel Platz gewidmet, wenn es nicht ein ideales Instrument für die Darlegung so vieler Prinzipien der Programmierung und von Situationen wäre, die auch in vielen anderen Anwendungen auftreten. Es scheint oft, dass man einen ganzen Programmierkurs auf Sortierbeispielen aufbauen könnte.

Ein anderes Thema, das gewöhnlich in Einführungskursen keinen Platz findet, spielt in der Konzeption vieler algorithmischer Lösungen eine wichtige Rolle, nämlich die Rekursion. Das dritte Kapitel ist deshalb *rekursiven Algorithmen* gewidmet. Es wird gezeigt, dass Rekursion eine Verallgemeinerung der Wiederholung (Iteration) ist. Als solche ist sie ein wichtiges und umfassendes Konzept der Programmierung. In vielen Programmierkursen wird Rekursion bloss an Beispielen gezeigt, die mit Iteration besser gelöst werden. Statt dessen konzentriert sich Kapitel 3 auf mehrere Beispiele von Problemen, in denen Rekursion eine äusserst natürliche Formulierung einer Lösung erlaubt, während die Verwendung von Iteration zu undurchsichtigen und schwerfälligen Programmen führen würde. Die Klasse der *Backtracking-Algorithmen* erweist sich als ideale Anwendung der Rekursion, aber die offensichtlichsten Kandidaten für die Verwendung der Rekursion sind Algorithmen, die auf rekursiv definierten Datenstrukturen operieren. Diese Fälle werden im letzten Kapitel behandelt, wozu das dritte Kapitel eine gute Grundlage bildet.

Kapitel 4 behandelt *dynamische Datenstrukturen;* das sind Datenstrukturen, die sich während der Ausführung des Programms ändern. Es wird gezeigt, dass rekursive Datenstrukturen eine wichtige Unterklasse der allgemein verwendeten dynamischen Strukturen sind. Obwohl eine unmittelbar rekursive Definition in diesen Fällen sowohl natürlich als auch möglich ist, wird sie gewöhnlich in der Praxis doch nicht verwendet. Statt dessen wird der zu ihrer Implementation verwendete Mechanismus dem Programmierer bewusst gemacht, indem man ihn zwingt, explizite Referenz- oder Zeiger-Variablen zu verwenden. Das vorliegende Buch folgt dieser Praxis und spiegelt den gegenwärtigen Stand der Technik wieder: Kapitel 4 ist der Programmierung mit Zeigern, Listen, Bäumen und Beispielen mit komplexeren Datengebilden gewidmet. Es zeigt die (etwas unpassend) so genannte "Listenverarbeitung". Ein beträchtlicher Teil behandelt die Organisation von Bäumen und besonders das Durchsuchen von Bäumen. Das Kapitel schliesst mit einer Darstellung der gestreuten Speicherung, auch "hash"-Code genannt, die oft Suchbäumen vorgezogen wird. Damit bietet sich eine Möglichkeit zum Vergleich zweier fundamental verschiedener Techniken für eine häufig vorkommende Anwendung.

Programmieren ist eine *konstruktive* Tätigkeit. Wie kann eine aufbauende, schöpferische Fähigkeit gelehrt werden? Eine Möglichkeit besteht im Herauskristallisieren elementarer Konstruktionsgesetze aus vielen Anwendungen und ihrer systematischen Darstellung. Aber Programmieren ist ein weites und vielfältiges Gebiet, das oft komplexe geistige Tätigkeit erfordert. Die Vorstellung, es je zu einem reinen "Unterrichten von Rezepten" zusammenfassen zu können, ist verfehlt. Aus unserem Arsenal von Unterrichtsmethoden bleibt nur die sorgfältige Auswahl und Darstellung von Musterbeispielen. Natürlich sollten wir nicht glauben, dass jedermann gleich viel aus dem Studium von Beispielen lernt. Wesentlich bei diesem Vorgehen ist, das vieles dem Studenten, seinem Fleiss und seiner Intuition überlassen bleiben muss. Dies gilt ganz besonders für relativ schwierige und lange Programmbeispiele. Ihr Einschluss in dieses Buch geschah nicht zufällig. Längere Programme sind in der Praxis der Normalfall und eignen sich eher dafür, die schwer zu fassende aber wesentliche Zutat, genannt Stil und methodische Struktur, aufzuzeigen. Sie sollen auch als Übung für das Lesen von Programmen dienen, das neben dem Schreiben zu oft vernachlässigt wird. Dies ist ein wichtiger Grund für den Einschluss grösserer vollständiger Programme als Beispiele. Der Leser wird durch eine allmähliche Entwicklung des Progamms geführt; es werden ihm mehere "Schnappschüsse" der Enstehung des Programms gezeigt, wobei sich diese Entwicklung als *schrittweise Verfeinerung* der Einzelheiten erweist. Ich halte es für wichtig, dass Programme in einer letzten Form unter hinreichender Berücksichtigung der Einzelheiten gezeigt werden, denn der Teufel steckt beim Programmieren im Detail. Obwohl eine Beschränkung der Ausführung auf das Prinzip eines Algorithmus und seine mathematische Analyse unter Ausschluss technisch bedingter Details für einen akademischen Geist anregend und herausfordernd wirken kann, vermag dieses Vorgehen den Praktiker nicht zu befriedigen. Ich habe mich daher strikt an die Regel gehalten, die Programme zum Schluss in einer Sprache anzugeben, in der sie von einer Rechenanlage direkt ausgeführt werden können.

Damit stellt sich natürlich das Problem, eine Form zu finden, die durch eine Rechenanlage ausführbar ist und gleichzeitig doch soweit maschinenunabhängig bleibt, wie es von einem Lehrbuch gefordert werden muss. In dieser Beziehung erwiesen sich weder weitverbreitete Sprachen noch abstrakte Notationen als geeignet. Die Sprache Pascal hingegen stellt einen guten Kompromiss dar; sie wurde mit genau diesem Ziel entwickelt und daher auch in diesem Buch verwendet. Die Programme können von Programmierern leicht verstanden werden, die mit irgendeiner anderen höheren Programmiersprache, wie ALGOL 60 oder PL/I vertraut sind. Dies heisst aber nicht, dass eine vorangehende Einführung nicht von Nutzen wäre. Das Buch "Systematisches Programmieren" [33] vermittelt eine ideale Grundlage, da es sich auch auf die Pascal- Notation stützt. Das vorliegende Werk ist nicht als Handbuch für die Sprache Pascal gedacht; zu diesem Zweck sei der Leser auf die einschlägige Literatur verwiesen [15].

Dieses Buch ist eine Zusammenfassung - und gleichzeitig eine Ausarbeitung verschiedener Programmierkurse, die an der Eidgenössischen Technischen Hochschule (ETH) Zürich gehalten wurden. Ich verdanke viele in diesem Buch dargelegten Ideen und Ansichten Diskussionen mit meinen Mitarbeitern an der ETH. Zu erwähnen sind auch der stimulierende Einfluss der Treffen der IFIP Arbeitsgruppen 2.1 und 2.3 und besonders die bei diesen Gelegenheiten geführten Gespräche mit E.W. Dijkstra und C.A.R. Hoare. Zum Schluss möchte ich nicht vergessen, der ETH für die grosszügige Bereitstellung der Rechenmöglichkeiten zu danken, ohne welche die Ausarbeitungen dieses Lehrbuchs unmöglich gewesen wäre. Ich möchte all denen, die direkt oder indirekt zu diesem Buch beigetragen haben, meinen herzlichen Dank aussprechen.

Der vorliegende Text wurde mit Hilfe des Computers Lilith erstellt und editiert, wobei der ISO Zeichensatz verwendet wurde. Da somit weder Höher- noch Tieferstellen zur Verfügung steht, wird zur Angabe der Exponentiation das Symbol ** verwendet, und Indizes werden in eckige Klammern gesetzt.

Sowohl die Übersetzung aus dem englischen Originaltext als auch die computertechnische Herstellung des druckreifen Textes sind das Werk von Dr. H. Sandmayr. Ihm gebührt für seine umfangreiche Arbeit mein besonderer Dank.

Zürich, im Sommer 1975 N. Wirth

Vorwort zur 3. Auflage

Die vorliegende dritte Auflage unterscheidet sich inhaltlich von den vorangegangenen kaum. Hingegen widerspiegelt sie den in den letzten acht Jahren erzielten Fortschritt in der durch Computer unterstützten Textverarbeitung deutlich. Anstelle einer einfachen Formatierung und Ausgabe mit konventioneller Schreibmaschine tritt ein System, das verschiedene Proportional-Schriften zu verwenden gestattet und diese mithilfe eines Laser-Druckers zu Papier bringt. Die Formatier- und Druckerprogramme wurden vom Autor selbst angefertigt und sind mit dem Arbeitsplatzrechner Lilith implementiert. Gewisse Mängel, die dem System noch anhaften, möge der Leser mit Nachsicht behandeln. Bei Gelegenheit dieser Neuausgabe sind auch zahlreiche kleine Fehler und Inkonsistenzen korrigiert worden. Ferner wurde die sprachliche Fassung mancherorts verbessert, was zur leichteren Verständlichkeit beitragen möge. Ich danke an dieser Stelle allen, die zur Korrektur beigetragen haben, insbesondere aber Dr. J. Gutknecht für seine zahlreichen konkreten Vorschläge und für seine gewissenhafte Durchsicht der Neufassung.

Zürich, im Frühjahr 1983 N.W.