Einführung in die maschinennahe, imperative, funktionale, relationale und objektorientierte Programmierung

-

EMIFROP 0.24

Markus Alpers
B.Sc. und Ausbilder f. Industriekaufleute

29. Februar 2016

Inhaltsverzeichnis

Hinweis bezüglich diskriminierender Formulierungen

In diesem Text wurde darauf geachtet Formulierungen zu vermeiden, die diskriminierend verstanden werden können. Im Sinne der Lesbarkeit wurden dabei Formulierungen wie „Informatiker und Informatikerinnen“ durch „InformatikerInnen“ (mit großem i) ersetzt. An anderen Stellen habe ich Formen wie eine/einer durch eineR zusammengefasst. Hier berufe ich mich auf den Artikel „Sprache und Ungleichheit“ der Bundeszentrale für politische Bildung, kurz BpB, vom 16. April 2014, insbesondere auf den Absatz „Zum Umgang mit diskriminierender Sprache“, online abrufbar unter:

`http://www.bpb.de/apuz/130411/sprache-und-ungleichheit?p=all`

Sollten Sie dennoch Formulierungen entdecken, die diesem Anspruch nicht entsprechen, möchte ich Sie bitten, mir eine entsprechende Nachricht zu senden, denn es ist mir wichtig, Ihnen mit diesem Buch eine wertvolle Unterstützung beim Start in die faszinierende Welt der Informatik zu bieten. Das sollte nicht durch verletzte Gefühle in Folge missverständlicher Formulierungen torpediert werden.

Sie erreichen mich unter `markus.alpers@haw-hamburg.de`.

Hinweis zur Lizenz

zenz"´. Wenn Sie einen Teil finden, in dem diese „Hinweise zur Lizenz"´ nicht zu finden ist, dann dürfen Sie für den persönlichen Gebrauch dennoch Kopien davon anfertigen und Sie dürfen diese Kopien außerhalb von kommerziellen Projekten frei verwenden.

Hinweis zur Verwendbarkeit in wissenschaftlichen Arbeiten

Bitte beachten Sie dabei aber, dass die Verwendung dieses Textes im Rahmen wissenschaftlicher Publikationen zurzeit aus anderen Gründen problematisch ist: Wie viele andere Quellen, die frei im Internet verfügbar sind, wurde auch dieser Text bislang nicht durch einen nachweislich entsprechend qualifizierten Lektor verifiziert. Damit genügen Zitate aus diesem Band streng genommen noch nicht den Ansprüchen wissenschaftlicher Arbeiten.

Bitte beachten Sie außerdem, dass dieses Buch eine Konvention nutzt, die in wissenschaftlichen Arbeiten verpönt ist: Wenn in einer wissenschaftlichen Arbeit ein Begriff hervorgehoben wird, dann wird dazu kursive Schrift verwendet. In diesem Buch verwende ich dagegen Fettdruck, da es vielen Menschen schwer fällt, einen kursiv gedruckten Begriff schnell zu finden und ich mir wünsche, dass Sie es möglichst effizient auch als Nachschlagewerk nutzen können.

Teil I

Einführung in die Programmierung für alle Studierenden im Bereich MINT

(Mathematik, Ingenieurswissenschaften, Naturwissenschaften, Technik)

Kapitel 1

Typische Irrtümer darüber, was Programmieren ist.

Kapitel 2

Von der Nachrichtentechnik zur Programmierung

Computer sind eine Kombination aus Bauteilen und Datenübertragungsleitungen. In vielen Programmiersprachen brauchen ProgrammiererInnen sich nicht direkt um die Datenübertragung zu kümmern. Allerdings folgen aus der Datenübertragung einige Fakten, die wir bei der Programmierung in jeder imperativen Programmiersprache beachten müssen. Tun wir das nicht, dann sind Fehler die Folge, die wir ohne ein allgemeines Verständnis der Grundlagen von Datenübertragungen nicht verstehen und damit lösen können.

2.1 Nachrichtentechnik – So kommen Nullen und Einsen in den Rechner.

Sie haben schon öfter gehört, dass ein Computer im Kern mit Nullen und Einsen arbeitet. Was den ein oder anderen vielleicht zu Spekulationen über die Qualität dieser Geräte gebracht haben könnte... Wer arbeitet schon freiwillig einen Großteil seiner Zeit mit Nullen zusammen?

Was Sie aber in aller Regel in einem Informatikstudium nicht hören, ist dass diese Folgen von Einsen und Nullen nur ein Hilfsmittel sind, eine **Repräsentation** dessen, was tatsächlich vorhanden ist. Doch wenn Sie etwas darüber hören, dann wird Ihnen in aller Regel erzählt, dass damit der Unterschied zwischen fließendem oder nicht fließendem Strom dargestellt wird. Das ist so aber in aller Regel falsch: Es gibt die unterschiedlichsten Formen von Datenübertragungen, deren Signale am Ende als Folgen von Nullen und Einsen **interpretiert** werden. Und nur die am einfachsten zu verstehende Form ist die, bei der Signale erzeugt werden, indem zwischen fließendem und nicht-fließendem Strom unterschieden wird. Dieje-

nigen von Ihnen, die Medientechnik studieren werden sich damit wenigstens fünf Semester lang beschäftigen, auch wenn Ihnen das anfangs also in Veranstaltungen wie Elektrotechnik 1 und 2 gar nicht bewusst sein wird.

Es folgen zwei Beispiele dafür, wie Einsen und Nullen auch anders dargestellt werden können:

- Am Karlsruher Institut für Technologie (KIT[1]) wird an der Möglichkeit geforscht, wie man Daten in Molekülen aus Lachs DNA speichern kann. Hier wurden Nullen mit 0,4 V und Einsen mit 0,9 V repräsentiert.
- Im Bereich der Speicherung mit DNA gibt es aber auch andere Ansätze. Wie Sie aus dem Biologieunterricht wissen, besteht DNA aus vier Molekülen, die als sogenannte Basensequenzen endlose Ketten bilden können. Und richtig, auch diese Basensequenzen kann man nutzen, um damit Einsen und Nullen darzustellen. Hier steht also gar kein Stromfluss für Nullen und für Einsen, weil sie chemisch und nicht elektrisch repräsentiert werden. DNA bietet dabei eine derart hohe Dichte pro Bit an, dass Sie den Inhalt von 50 Millionen BlueRay Disks in einer Kaffeetasse unterbringen können[2]. Na gut, vielleicht war es auch ein Übersetzungsfehler und im Originalartikel war die Rede von einem Kaffeebecher, aber spielt das eine Rolle? Denn das bedeutet, dass Sie die gesamte Videothek der Welt in einem Kaffeebecher herumtragen könnten. Und überlegen Sie jetzt, wie viel Raum eine solche Videosammlung in Form von BlueRays oder Festplatten einnehmen würde.

Die wissenschaftliche Disziplin, die sich unter anderem mit der Frage beschäftigt, wie man Nullen und Einsen übertragen kann, nennt sich **Nachrichten- und Kommunikationstechnik** und ist ein Teilbereich der **Elektrotechnik**. Während es bei der Datenübertragung innerhalb eines Computers noch recht problemlos möglich ist, mittels Ein- und Ausschalten von Stromflüssen Signale zu erzeugen und zu interpretieren, ist das bei Übertragungen über längere Distanzen eine eher ineffiziente Methode. Deshalb werden hier Schwingungen manipuliert, um die Datenübertragung zu realisieren. Das wichtigste mathematische Werkzeug ist dabei die **Fourier-Transformation** (kurz **FT**). Diese transformiert Nullen und Einsen in eine Schwingung, indem Winkelfunktionen auf eine bestimmte Weise beliebig häufig angewendet werden. (Je häufiger, desto präziser.)

In der Nachrichtentechnik spielen dann Begriffe und Ansätze eine Rolle, die kaum etwas mit dem gemein haben, was die Tätigkeit von InformatikerInnen bestimmt. Während die **Media Systems** Studierenen sich also vor-

[1] `http://www.kit.edu`

[2] `http://www.spektrum.de/news/auf-petabyte-pro-gramm/1182773`

rangig damit beschäftigen Gesamtaufgaben in Teilaufgaben zu unterteilen und mittels abstrakter Konzepte die Teilaufgaben möglichst übersichtlich zu gestalten und sie dabei möglichst effizient zu lösen, nutzen **Medien- und NachrichtentechnikerInnen** Formeln und Funktionen, um Daten so effizient wie möglich über einen bestimmten Leitungstyp zu übertragen.

Kontrolle
Die Nachrichtentechnik entwickelt die Systeme, die es uns überhaupt erst ermöglichen, Informatik zu betreiben. Einsen und Nullen sind zwar die Basis der IT-Systeme, die wir programmieren, was aber tatsächlich bei der Datenübertragung im Computer oder im Internet passiert, um Einsen und Nullen zu übertragen oder besser gesagt um eine jeweils passende Repräsentation für die Übertragung von Einsen und Nullen zu finden, hat damit größtenteils nichts gemein.

2.2 Codierung – Was steht wofür?

Eben haben Sie gelesen, dass Nullen und Einsen nur die Interpretation von z.B. Stromflüssen sind. Und damit sind wir bei einem zentralen Begriff, mit dem die meisten InformatikerInnen sich eher ungern beschäftigen, weil es dabei ausschließlich um die Frage geht, wie etwas interpretiert wird. Der Bereich, von dem hier die Rede ist, wird als **Codierung** bezeichnet.

Vielleicht haben Sie schon etwas vom **ASCI**-Code gehört. Der ASCI-Code ist nichts anderes als eine Tabelle, bei der jedem Buchstaben und verschiedenen anderen Zeichen eine Zahl zugeordnet wird. Wenn Sie nur mit iOS- oder Windows-Rechnern arbeiten, haben Sie mit der Codierung im Regelfall nichts zu tun, aber sobald Sie Daten zwischen Computern austauschen, müssen sie darauf achten. Denn im Hintergrund werden alle Daten, die Sie eingeben in Form von Codes gespeichert. Wenn Sie Daten auf einen anderen Rechner übertragen, dann werden die Codes ausgetauscht. Und nur wenn beide Rechner die gleiche Codierung verwenden empfängt der zweite Computer die Daten, die Sie eingegeben haben.

An dieser Stelle werden wir uns einen Begriff ansehen, der Ihnen immer wieder mit jeweils unterschiedlicher Bedeutung begegnen wird: **Interface**s bzw. **Schnittstelle**n. Ein Interface ist etwas, das die Verbindung zwischen zwei „Dingen"herstellt. Wenn wir über den Datenaustauch zwischen zwei Rechnern reden, die beide unterschiedliche Codierungen verwenden, dann ist eine mögliche Bedeutung des Begriffs Interface der eines Übersetzers. Das Interface kennt dann die beiden Codierungen und wandelt die übertragenen Daten von der einen Codierung in die andere um, damit der Empfänger Daten in einer Codierung erhält, die er versteht.

Hier schon einmal ein Beispiel für eine ganz andere Bedeutung des Begriffs Interface: Bei der Programmierung in objektorientierten Sprachen kann ein Interface eine abstrakte Definition für einen Programmteil sein, der noch programmiert werden muss. Diese Interpretation des Begriffs Interface ist dann sinnvoll, wenn ein Programm im Team entwickelt werden soll. Das Interface enthält dann die Festlegung von Namen für „Befehle", die zwar noch nicht funktionieren, aber bei denen schon festgelegt ist, was sie später tun sollen. So können dann alle Mitglieder des Teams mit Ihren Aufgaben beginnen: Sie können zwar Ihren Programmcode noch nicht ausprobieren, aber da Sie bereits wissen, wie die Befehle lauten, die später eine bestimmte Funktion erfüllen werden, können Sie zumindest schon mit der Entwicklung neuer Programmteile beginnen.

Ähnlich wie für den Begriff des Programmierens gilt auch für den Begriff der Codierung, dass er eine Vielzahl unterschiedlicher Methoden zusammenfasst, die jeweils für einen bestimmten Anwendungsfall sinnvolle Lösungen anbieten. Mehr dazu können Sie in dem großartigen Buch **„Information und Codierung"** von **R.W. Hamming**[3] nachlesen. Codierung ist eines der zentralen Themen der Nachrichtentechnik und wird einführend in Veranstaltungen zur **Technischen Informatik** behandelt.

Im Bereich der **maschinennahen Programmierung** kommt dann noch dazu, dass es zwei unterschiedliche Reihenfolgen gibt, in der Daten im Speicher abgelegt werden können. Die unterscheiden sich jedoch nicht (!) dadurch, dass Sie einfach nur spiegelverkehrt wären. Auch hierüber müssen Sie Bescheid wissen, wenn Sie zwischen zwei Computer Daten austauschen wollen. Denn selbst wenn Sie sichergestellt haben, dass die Codierung des einen Computers richtig in die Codierung des anderen übersetzt wird, kann Ihnen die unterschiedliche Reihenfolge im Speicher noch alles zerschießen. Die meisten von Ihnen werden dieses Problem aber nicht haben, da die Prozessoren heutige Rechner von Apple und Microsoft die gleiche Reihenfolge bei der Datenspeicherung nutzen. Vor einigen Jahren, als Apple noch Prozessoren von Motorola verwendete, war es dagegen eines der zentralen Probleme beim Datenaustausch zwischen Rechnern mit dem Betriebssystem beider Rechner.

Mit Codierungen bekommen es bereits Programmiereinsteiger sehr schnell zu tun, auch wenn wir dann von **Datentyp**en sprechen. Denn letztere basieren im Grunde auf der Codierung: Sobald Sie eine Zahl oder eine andere Zeichenfolge in einem Programm verwenden, kann es dazu kommen, dass Ihr Programm vermeintlich falsche Ergebnisse liefert. Aber wie schon aber

[3] Hamming ist gewissermaßen der Godfather der Nachrichtentechnik.

geschrieben: Das ist nicht der Fehler der Rechners, sondern der Fehler des Entwicklers, also ggf. von Ihnen, da hier die Arbeitsweise des Rechners ignoriert wurde.

Kontrolle Texte und andere Inhalte werden bei der Speicherung codiert. Das bedeutet in der Informatik, dass Sie als Folgen von Binärziffern im Speicher liegen. Je nachdem, wie oder was Sie programmieren, werden Sie detaillierte Informationen über verschiedene Codierungsverfahren benötigen.

2.3 Informatik und Nachrichtentechnik - Die zankenden Geschwister

Wie beschrieben beschäftigen sich **Nachrichtentechnik**erInnen vorrangig mit der Frage, wie Daten mit möglichst geringem Energieaufwand so übertragen werden können, dass sie am Ende der Leitung empfangen und verstanden werden können. **Informatik**erInnen dagegen beschäftigen sich vorrangig mit der Frage, wie Daten möglichst schnell verarbeitet werden können.

Aus diesen unterschiedlichen Perspektiven resultiert eine vollständig andere Herangehensweise an Aufgaben. Und leider führt das häufig zu einer Trennung zwischen NachrichtentechnikerInnen und InformatikerInnen bzw. zwischen Ihnen und Ihren Kommilitonen der Medientechnik bzw. in der Medieninformatik. Dabei ließen sich ungemein spannende Projekte realisieren, wenn Sie es nur schafften, diese Kluft zu überwinden. Somit verfolgt dieses Kapitel auch das Ziel, Ihnen ein wenig Verständnis für dieses ungemein spannende aber auch höchst anspruchsvolle Gebiet zu vermitteln.

Das was wir heute als Computer bezeichnen, ist in aller Regel eine Art Black Box, in der mehrere Hundert Komponenten unabhängig voneinander arbeiten und über verschiedene Arten von Datenübertragungswegen miteinander kommunizieren. Vermutlich sind Sie jetzt skeptisch, da Sie wissen, dass in einem Computer Komponenten wie das Mainboard, Festplatten, Grafikarten und ähnliches vorhanden sind. Der Begriff der Black Box gilt insbesondere bei Smartphones, wo sie im Gegensatz zu den meisten anderen Rechnern nicht einmal mehr eine Grafikkarte oder einzelne Laufwerke erkennen können. Dennoch ist es so, denn wenn Sie beispielsweise ein Mainboard als eine einzige Komponente betrachten, dann ignorieren Sie, dass es sich bereits hier um eine Ansammlung von Dutzenden Komponenten handelt. Und dann gibt es zusätzlich noch Technologien wie die **VLSI**, die very large scale integration, bei der es darum geht, eine sehr

große Anzahl von eigenständigen Einheiten in einem Bauteil zu integrieren.

Wie Sie in den Veranstaltungen der **Technischen Informatik** lernen, besteht ein Prozessor (fachlich korrekt ein **Mikroprozessor**) mindestens aus den drei Komponenten Rechenwerk, Steuerwerk und Speicher sowie den Verbindungen dazwischen, die als Bus bezeichnet werden. Das ist eine Abstraktion, die einem Prozessor aus den 50er Jahren entspricht, die aber genau dem entspricht, was wir bei der **imperativen Programmierung** beachten müssen. Wenn Sie mehr über den Aufbau von Computern mit Mikroprozessoren wissen wollen, werfen sie beispielsweise einen Blick in den Band **„Rechnerarchitektur"** von **Paul Herrmann**.

Wenn wir uns nun die Arbeitsweise eines Computers unvoreingenommen[4] ansehen, dann ist die wichtigste Komponente aber nicht „der"Prozessor, sondern die Vielzahl der Kommunikationswege zwischen all den Komponenten, aus denen er besteht. Die bekannteste Art dieser Kommunikationswege wird als **Bus** bezeichnet. Womit wir bei der Antwort auf die Frage wären, warum anstelle der Bezeichnung Nachrichtentechnik häufig den Begriff der **Kommunikationstechnik** benutzt wird und warum die auch für **Informatik**erInnen so wichtig ist: Wenn die Komponenten unseres Rechners keine Daten austauschen könnten, könnten wir mit Ihnen exakt gar nichts anfangen. Ohne die Arbeit der Kommunikations- und Nachrichtentechnik wäre unsere Arbeit also gar nicht möglich.

Für Einsteiger in die imperative Programmierung scheint diese Aufteilung in Prozessor, Bus und Speicher irrelevant zu sein, doch das ist schlicht falsch: Die meisten Fehler und Missverständnisse von Einsteigern kommen schlicht dadurch zustanden, dass Ihnen häufig nicht erklärt wird, wie eine **Variable** im Speicher abgebildet wird und was der **Datentyp** damit zu tun hat. Damit Sie professionell Software entwickeln können müssen Sie aber die aus der genannten Dreiteilung erwachsenden Probleme und Lösungen kennen: Wenn Sie später scheinbar simple Dinge wie den Zugriff auf eine Datei (Speichern bzw. Laden) selbst programmieren müssen, dann müssen Sie zuerst verstanden haben, dass es eine Datenübertragung zwischen dem Laufwerk und dem Prozessor gibt. Als nächsten müssen Sie verstanden haben, dass Sie niemals auf das Laufwerk, sondern nur auf den Datenstrom zugreifen können, der zwischen Prozessor und Laufwerk existiert. Sie müssen weiterhin beachten, dass die Datenübertragung über den Datenstrom eine gewisse Zeit dauert. Dann müssen Sie verstanden ha-

[4] Mit unvoreingenommen ist hier gemeint, dass Sie sich die Arbeitsweise eines Computers ansehen und nicht die Darstellung auf einem Display bzw. die Eingabemöglichkeiten in Form von Maus, Tastatur usw.

ben, dass bei dieser Datenübertragung einiges schieflaufen kann, was das ist und was Sie ggf. tun müssen, damit die Datenübertragung trotzdem klappt. Weiterhin müssen Sie sich unter Umständen damit auseinander setzen, welche Codierung bei der Speicherung der Daten verwendet werden muss. In dem Fall müssen Sie die Codierung und Decodierung programmieren. Und es gibt noch eine Vielzahl weiterer Probleme, die bei der Nutzung eines Buses auftreten können. Für viele davon hat die **Nachrichtentechnik** Lösungen entwickelt, aber einige müssen Sie selbst im Rahmen der **Programmierung** lösen. Und das können Sie dann und nur dann, wenn Sie intensiv mit all den Problemen beschäftigen, die z.B. bei der Programmierung eines Taschenrechners irrelevant sind. Aber keine Sorge, wir fangen ganz einfach an. So lange Sie die Inhalte dieses Buches konsequent durcharbeiten und in eigenen Programmen umsetzen, werden Sie all das und noch viel mehr im Laufe der Zeit beherrschen.

Umgekehrt machen jedoch auch die Kommunikations- und NachrichtentechnikerInnen häufig den Fehler, die Konzepte und Modellierungen der Informatik als größtenteils untauglich oder überflüssig zu betrachten. Dabei sind diese Ergebnisse der Informatik Lösungen zu genau den Problemen, die bei der Nutzung nachrichtentechnischer Systeme entstanden sind bzw. entstehen. Hier sei wieder einmal auf das große Gebiet der **Praktischen Informatik** verwiesen, dass TechnikerInnen in aller Regel nicht kennen.

Bitte beachten Sie, dass wir hier über Kommunikationstechnik reden; in den **Kommunikationswissenschaften** (Teilbereich der Sozial- und Geisteswissenschaften) geht es um die Kommunikation zwischen Menschen.

Kontrolle Informatik und Nachrichtentechnik sind im Grunde wie zwei Seiten einer Medaille: Die eine kann ohne die andere nicht existieren. Im Regelfall sind beide für die Leistungen der jeweils anderen jedoch blind.

2.4 Von der Nachrichtentechnik zu logischen Gattern

Ein zweiter Bereich, mit dem sich vorrangig die Elektro- bzw. **Nachrichtentechnik** beschäftigt, ist der Aufbau von Komponenten, die logische Operationen ausführen. Diese bestehen seit mehr als fünfzig Jahren aus Kondensatoren. Wenn Sie sich nicht mehr an den Physikunterricht erinnern: Ein Kondensator ist ein Bauteil, über das mittels einer Eingangsleitung gesteuert werden kann, ob Strom weitergeleitet wird oder nicht. Hier sind wir dann auch in einem Bereich, in dem eine 1 gleichbedeutend ist mit fließendem Strom und eine 0 mit nicht fließendem Strom.

Solche logischen Gatter können Sie bei den **FPGA**s, den Field-Programmable Gate Arrays direkt programmieren[5]. Wenn Sie das tun, dann befinden Sie sich in einem der wenigen Gebiete, in dem Elektrotechniker und Informatiker dieselbe Sprache sprechen. Diese Technologie wurde erst Mitte der 80er Jahre entwickelt. Und obwohl wir hier von einer Programmiertechnik reden, werden Sie in Büchern über Programmierparadigmen kein Kapitel finden, dass das entsprechende Paradigma enthält.

Der Grund dafür ist ganz einfach: Diese Art der Programmierung ist eine direkte Umsetzung dessen, was in der Mathematik **boolesche Algebra** genannt wird. Und an dieser Stelle möchte ich eine Lanze für die **Mathematik** brechen: Immer wieder fallen Formulierungen wie „Mathe ist doch nur eine Hilfswissenschaft." Das allerdings ist ein Ausdruck, der eher auf die Ignoranz der Person verweist, die ihn gebraucht. Denn tatsächlich gab es die bei Computern verwendete Logik lange vor dem ersten Computer.

Und Mathematik hat zunächst auch nichts mit Rechnen zu tun: Im alten Griechenland gab es eine Disziplin namens Philosophie. Die Philosophen versuchten die Welt und das was in ihr geschieht zu erklären. Daraus entstanden dann weitere Disziplinen wie die Naturwissenschaften, die sich auf bestimmte Teilgebiete der Welt und des Universums konzentrieren. Wenn Sie also bislang über Philosophie als eine Wissenschaft betrachtet haben, in der es darum geht, wie Menschen sich verhalten sollten, dann haben Sie hier eine zu beschränkte Vorstellung.

Es gab aber auch Forscher, die sich fragten, wie ein Universum aussehen würde, wenn sie den umgekehrten Weg wählen würden. Sie untersuchten deshalb, wie ein Universum aussehen würde, für das Eigenschaften willkürlich festlegt werden. Um das nochmal zu betonen: Im Gegensatz zu allen anderen Wissenschaften setzt sich die Mathematik nicht mit der Beschaffenheit eines einzigen (nämlich unseres) Universums auseinander: Sie geht wesentlich weiter! Sie stellt sich die Frage wie jedes nur denkbare Universum aussehen würde. Und wer so etwas als Hilfswissenschaft abtut, der... Nun ja, wie soll ich es höflich ausdrücken?

Wenn Sie jetzt genau gelesen haben, dann verstehen Sie, warum die Aussage wie „das wurde mathematisch bewiesenïn den Naturwissenschaften häufig einen falschen Eindruck vermittelt: Sie können mathematisch alles beweisen, wenn Sie nur kreativ genug bei der Entwicklung von Annahmen sind. Ein mathematischer Beweis alleine genügt also nicht, um zu bewei-

[5]Diese Art der Programmierung ist Teil der Veranstaltung Informatik 2 für Media Systems. Das bedeutet leider auch, dass Sie am Studienende keine Kenntnisse der Praktischen Informatik haben werden. Dafür werden Sie im Bereich der Technischen Informatik deutlich mehr Kenntnisse besitzen als Ihre Kommilitonen von anderen HAWs bzw. FHs.

sen, dass etwas wahr ist. Sie müssen außerdem beweisen, dass die Voraussetzungen bzw. Annahmen wahr sind, die Sie für Ihre Beweisführung vorausgesetzt haben.

Ein aktuelles Beispiel sind die „Beweise"rund um den Urknall. Die Physik ist zurzeit im Stande, zu beweisen, in welchem Zustand sich das Universum wenige Sekunden nach dem sogenannten Urknall befand. Was also direkt zum „Zeitpunkt"des Urknalls passiert oder was möglicherweise davor „warïst zumindest momentan nicht beweisbar. Es gibt jedoch PhysikerInnen, die darauf beharren, das zu können. Sie legen dafür mathematische Beweise vor, die in sich schlüssig sind. Der Fehler liegt hier aber nicht in der mathematischen Beweisführung, sondern darin, dass sie Annahmen treffen, die zumindest noch nicht bewiesen sind. Wie Sie in „**Mathematik 1**" lernen ist es aber möglich, aus einer falschen Annahme wahre und falsche Schlussfolgerungen zu ziehen. Wenn also PhysikerInnen Annahmen für eine mathematische Beweisführung verwenden, die noch nicht bewiesen sind, dann kommen sie damit zu Aussagen, bei denen unklar ist, ob sie nun wahr oder falsch sind. Also haben die eingangs genannten Forscher nicht etwa den Urknall oder Abläufe rund um den Urknall bewiesen, sondern lediglich gezeigt, welche Abläufe dort stattgefunden haben, wenn bestimmte Annahmen wahr sind.

Jedenfalls sollte Ihnen damit klar sein, dass MathematikerInnen häufig schon Lösungen parat haben, wenn die zugehörigen Probleme noch gar nicht in der Praxis auftreten. Und nein, wir reden hier nicht von Monaten; einige Lösungsansätze, die in den letzten Jahrzehnten zum Einsatz kamen wurden bereits vor mehreren hundert Jahren von Mathematikern konzipiert. Wenn Sie also in einem mathematischen Lehrwerk Formulierungen lesen wie „Es sei ein Universum mit den Eigenschaften...", dann wissen Sie jetzt, dass das wortwörlich gemeint ist: Es geht wirklich darum, ein Universum oder einen Teilbereich eines Universums zu beschreiben, das über bestimmte Eigenschaften definiert wird. Dieses Universum hat aber in aller Regel nicht das geringste mit dem zu tun, was Sie sich als ein Universum vorstellen.

Hier auch noch ein Exkurs: Häufig wird von radikalen (Mono-)theisten behauptet, Naturwissenschaftler würde behaupten, dass es Gott nicht gäbe. Das ist falsch: Naturwissenschaftler untersuchen, wie das Universum aussieht. Sie untersuchen dabei, welche Gesetzmäßigkeiten im Universum nachweisbar sind. Das hat aber nichts mit der Frage zu tun, ob es einen Gott gibt, der all das erschaffen hat.

Aber zurück zu den FPGAs: Wenn Sie die boolesche Algebra nicht beherrschen, können Sie auch kein FPGA programmieren. Unabhängig davon

kommt die boolesche Algebra auch immer dann zum Einsatz, wenn Sie den Rechner etwas prüfen lassen wollen. Für Fortgeschrittene gibt es dann zwar noch Mittel wie die Fuzzy Logic, aber das überlassen wir mal lieber den MathematikernInnen und universitären InformatikerInnen.

Kontrolle
Eine erste Art der Programmierung besteht in der Umsetzung der booleschen Algebra, einer mathematischen Methode, um aus einfachen Ja-/Nein-Fragen komplexe Modelle zu entwickeln. Prozessoren basieren auf nichts anderem.

2.5 Weiter zur Maschinensprache

Wenn wir nun einen Prozessor nicht entwickeln, bzw. seine Arbeitsweise programmieren wollen, sondern ihn so nutzen wollen wie er ist, um Programme zu entwickeln, dann ist die erste Methode die **Maschinensprache**. Danach folgt die **maschinennahe Programmierung**.

Wie Sie wissen reden wir von Prozessoren mit einer gewissen **Bittigkeit**. Heute sind die 32-Bit- und die 64-Bit-Prozessoren am bekanntesten. Diese Bittigkeit bedeutet nichts anderes, als dass der Prozessor bei jedem Rechenschritt eine Zahl mit genau dieser Anzahl an Bits addieren kann. Ein 32-Bit-Prozessor kann also bei jedem Rechenschritt eine Zahl zwischen 0 und $2^{32} - 1$ zu einer anderen Zahl in diesem Bereich addieren[6]. Wie Sie in der **Technischen Informatik** erfahren liegt darin auch der Grund, dass 32-Bit-Prozessoren nur einen Speicher mit rund 4 GB verwalten können: Für mehr Speicherbereiche haben Sie schlicht keine „Hausnummern".

Aufgabe:
Berechnen Sie doch mal eben die maximale Größe für Speicher, die ein 64-Bit-Prozessor nutzen kann. Und suchen Sie nach einem anschaulichen Beispiel, dass diese Zahl verdeutlicht. (Anmerkung, die nötig ist, weil leider einige Studienanfänger das Potenzrechnen nicht beherrschen: 2^{64} ist nicht (!) $2 \cdot 2^{32}$. Die Antwort lautet also nicht „etwas weniger als 8,6 Millionen".)

Die Maschinensprache besteht dementsprechend aus nichts anderem als Zahlen, die in dem Bereich liegen, der von der Bittigkeit des jeweiligen Prozessors abhängt. Ob eine Zahl nun ein Befehl darstellt, ob Sie als Zahl zu verwenden ist oder ob es eine Position im Speicher sein soll, kann nur wissen wer die Maschinensprache eines Prozessors beherrscht; es gibt keine Möglichkeit, einer Zeile eines Programms in Maschinensprache anzuse-

[6] 2^{32-1} entspricht etwas weniger als 4,3 Millionen

hen, wofür sie steht.

Kontrolle
Maschinensprache ist so schlecht lesbar, dass Sie sie am besten gleich ignorieren. Verwechseln Sie aber bitte nicht die Maschinensprache (siehe dieser Abschnitt) mit der maschinennahen Programmierung (Assembler, siehe nächster Abschnitt).

2.6 Maschinennahe Programmierung – Assembler

Nachdem wir jetzt geklärt hätten, wie man einen Prozessor besser nicht programmiert, kommen wir zur ersten Methode, um einen Prozessor sinnvoll zu programmieren: Die maschinennahe Programmierung, die auch als **Assembler** oder Assembler Programmierung bezeichnet wird. Es ist die erste Variante der **imperativen Programmierung** und leider ist sie alles andere als anschaulich. Aber wenn Sie sich hiermit beschäftigen, dann zu **C** und später zu **C++** oder **Java** weitergehen, werden sie verstehen, was die Vorteile dieser Sprachen sind und werden diese zu schätzen wissen.

Im Regelfall lernen Sie zu Beginn einer Veranstaltung, in der Sie die maschinennahe Programmierung kennen lernen, etwas über den Aufbau des Prozessors, den Sie maschinennah programmieren werden. Da tauchen dann Begriffe wie Pipeline, RISC und CISC und andere auf. Das kann Ihnen helfen, bestimmte Abläufe besser zu verstehen. Für den Einstieg genügt es aber, wenn Sie die Dinge kennen lernen, die Sie tatsächlich programmieren können.

Aber bevor wir auf die eingehen, sollten Sie wieder (wie zu Beginn dieses Buches) die Antwort auf die Frage erfahren, was **maschinennahe Programmierung** eigentlich ist. Und die Antwort hierauf ist genau die, die allgemein mit dem Begriff Programmieren verbunden wird: Sie tippen Zeile für Zeile ein, was der Computer (besser gesagt der Prozessor) tun soll und der führt es dann in genau dieser Reihenfolge aus. Einzige Ausnahme: Sie können Sprünge ins Programm einbauen, die dafür sorgen, dass der Prozessor einen anderen Teil Ihres Programms ausführt, aber am zeilenweisen Ablauf ändert sich ansonsten nichts. Diese anderen Teile werden als **Subroutine**n bezeichnet. Es sind Programmteile, die an verschiedenen Stellen im Programm ausgeführt werden sollen. Und sie werden einzig und allein deshalb ausgelagert, weil sie dadurch nur einmal programmiert, aber mehrfach ausgeführt werden können. Das reduziert die Fehleranfälligkeit und erleichtert die spätere Verbesserung des Programms.

In der maschinennahen Programmierung besteht nun jede Zeile aus ei-

nem sogenannten **Mnemon** und zwischen keiner und mehreren Zahlen. Ein Mnemon ist einem Hilfswort, das einem Befehl des Prozessors entspricht. Im Gegensatz zur tatsächlichen Maschinensprache können Sie hier also Buchstabenkombinationen für Befehle benutzen. Aber ob eine Zahl eine Zahl oder eine Speicheradresse ist, das müssen Sie zusammen mit der Definition jedes Mnemons lernen.

Dennoch gibt es Gründe, wegen denen bestimmte Entwickler immer noch in Assembler programmieren und aus denen es tatsächlich Sinn macht, sich auf diese Art der Programmierung zu konzentrieren: Es gibt keine effizientere Möglichkeit, einen Computer zu programmieren und viele Prozessoren lassen sich nicht vollständig in anderen Sprachen programmieren. Der Bedarf an Effizienz muss allerdings sehr hoch sein, denn mit C als höherer Sprache können Sie immer noch recht maschinennah und effizient programmieren.

Das Standardwerk für die maschinennahe Programmierung stammt von **Donald E. Knuth** und hat den Titel „**The Art of Computer Programming**". Es besteht aus sieben Bänden, von denen die ersten drei veröffentlich wurden. Band vier wurde in zwei Teile unterteilt, von denen der erste ebenfalls veröffentlicht wurde. Die übrigen Bände sind noch in Arbeit. Das mag so klingen, als wenn eigentlich ein anderes Buch als Standard gelten sollte, doch im Gegensatz zu allen mir bekannten Autoren untersucht Knuth in seinem Buch jedes grundlegende Problem, das bei imperativer Programmierung vorkommen kann. Insbesondere wird in seinem Band praktisch ab der ersten Seite klar, warum InformatikerInnen welche mathematischen Grundlagen beherrschen müssen.

Dennoch lassen sich die Bücher auch ohne diese mathematischen Grundlagen durcharbeiten: Knuth hat viele Passagen so verfasst, dass klar erkennbar ist, ob sie auch ohne mathematische Grundlagen verständlich sind. Das gleiche gilt für die Vielzahl an Aufgaben, die den Text begleiten. Die Musterlösungen machen beispielsweise rund ein Drittel des ersten Bandes aus. Damit kann jedeR Interessierte sich unabhängig von den persönlichen Kenntnissen weitgehend in das Themengebiet einarbeiten.

Kontrolle
Auch die maschinennahe Programmierung ist eher abstrakt. Die Zeilen eines solchen Programms bestehen aus Buchstabenkürzeln und Zahlen. Was das Programm bei der Ausführung an welcher Stelle tut ist sehr schwer zu erkennen.

2.7 Zusammenfassung

In diesem Kapitel haben Sie erfahren, dass die Nachrichtentechnik die Techniken und Technologien liefert, mit der InformatikerInnen Ihre Ergebnisse in die Praxis umsetzen können. Ohne Nachrichtentechnik gibt es keine IT-Systeme und letztlich würden Sie nicht nur dieses Buch nicht lesen, Sie würden wahrscheinlich etwas ganz anderes studieren, weil Computer und das Internet nicht existieren würden. Unter Umständen hätten Sie nicht einmal die Hochschulreife erreicht.

Sie haben außerdem einen ersten Einblick in das erhalten, was hinter den Nullen und Einsen steht, und dass das etwas ganz anderes ist, als das, was die meisten sich darunter vorstellen. Sie haben ebenfalls erfahren, dass wir auch die Einsen und Nullen in aller Regel nicht wahrnehmen, weil sie sich hinter den Ergebnissen der Codierung verbergen.

Abschließend haben Sie einen ersten Blick darauf erhascht, wie all das zu Programmiersprachen führt. Im nächsten Kapitel geht es dann um Formen der Programmierung, mit denen Sie in den nächsten Jahren voraussichtlich am häufigsten zu tun haben werden.

Stichwortverzeichnis