Georg P. Loczewski

Von A++ nach ARS++

A++ mit einer Schnittstelle zu anderen Sprachen

tredition®

Von A++ nach ARS++

Georg P. Loczewski

Von A++ nach ARS++

A++ mit einer Schnittstelle zu anderen Programmiersprachen

Mit einer Einführung in das Lambda-Kalkül

tredition®
www.tredition.de

IMPRESSUM

Copyright c 2018 Georg P. Loczewski
Von A++ nach ARS++
A++ mit einer Schnittstelle zu anderen Programmiersprachen
1. Auflage 2018 –Hamburg
Verlag & Druck: tredition GmbH

ISBN
978-3-7469-3643-7 (Paperback)
978-3-7469-3644-4 (Hardcover)
978-3-7469-3645-1 (e-Book)

Meiner Frau Ursula
und meinen Söhnen Thomas und Johannes in Liebe
gewidmet

Vorwort

Einleitung

Dieses Buch ist basiert auf dem 1032-seitigen Buch **‘Programmierung pur’**, das in der 2. erweiterten Auflage von demselben Autor im Verlag S.Toeche-Mittler in Darmstadt im Jahre 2003 veröffentlicht wurde. Die 1. Auflage erschien im Jahre 2002. *(Siehe [56].), [55])*.

Die Eigenschaften von A++ werden in verschiedenen Büchern in dieser Reihe vorgestellt, jedesmal mit einem anderen Schwerpunkt. Das **vorliegende Buch** wird unter **Punkt 4** in der folgenden Liste beschrieben.

1. In **A++ Die kleinste Programmiersprache der Welt**[58] geht es darum, die theoretischen Grundlage dieser Sprache zu präsentieren und die Mächtigkeit von A++ aufzuzeigen.
2. In dem 2. Buch **Programmieren lernen mit A++**[61] ist auch eine Einführung in A++ enthalten, der Fokus des Buches liegt aber darauf zu zeigen, wie in den anwendungsorientierten **Programmiersprachen Python und Java** A++-orientiert programmiert werden kann.

In der englischsprachigen Ausgabe *A++ The Smallest Programming Language in the World* wird außerdem die **Programmiersprache Perl** und die A++-orientierte Programmierung in dieser Sprache erläutert.*Siehe:* [59]!

3. In dem 3. Buch **A++ und systemnahe Programmiersprachen**[60] liegt der Schwerpunkt darauf, aufzuzeigen, wie auch in den Programmiersprachen C und C++ A++-orientiert programmiert werden kann.
4. In dem 4. Buch **Von A++ nach ARS++**[62] wird schließlich eine Erweiterung von A++ vorgestellt, die einer neuen Programmiersprache entspricht (mit Compiler und virtueller Maschine) in der die Funktionalität von Python, Java, C++ und C enthalten ist. Dies ist mglich, da in ARS++ eine Schnittstelle zu den anderen Sprachen namens ARSAPI eingebaut ist.

Die **Werkzeuge und Hilfsmittel** zu diesem Buch werden in einem Download-Verzeichnis im Internet bereitgestellt. *Siehe: B.3 auf Seite 397*.

Theoretische Grundlagen von A++

Das Wesentliche der Programmierung

Das Buch stellt auf einfache Weise die Grundlagen der Programmierung auf der Basis des Lambda-Kalküls vor. In dem Bemühen, Programmierung aus der Sicht des Lambda-Kalküls zu verstehen, geht es darum, das *Wesentliche der Programmierung* zu erfassen und uns davor zu bewahren, uns von einer Unzahl von Vorschriften und Regeln einer

bestimmten Programmiersprache die Programmierung an sich vergraulen zu lassen. Es wird hier dank des Lambda-Kalküls eine Sicht der Programmierung gewonnen, die eine **befreiende Wirkung** hat. Das Denken wird aus den *Niederungen des komplexen Regelwerks einer bestimmten Programmiersprache* herausgeholt und heraufgehoben auf die *Höhen eines einfacheren, umfassenderen und deshalb mächtigeren Denkens*. Das Lambda-Kalkül bietet die theoretische Grundlage für eine solche Sicht.

ARS — *Verallgemeinerung des Lambda-Kalküls*

Die meisten Bücher, die die Thematik des Lambda-Kalküls aufgreifen, münden ein in die funktionale Programmierung und wenden sich ausschließlich den rein funktionalen Sprachen zu. Imperative oder objekt-orientierte Sprachen finden in solchen Büchern keine Berücksichtigung. In diesem Buch wird gezeigt, dass die drei Grundoperationen des Lambda-Kalküls, nämlich **A***bstraktion*, **R***eferenz* und **S***ynthese*, so allgemein sind, dass sie in fast jeder Programmiersprache angewandt werden können. Diese Grundoperationen mögen zwar sehr trivial und abstrakt klingen, *sie verändern jedoch die Denkweise und die Art zu programmieren sehr*.

Erstaunlicherweise scheint die Programmierung als solche eine große *Verwandtschaft mit der Kunst* zu haben. In beiden menschlichen Aktivitäten wird die einen umgebende Wirklichkeit wahrgenommen und *Abstraktionen* dieser Wirklichkeit gebildet und in sich aufgenommen. Das Abstrahieren wird besonders deutlich in der urmenschlichen Aktivität der Namensvergabe. Mit der Bildung eines Namens wird eine Abstraktion vorgenommen. Der Name steht für die

Wirklichkeit. Von allen Details dieser Wirklichkeit wird mit der Namensbildung abstrahiert. Die so gebildeten vielfältigen, ja unzähligen Abstraktionen werden vom Künstler wie vom Programmierer je nach Eigenart der Person gegebenenfalls neu zusammengesetzt. Es entsteht etwas Neues, mitunter etwas noch nie Dagewesenes. Diese Operation nennen wir *Synthese* oder auch *Applikation*. Die Bezugnahme auf die Abstraktionen der Wirklichkeit bezeichnen wir als *Referenz*.

Hiermit haben sich *drei Grundoperationen* herauskristallisiert, die das künstlerische Schaffen charakterisieren, so wie das der Programmierer.

Abstraktion: Etwas einen Namen geben.

Referenz: Auf etwas mit seinem Namen Bezug nehmen.

Synthese: Zwei oder mehrere Abstraktionen miteinander verknüpfen und dadurch eine neue Abstraktion bilden.

Was für ein Zufall, dass die ersten Buchstaben dieser drei Begriffe tatsächlich *das Wort Kunst* ergeben, nämlich in der lateinischen Sprache, einer der beiden Muttersprachen unseres abendländischen Kulturraumes: **ARS**! Die Kunst besteht nun darin, sinnvolle Abstraktionen zu bilden und ebenso sinnvolle Synthesen vorzunehmen.

Dass mit diesen drei Grundoperationen ein mathematisch-logisches System aufgebaut werden kann, hat der amerikanische Logiker *Alonzo Church* 1941 bereits in seinem Werk: *The Calculi of Lambda Conversion* bewiesen.

Seine Theorie ist unter dem Namen *Lambda Calculus* oder auf deutsch *Lambda-Kalkül* bekannt geworden. Siehe auch hierzu [5].

Dass auf dieser Grundlage auch programmiert werden kann, ist in verschiedenen Büchern der Informatik ebenfalls bewiesen worden. Siehe hierzu [48],[47] und [13]. Wer mit der Kargheit dieser drei Grundoperationen gelernt hat umzugehen, sich sozusagen bewährt hat, der kann sich gestatten, ein paar weitere Primitivoperationen bzw. Syntax-Elemente hinzuzunehmen, um sich das Leben etwas zu versüßen. Dies haben *Guy L. Steele* und *Gerald J. Sussman* 1975 am MIT gemacht und so die Programmiersprache *Scheme* geschaffen.

Genau genommen stellt **ARS** eine *Verallgemeinerung des Lambda-Kalküls* dar. Die Verallgemeinerung besteht darin, dass der Begriff der Abstraktion allgemeiner als im Lambda-Kalkül definiert wird, nämlich als ’etwas einen Namen geben’. Hinter dem Namen verbergen sich alle Details des Definierten. *Eine solche Namensvergabe setzt eine explizite Namensdefinition voraus*.

Im Lambda-Kalkül dagegen ist eine *explizite Vergabe eines Namens für eine Lambda-Abstraktion* bei deren Bildung nicht vorgesehen. Dort erfolgt sie lediglich implizit bei einer Synthese von Lambda-Ausdrücken.

Die Auswirkungen dieses zunächst als klein erscheinenden Unterschiedes sind gewaltig: Während ein Ausbau des Lambda-Kalküls immer in die Funktionalen Programmiersprachen mündet, können mit ARS **allgemeine Muster der Programmierung** definiert werden, die

sowohl auf die *Funktionale Programmierung* als auch auf die *Objekt-orientierte* und die *Imperative Programmierung* angewandt werden können.

Wer sich in seinem Programmieren von den *in ARS formulierten Prinzipien* leiten lässt, wird Programme erstellen, die nicht nur funktionieren, sondern die auch schön sind, Programme, bei denen der Leser mit Bewunderung die Abstraktionen und Synthesen nachempfinden und genießen kann.

A++ — *die kleinste Programmiersprache der Welt([58])*

Dass es möglich ist, ausgehend von den drei aufgezeigten Grundoperationen zu programmieren, und zwar ausschließlich auf ihnen aufbauend, wird im Teil 2 des Buches ausführlich aufgezeigt. Hier wird **ARS** als die kleinste Programmiersprache der Welt mit dem Namen **A++** unter dem Gesichtspunkt der Sprache an sich und dem ihrer Anwendung vorgestellt. Der Name A++ ergibt sich ganz von selbst aus ARS: **A**bstraction **+** Reference **+** Synthesis.

A++ wird eingeführt als das *universale Instrument zum Erlernen der Programmierung*, das uns mit dem Wesentlichen der Programmierung konfrontiert und uns hilft diese Konfrontation zu meistern.

ARS++ — *A++ im großen Stil*

Nach intensiver Auseinandersetzung mit dem, was Programmierung im Wesentlichen ist, stellen wir uns den

Problemen der Programmierpraxis. Um diese Konfrontation bestehen zu können müssen wir A++ ausbauen zu einem *Instrument der Praxis*, d.h. zu einer Programmiersprache, die den Anforderungen der Programmierpraxis genügt.

Diese neue Konfrontation führt uns zu ARS++, einer Programmiersprache, die auf ARS basiert, die Funktionalität der Programmiersprache Scheme einschließt und durch Zusätze Scheme übertrifft. Der *Name ARS++* ist abgeleitet aus **ARS +** *Scheme* **+** *Extensions*.

Es sei noch kurz angesprochen, warum statt ARS++ nicht Scheme selbst als Programmiersprache in diesem Buch eingesetzt wurde. *Scheme ist eine wunderbare Programmiersprache*, aber leider sind in der Sprache selbst für die Programmierpraxis wichtige Randzonen nicht geregelt.

Die verschiedensten *Scheme-Implementierungen* versuchen nun auf ihre Weise Ergänzungen vorzunehmen, wie wir das ja auch in ARS++ getan haben. Leider scheint es jedoch keine Scheme-Implementierung zu geben, die all die Bereiche abdeckt, die wir in den Fallstudien benötigen und die außerdem noch auf allen Plattformen zur Verfügung steht und schließlich auch noch freie Software ist.

So waren wir gezwungen ARS++ zu kreieren mit den **zusätzlichen Vorteilen** der vollen *ARS-Kompatibilität*, der *Offenheit und Nachvollziehbarkeit der Implementierung* und der sich daraus ergebenden *Ergänzungsmöglichkeit* nach einem bereits vorgegebenen Muster.

ARS und die anderen Sprachen

Ein weiteres Hauptthema von ‘A++ in Theorie und Praxis’ ist die Auswirkung der sich in ARS manifestierenden Denkweise auf den Umgang mit den großen wohl etablierten Programmiersprachen.

Es wird Wert darauf gelegt aufzuzeigen, dass selbst das Lernen von neuen Programmiersprachen nach einer Intensivschulung in A++ um ein Vielfaches erleichtert wird und schneller zu Produktivität in der neuen Sprache führt.

Für diese neue Konfrontation **wurde in diesem Buch die Programmiersprache Java ausgewählt.** In dem Buch *‘Programmierung pur’[56] (ISBN 3-87820-108-7)*, das vom selben Autor im Jahre 2003 im S.Toeche-Mittler Verlag veröffentlicht wurde, werden ebenso folgende Programmiersprachen berücksichtigt: C, C++, Python und Perl.

Für die **Auswahl der Sprachen** in den beiden Büchern waren folgende Überlegungen entscheidend:

1. **Java** ist die populärste Programmiersprache.
2. **Python** ist eine relativ junge Sprache, die sich auszeichnet durch ein hohes Maß an Einfachheit, Mächtigkeit, Flexibilität und Dynamik. Außerdem gibt es für Python Anwendungsbeispiele aus fast allen Gebieten und eine überaus reichhaltige Bibliothek von Funktionen und Klassen. All dies ist im Quellcode verfügbar.
3. **Perl** Es hat sich gezeigt, dass es sinnvoll ist die Programmiersprache **Perl** ebenfalls in diesem Buch zu berücksichtigen, da sie nicht nur eine von Systemadministratoren sehr geschätzte Skriptspache

ist und als Internetsprache ebenso weite Verbreitung gefunden hat, sondern auch deswegen, weil in ihr die wesentlichen Gedanken dieses Buches direkt umgesetzt werden können. Man kann sagen, dass ’Perl’ eine Lambda-Sprache ist.

4. **C++** ist die Programmiersprache, die heutzutage meistens gewählt wird, wenn es darum geht, große Anwendungspakete zu erstellen, bei denen Effizienz eine hohe Priorität besitzt.
5. **C** ist die Sprache, aus der C++ entstanden ist, der Sprache Nr. 1, wenn es darum geht, systemnahe Software, wie Betriebssysteme (z.B. Linux-Kernel, Compiler etc) zu programmieren, oder wenn Effizienz der Anwendung unverzichtbar an erster Stelle steht.

   Die Programmiersprache C ist nun mal die Sprache mit der größten Maschinennähe und damit die Sprache, in der die Computer-Ressourcen am besten genutzt werden können. Das was ihr fehlt ist die Nähe zu den zu programmierenden Anwendungen. Diese Verbindung schafft **ARSAPI**.

In ARS geht es nur um die Lösung der Probleme. In ARS wird abstrahiert von dem Computer und der Syntax irgendeiner Sprache. **ARSAPI** schlägt die Brücke zwischen beiden extremen Polen. Mit Hilfe von **ARSAPI** ist es möglich auf einfache Weise in C objekt-orientiert und funktional zu programmieren ohne viel von der Effizienz der Sprache aufzugeben.

Adressatenkreis

Primäre Zielgruppe

Das Buch wendet sich an **Programmierer** und solche, die es werden wollen. Es möchte ihnen mit der speziellen Denkweise die Programmierung erleichtern, besonders auch das Erlernen neuer Sprachen. Mit der nahe gelegten Sicht der Programmierung wird eine Sprachenunabhängigkeit gewonnen, ja man ist sogar offen für verschiedene Paradigmen der Programmierung. Mit der Erfahrung der gewonnenen Flexibilität ausgerüstet wird ein Programmierer oder eine Programmiererin *mit mehr Freude und größerer Effizienz* die Probleme der Programmierung meistern.

Das Buch wendet sich auch an **Anfänger der Programmierung**. Jedoch sollte ein *großes Interesse für die Programmierung* aufgrund einer persönlichen Eignung und Neigung vorhanden sein.

Zusammenfassend ist die primäre Zielgruppe des Buches wie folgt zu beschreiben:

> *Das Buch ist gedacht für Menschen, die einen Ausbildungsbedarf in den Grundlagen der Programmierung besitzen*.

- Dies sind *Studenten aller Fachrichtungen der Informatik* sowie Studenten der Mathematik und Physik.
- Dies sind ferner alle *Angestellten in der Industrie*, die sich, aus welchen Gründen auch immer, mit der Programmierung auseinandersetzen müssen.
- *Programmierer, die bereits programmieren können*, sich aber nicht scheuen, etwas Neues kennen zu

lernen, kommen als potentielle Nutznießer von ‘A++ im großen Stil’ gewiss ebenfalls in Betracht.

Sekundäre Adressaten

- In Fachkreisen gilt das Lehrbuch der Informatik vom Massachusetts Institute of Technology (MIT), das legendäre SICP-Buch, als eines der besten Lehrbücher der Informatik. An den deutschen Hochschulen wird dieses Buch zwar empfohlen, aber die Vorlesungen orientieren sich meistens nicht an ihm, sodass viele Studenten entmutigt werden, sich mit dem Buch auseinander zu setzen.

  Hier könnte *‘A++ in Theorie und Praxis’* eine Lücke füllen. Es ist einfacher, grundlegender und nicht so komplex wie das SICP-Buch, geht aber in dieselbe Richtung, nämlich vom Lambda-Kalkül her die Programmierung aufzubauen. Dazu kommt der größere Bezug zur Praxis durch die gleichzeitige Anwendung der Grundlagen auf fünf Programmiersprachen und die vier großen Fallstudien.

Fallstudien

1. Die **erste Fallstudie** ist ein *A++ – Interpreter*. Dies ist ein Programm, das A++ – Programmcode auswertet. Diese Fallstudie ist in diesem Buch in ARS++ implementiert[1]wird auch die Implementierung in Perl, Python, Java, C++ und C vorgestellt. Der ARS-Interpreter soll nicht nur dazu dienen, die A++ – Beispiele nachvollziehen zu können, sondern ist

gleichzeitig eine Anwendung der ARS-gemäßen Programmierung in der jeweiligen Zielsprache.

2. Die **zweite Fallstudie** ist in Java implementiert und besteht aus einem Applet („OntoSimula“), das das Leben mehrerer Philosophen simuliert. Die Fallstudie enthält ein *eigenes dynamisches Objektsystem*, das die Möglichkeiten des statischen Standard-Objektsystems von Java (das allerdings mitbenutzt wird), weit übersteigt. Diese Dynamik im Objektsystem ist gefordert, wenn es darum geht, ein lebendiges System zu simulieren, in dem es Wachstum, Veränderung, Altern und Ähnliches gibt.

Nutzen des Buches

Auf die Frage: „Was bringt es dem Leser oder der Leserin dieses Buch zu lesen; welchen Nutzen kann man aus ihm ziehen?“, können folgende Punkte eine Antwort geben:.

- Das Lesen von ‘A++ in Theorie und Praxis’ **schafft eine Vertrautheit mit den drei Paradigmen der Programmierung**, nämlich mit der funktionalen Programmierung, der objektorientierten Programmierung und der imperativen Programmierung. Leser können später selbst entscheiden, welcher Programmierstil im konkreten Fall zu bevorzugen ist. Konkreter Fall ist hier nicht identisch mit irgendeinem Projekt, denn selbst innerhalb eines Programmes kann ‘hier’ der eine Stil und ‘dort’ ein anderer Stil zu empfehlen sein.
- Da die *in ‘A++ in Theorie und Praxis’ vermittelten Grundlagen der Programmierung* sehr allgemein gehalten sind, kann man sie in den meisten

Programmiersprachen anwenden. Das bedeutet, dass Leserinnen und Leser neue Programmiersprachen leichter erlernen werden, weil sie die in ‘A++ in Theorie und Praxis’ aufgezeigten **Muster der Programmierung** in den neuen Sprachen lediglich zum Ausdruck bringen müssen.

> *Diese Muster der Programmierung sind sehr einfach, aber dennoch umfassend und mächtig. Mit ihnen können die meisten Programmieraufgaben gelöst werden.*

- In ‘A++ in Theorie und Praxis’ **lernt man nebenbei, sich ein eigenes Objektsystem aufzubauen**. Diese Technik kann *sehr hilfreich sein beim Umgang mit Programmiersprachen, die nicht objekt-orientiert sind*.

>Dieses Objektsystem ist außerdem noch ein dynamisches Objektsystem. Angewandt auf die Programmiersprache Java bedeutet dies eine *Erweiterung und Bereicherung des Standard-Objektsystems*.

Besonderer Wert des Buches

Die solide Grundlage des Lambda-Kalküls wird in ein Grundlagenstudium der Programmierung integriert,

- *ohne, dass* eine Fixierung auf spezielle funktionale Programmiersprachen erfolgt
- und *ohne, dass* dieser Versuch in eine Mathematisierung der Programmierung mündet.

In **anderen Büchern** wird *funktionale Programmierung* meistens nur im Zusammenhang mit einer ganz speziellen komplexen Programmiersprache, wie z.B. Haskell eingeführt und das *Lambda-Kalkül* wird, wenn es überhaupt als theoretische Grundlage für die funktionale Programmierung erwähnt wird, immer als logisch-mathematisches System gesehen und entsprechend in mathematischer Notation präsentiert. Es steht auf einem hohen mathematischen Podest, das immer noch für viele Programmierer abschreckend wirkt.

In **'A++ in Theorie und Praxis ' dagegen** werden die *Grundoperationen des Lambda-Kalküls*, Abstraktion, Referenz und Synthese als *einfache grundlegende Programmiersprache mit Programmbeispielen* und einem eigenen ARS-Interpreter vorgestellt.

Es ist somit die **besondere Leistung dieses Buches**, Programmierern und Programmiererinnen den Schrecken vor dem Lambda-Kalkül zu nehmen und letzteres nutzbar zu machen für den Prozess des Erlernens der Kunst des Programmierens.

Die *Zielgruppe für dieses Kapitel* und das ganze Buch ist somit *nicht eine kleine elitäre Gruppe von Wissenschaftlern, sondern alle Studenten und Studentinnen der Informatik*, weil es hier um die Grundlagen der Programmierung geht, auf einem Niveau von Programmierern und nicht von Mathematikern.

Danksagung

Dieses Buch wurde auf dem PC erstellt mithilfe *qualitativ erstrangiger Software-Tools, die von* **großen Idealisten** *der Welt* **frei** *zur Verfügung gestellt* wurden. Die primären Werkzeuge[2] sind:

1. Das 1982 von *Donald E. Knuth* veröffentlichte Textsatzsystem **T_EX** und das 1984 von *Leslie Lamport* dazu entwickelte Makropaket **LaT_EX**.
2. Die von *Thomas Esser (Universität Hannover)* für Linux erstellte Implementierung des Tex/Latex-Systems **teTeX**.
3. Das von *Linus Thorvalds (Universität Helsinki)* 1991 geschaffene PC-Betriebssystem **Linux**
4. Das *X Window System* für Linux **XFree86** mit den vielfältigen Window-Managern und Desktop-Environments (z.B. **fvwm SuSE-Desktop**, **kde**, **gnome**).
5. Die programmierbaren Texteditoren **elvis** und **vim**, die von *Steve Kirkendall* bzw. von *Bram Moolenaar* entwickelt wurden.
6. Die von *Brian Fox (Free Software Foundation)* und von *Chet Ramey (Case Western Reserve University)* erstellte **bash** (Bourne Again Shell).
7. Das von *Tomas Rokicki* (Stanford University) erstellte Konvertierungsprogramm **dvips** zum Erstellen eines Postscript-Dokumentes aus den von T_EX gelieferten dvi-Dokumenten und die von *Angus J.C. Duggan* zum Manipulieren von Postscript-Dokumenten (Skalieren der

Seiten, Anordnung der Seiten in Faszikeln, etc.) erstellten Werkzeuge **psutils**.

8. Das von *Timothy O. Theisen (University of Wisconsin-Madison)* erstellte Programm **ghostview** zum Visualisieren und Drucken von Postscript-Dokumenten.
9. Das von *Carsten Heinz* erstellte LaTeX-Makro-Paket **listings.dtx** zum Formatieren von Programmlisten für die meisten Programmiersprachen.
10. Das von *Nikos Drakos (University of Leeds)* ursprünglich erstellte und von *Ross Moore (Macquerie University, Sydney)* sowie *Marek Rouchal (Infineon Technologies AG, München)* weiterentwickelte Programm **latex2html** LaTeX-Dokumente in HTML übersetzt werden können.

Mein besonderer Dank gilt denjenigen, die Scheme geschaffen haben, nämlich **Guy L. Steele** und **Gerald J. Sussman**, sowie den vielen Autoren, die so ausgezeichnete Bücher über Scheme und andere Programmiersprachen geschrieben haben, wie **Jacques Chazarain**, mit seinem Buch *Programmer Avec Scheme*, das ich für mich persönlich als eines der besten Lehrbücher der Informatik halte, sowie **Peter Norvig** und **Samuel N. Kamin**. (Siehe hierzu [13], [70] und [48]).

Alonzo Church, dem wir das *Lambda-Kalkül* verdanken, sei ganz besonderer Dank, obwohl er selbst natürlich nichts davon ahnte, dass er einen so großen Beitrag zur Informatik leistete, dass er für die Theorie der Programmiersprachen eine Grundlage geschaffen hatte. Der Dank erstreckt sich selbstverständlich auch auf **alle anderen Autoren**, die in dem *bibliographischen Anhang* aufgeführt sind.

Auch **Brent Benson** gebhrt mein Dank, der es mir ermöglicht hat mit Hilfe seiner Bibliothek *‘libscheme’* die Programmiersprache **ARS++** zu schaffen, die aus **A++** mit Hilfe der vorgegebenen, aus *libscheme importierten Primitivoperationen*, eine Programmiersprache der Praxis macht, angereichert durch über Scheme und ‘libscheme’ hinausgehende Erweiterungen, um so den praktischen Anforderungen der Fallstudien voll gerecht werden zu können.

Ähnlichen Dank schulde ich **Hans J. Boehm**, der mit seinem ‘black-box’ Garbage-Collector für C und C++ Programme, die Komplexität der Fallstudien für dieses Buch auf ein vertretbares Maß herunterschrauben half.

Was wäre schließlich die Welt ohne **Richard Stallman**, dem wir Programmierer nicht nur den Emacs sondern auch den *GNU-C-Compiler* verdanken und all den Reichtum, den die von ihm gegründete *Free Software Foundation* der Welt anzubieten hat! Auch dieses Buch hätte ohne ihn und die **vielen Programmierer**, die ihre Software *als freie Software veröffentlichen*, nicht realisiert werden können!

Möge dieses Buch Programmierern das Faszinierende von ARS entdecken helfen und sie zu einem souveränen Umgang mit den Elementarteilchen der Programmierung führen und sie dadurch befähigen, mit größerer Freude und Produktivität die Herausforderungen der Programmierung zu meistern.

Georg P. Loczewski Groß-Zimmern, im Mai 2018

Inhaltsverzeichnis