Vorwort

All problems in computer science
can be solved by another level of indirection,
except for the problem of too many layers of indirection.
– David J. Wheeler

C++ fühlt sich wie eine neue Sprache an. Das heißt, ich kann in C++11 meine Ideen klarer, einfacher und direkter ausdrücken als ich es in C++98 konnte. Darüber hinaus werden die Programme durch den Compiler besser überprüft und laufen auch schneller.

In diesem Buch strebe ich nach *Vollständigkeit*. Ich beschreibe alle Sprachfeatures und Komponenten der Standardbibliothek, die ein professioneller Programmierer wahrscheinlich braucht. Für jedes dieser Elemente gebe ich Folgendes an:

- *Grundprinzip:* Welche Probleme soll es lösen helfen? Welche Prinzipien liegen dem Design zugrunde? Was sind die grundsätzlichen Beschränkungen?
- *Spezifikation:* Wie ist es definiert? Dabei ist die Detailebene auf den erfahrenen Programmierer ausgerichtet; der angehende Sprachanwalt kann den vielen Verweisen zum ISO-Standard folgen.
- *Beispiele:* Wie kann man es an sich und in Kombination mit anderen Features sinnvoll einsetzen? Wie sehen die Schlüsseltechniken und Idioms aus? Welche Bedeutung hat es für Wartbarkeit und Performance?

Die Verwendung von C++ hat sich im Lauf der Jahre drastisch gewandelt und das Gleiche gilt auch für die Sprache selbst. Aus dem Blickwinkel eines Programmierers sind die meisten Änderungen als Verbesserungen einzustufen. Das aktuelle ISO-Standard-C++ (ISO/IEC 14882-2011, üblicherweise C++11 genannt) ist ein weit besseres Tool für Qualitätssoftware als die vorherigen Versionen. Worin äußert sich dies? Welche Programmierstile und -techniken unterstützt modernes C++? Welche Features der Sprache und der Standardbibliothek unterstützen diese Techniken? Was sind die Bausteine von elegantem, korrektem, wartbarem und effizientem C++-Code? Dies sind die Schlüsselfragen, die in diesem Buch beantwortet werden. Viele Antworten sind nicht die gleichen, wie Sie sie für Vintage-C++ der Jahre 1985, 1995 oder 2005 finden: Der Fortschritt ist nicht aufzuhalten.

C++ ist eine universelle Programmiersprache, die den Entwurf und die Verwendung von typreichen, kompakten Abstraktionen betont. Die Sprache ist besonders für ressourcenbeschränkte Anwendungen geeignet, wie man sie beispielsweise in Softwareinfrastrukturen findet. Es lohnt sich für den Programmierer, Zeit in das Erlernen der Techniken für qualitätsgerechten Code zu investieren. C++ ist eine Sprache für jemanden, der Programmierung ernst nimmt. Unsere Zivilisation hängt entscheidend von Software ab; es wäre also besser, wenn es sich dabei um Qualitätssoftware handelt.

Es gibt Milliarden von Zeilen in C++-Code. Dabei wird besonderer Wert auf Stabilität gelegt, sodass der C++-Code von 1985 und 1995 immer noch funktioniert und auch für weitere Jahrzehnte funktionieren wird. Allerdings können Sie mit modernem C++ besser arbeiten; wenn Sie sich an die älteren Stile klammern, werden Sie Code von geringerer Qualität und schlechterer Leistung schreiben. Die Betonung von Stabilität bedeutet auch, dass standardkonformer Code, den Sie heute schreiben, auch noch in einigen Jahrzehnten funktionieren wird. Der gesamte Code in diesem Buch ist zum 2011-ISO-C++-Standard konform.

Dieses Buch richtet sich an drei Leserkreise:

- C++-Programmierer, die wissen möchten, was der neueste ISO-C++-Standard zu bieten hat,
- C-Programmierer, die daran interessiert sind, was C++ über C hinaus bietet, und
- Programmierer, die von Anwendungssprachen wie zum Beispiel Java, C#, Python und Ruby kommen und nach etwas suchen, das „näher an der Maschine“ dran ist – etwas, was flexibler ist, was eine bessere Prüfung zur Übersetzungszeit realisiert oder was eine bessere Performance bietet.

Natürlich gibt es keine klare Trennung zwischen diesen drei Gruppen – ein professioneller Softwareentwickler beherrscht schließlich mehr als nur eine Programmiersprache.

In diesem Buch wird davon ausgegangen, dass die Leser Programmierer sind. Wenn Sie fragen: „Was ist eine **for**-Schleife?“ oder „Was ist ein Compiler?“, dann ist dieses Buch (noch) nichts für Sie; stattdessen empfehle ich mein *„Principles and Practice Using C++ to get started with programming and C++“*. Darüber hinaus nehme ich an, dass die Leser bereits eine gewisse Reife als Softwareentwickler haben. Wenn Sie fragen: „Warum sich mit Testen abmühen?“ oder sagen „Alle Sprachen sind prinzipiell gleich; zeige mir nur die Syntax“, vielleicht auch überzeugt davon sind, dass es genau eine Sprache gibt, die für sämtliche Aufgaben ideal ist, dann ist dieses Buch ebenfalls nichts für Sie.

Welche Features bietet C++11 gegenüber C++98 und darüber hinaus? Ein Maschinenmodell, das für moderne Computer mit jeder Menge Parallelität geeignet ist. Sprach- und Standardbibliotheksinstrumente für nebenläufige Programmierung auf Systemniveau (z.B. mithilfe von Mehrkernprozessoren). Verarbeitung regulärer Ausdrücke, Ressourcenverwaltungszeiger, Zufallszahlen, verbesserte Container (einschließlich Hashtabellen) und vieles mehr. Allgemeine und einheitliche Initialisierung, eine einfachere **for**-Anweisung, Verschiebesemantik, grundlegende Unicode-Unterstützung, Lambda-Ausdrücke, allgemeine konstante Ausdrücke, Kontrolle über Standardwerte von Klassen, variadische Templates, benutzerdefinierte Literale und mehr. Denken Sie bitte daran, dass diese Bibliotheken und Sprachfeatures dafür da sind, Programmiertechniken für die Entwicklung von Qualitätssoftware zu unterstützen. Kombinieren Sie diese – wie Bausteine aus einem Baukasten –, um ein konkretes Problem zu lösen, anstatt sie einzeln in relativer Isolation zu verwenden. Ein Computer ist eine universelle Maschine und mit C++ machen Sie dessen Kapazität nutzbar. Insbesondere ist C++ konzeptionell darauf ausgelegt, genügend flexibel und allgemein zu sein, um mit zukünftigen Problemen umgehen zu können, von denen die Designer der Sprache noch nicht einmal geträumt haben.

Danksagung

Außer den Leuten, die in den Danksagungen der vorherigen Ausgaben erwähnt wurden, möchte ich Pete Becker, Hans-J. Boehm, Marshall Clow, Jonathan Coe, Lawrence Crowl, Walter Daugherty, J. Daniel Garcia, Robert Harle, Greg Hickman, Howard Hinnant, Brian Kernighan, Daniel Krügler, Nevin Liber, Michel Michaud, Gary Powell, Jan Christiaan van Winkel und Leor Zolman danken. Ohne ihre Hilfe wäre dieses Buch mit Sicherheit nicht so gut geworden.

Dank auch an Howard Hinnant für die Beantwortung vieler Fragen zur Standardbibliothek.

Andrew Sutton ist der Autor der Origin-Bibliothek, die als Testumgebung für die Emulation von Konzepten in den Template-Kapiteln dient, und der Matrix-Bibliothek, die Thema von Kapitel 29 ist. Die Open-Source-Bibliothek Origin finden Sie im Web, wenn Sie nach „Origin“ und „Andrew Sutton“ suchen.

Ein Dank geht auch an meine Design-Diplomanden, die mehr Probleme mit den „Tour-Kapiteln“ als sonst jemand gefunden haben.

Hätte ich sämtlichen Hinweisen meiner Rezensenten folgen können, wäre das Buch noch besser, doch auch Hunderte Seiten länger geworden. Jeder Fachlektor hat zusätzliche technische Details vorgeschlagen, erweiterte Beispiele und viele nützliche Entwicklungskonventionen; von jedem neuen Rezensenten (oder Lehrer) kamen Vorschläge für ergänzende Beispiele; und viele Rezensenten haben (zu Recht) eingeschätzt, dass das Buch zu umfangreich werden könnte.

Danken möchte ich der Computer Science Department der Princeton University und speziell Prof. Brian Kernighan, die mich für einen Teil des Sabbaticals aufgenommen haben, was mir Zeit verschaffte, dieses Buch zu schreiben. Aus dem gleichen Grund sei dem Computer Lab der Cambridge University und speziell Prof. Andy Hopper gedankt.

College Station, Texas *Bjarne Stroustrup*