

1 Vorbemerkungen

1.1 Zur 2. Auflage

Sechs Jahre sind vergangen seit der ersten Auflage – mehr als drei Jahre seit dem Nachdruck – und die Zeit ist reif für eine zweite, vollständig überarbeitete Auflage.

Vieles hat sich in der Zwischenzeit geändert, aber im Wesentlichen ist die Grundlage des ersten Buchs, der Draft ANSI/ISO-Standard aus dem Jahr 1996, tatsächlich mehr oder weniger unverändert zum offiziellen Standard geworden. Fehler wurden beseitigt, das Buch wurde in den wesentlichen Teilen überarbeitet, einzelne Abschnitte Änderungen unterzogen und an den aktuellen Sprachstandard angepasst. Zahlreiche Anmerkungen und Anregungen von Lesern wurden eingearbeitet. Stellvertretend möchte ich hier vor allem Prof. Eisenecker für seine Kommentare danken.

Dem aktuellen Trend folgend, liegt die der ersten Auflage beigefügte CD-ROM dem Buch nicht mehr bei, die entsprechenden Informationen und Beispiele können aber *online* bezogen werden (siehe dazu Abschnitt 1.8 auf Seite 6). Damit ist es auch möglich, den Lesern Verbesserungen einfach und schnell zur Verfügung zu stellen.

1.2 Zum Nachdruck

Inzwischen hat sich der C++-Standard etabliert. Für den ersten Nachdruck wurde das Buch in zahlreichen Punkten verbessert, viele Fehler und Unklarheiten beseitigt. Ich habe mich bemüht, die zahlreichen Anregungen der Leser zu berücksichtigen und einzuarbeiten. An dieser Stelle vielen Dank an alle, die mir halfen, das Buch zu verbessern!

1.3 Zur 1. Auflage

Das vorliegende Buch beschreibt die Programmiersprache C++ nach dem ANSI/ISO-Standard. Zum Zeitpunkt der Entstehung dieses Buchs (Sommer/Herbst 1996) ist der Standard allerdings noch nicht verabschiedet. Es gibt daher kein offizielles Dokument, das den endgültigen Standard be-

schreibt. Alle existierenden Dokumente sind entweder *Drafts*, das heißt vorläufig, oder aber inoffizielle Beschreibungen.

Ein Buch über etwas schreiben zu wollen, das noch nicht existiert, wirft eine Reihe von Problemen auf: Die zur Verfügung stehenden Informationen sind meist vage und unsicher und es gibt noch keine Entwicklungssysteme, die genau dem kommenden Sprachstandard entsprechen usw.

Die Bedeutung des Sprachstandards ist aber absehbar. Über kurz oder lang werden alle C++-Compiler dieser Sprachdefinition folgen. Es wäre also widersinnig, den existierenden 2147618 Büchern über C++ ein weiteres hinzuzufügen, das nicht auf die kommenden Sprachfeatures eingeht. Zudem sind die *wesentlichen* neuen Sprachelemente schon seit einiger Zeit bekannt und werden sich nicht mehr verändern.

Die folgenden Abschnitte sollen kurz in die Thematik und den Aufbau des Buchs einführen.

1.4 Was das Buch nicht ist ...

Als Erstes soll erläutert werden, was dieses Buch *nicht* ist:

- ❑ Das Buch ist keine komplette und umfassende Referenz für die Programmiersprache C++ nach dem ANSI/ISO-Standard. Die Standardbibliothek von C++ weist inzwischen einen derart großen Umfang auf, dass sie nur in sehr geringem Ausmaß berücksichtigt werden konnte (Ein-/Ausgabe). Wesentliche Bereiche der Bibliothek wie die unter dem Namen *STL* (*Standard Template Library*) bekannten generischen Algorithmen und Datenstrukturen wurden ausgeklammert. Zum Erlernen der Programmiersprache und der wesentlichen Konzepte sind sie zum einen nicht notwendig, und zum anderen erfordert ihre Komplexität eine umfangreiche Abhandlung, wie sie in diesem Buch nicht möglich wäre. Der Schwerpunkt dieses Buchs liegt auf der Vermittlung des (nach Meinung des Autors) für C++-*Einsteiger* und -*Anfänger* wesentlichen Wissens.
- ❑ Das Buch ist kein herstellerepezifisches C++-Lehrbuch, sondern bezieht sich ausschließlich auf den ANSI/ISO-Standard. Die hier besprochenen Konzepte sind damit auf allen ANSI/ISO-kompatiblen Entwicklungssystemen gültig, die hier angegebenen Programme können unabhängig von Compiler und Plattform übersetzt werden (vorausgesetzt, der Compiler entspricht dem Standard).
- ❑ Das Buch setzt keine C-Kenntnisse des Lesers voraus. Der Autor ist der Meinung, dass C++ eine eigenständige Programmiersprache ist, die auch als solche gelehrt werden sollte. C-Kenntnisse sind nützlich, aber nicht erforderlich.

- Das Buch konzentriert sich zudem ausschließlich auf den Bereich der Programmierung. Andere Themenbereiche wie etwa die objektorientierte Analyse, der Entwurf von objektorientierten Systemen oder Entwurfsmuster (*Design Patterns*) werden in diesem Buch daher nicht erläutert.
- Weiterführende Themen wie *Handles*, *Master Pointer*, Speichermanagement und dergleichen werden ebenfalls nicht behandelt. Diese Themenbereiche haben eine andere Zielgruppe und setzen ein bereits gefestigtes Wissen um die Sprache C++ voraus.

Die in Anhang C angeführte Übersicht bietet dem Leser eine (subjektive) Auswahl von C++-Titeln, die auch weiterführende Themenbereiche abdecken.

1.5 Aufbau des Buchs

Der Aufbau dieses Buchs unterscheidet sich bewusst von dem der meisten anderen C++-Lehrbücher. Durch seine Konzeption als Lehrbuch für C++-Einsteiger ist es unabhängig von den verschiedenen Entwicklungssystemen und -plattformen. Im Vordergrund steht nicht das Entwicklungssystem XY der Firma Z, sondern vielmehr die *Sprache* C++.

Dabei wurde versucht, das Buch nach einem alten österreichischen Schilehrer-Grundsatz (»Vom Einfachen zum Komplexen«) zu gliedern. Die ersten Kapitel beschäftigen sich daher mit den »einfachen« Aspekten der Sprache: Sprachelemente & Grammatik, Basisdatentypen, Ausdrücke und so weiter bis hin zu C++-Modulen. Diese Grundlagen mögen zwar (vor allem für Leser mit C-Erfahrung) langatmig erscheinen, doch eine gründliche und für den Leser nutzbringende Abhandlung der wesentlichen Konzepte ist nur möglich, wenn die »einfachen« Sprachelemente klar erläutert sind.

Leser, die schon über entsprechende Erfahrungen verfügen, können sich bei den einführenden Kapiteln (Kapitel 2 bis einschließlich 6) auf eine auszugswise Lektüre beschränken.

Im Detail ist das Buch wie folgt aufgebaut:

- Kapitel 2 bringt eine kurze Einführung in die Sprache C++ anhand von einigen kleinen Programmen. Im Vordergrund steht dabei nicht die Erläuterung von Sprachkonzepten, sondern vielmehr die Vorstellung des Schemas von einfachen C++-Programmen. Diese einfachen Programme können vom Leser abgeändert und getestet werden. So ist er in der Lage, von Anfang an Codebeispiele besser verstehen und eigene kleine Programme erstellen zu können.
- Kapitel 3 beschreibt, aus welchen Zeichen und Zeichenfolgen (lexikalischen Elementen) C++-Programme aufgebaut sind.

Vom Einfachen zum Komplexen

Einfache Programme

Lexikalische Elemente

- Basisdatentypen und Deklarationen* □ Kapitel 4 stellt die grundlegenden Datentypen von C++ vor. Auf diesen einfachen Datentypen bauen die höheren und strukturierten Datentypen auf.
- Ausdrücke und Operatoren* □ Datenelemente werden in Ausdrücken, die aus Operanden und Operatoren bestehen, verarbeitet. In Kapitel 5 werden die verschiedenen Arten von Ausdrücken sowie die einzelnen Operatoren vorgestellt.
- Anweisungen* □ Grundelemente von C++-Programmen sind Anweisungen. Anweisungen enthalten Ausdrücke, deklarieren Objekte und steuern den Ablauf von Programmen. Kapitel 6 beschreibt die verschiedenen Arten von Anweisungen.
- Funktionen* □ Eine spezielle Anweisung ist die Funktionsanweisung. Sie ist das erste (und einfachste) Strukturierungs- und Abstraktionsmittel von C++ und fasst mehrere Anweisungen zu einer »Super-Anweisung« zusammen. Kapitel 7 beschreibt Funktionen im Allgemeinen sowie verschiedene C++-Spezifika wie Operatorfunktionen, das Konzept des Überladens oder vorbelegte Funktionsparameter.
- Höhere Datentypen* □ Aufbauend auf diesen Kapiteln werden in Kapitel 8 die höheren Datentypen (Referenzen, Zeiger und Vektoren) im Detail besprochen. Neben den eigentlichen Datentypen steht dabei auch ihr Einsatz (zum Beispiel die Verwendung von Referenzen bei der Parameterübergabe) im Vordergrund. Auch die strukturierten Datentypen (Klassen) werden überblicksmäßig besprochen.
- Gültigkeitsbereiche, Deklarationen, Typumwandlungen* □ Kapitel 9 vertieft einige bisher nur oberflächlich behandelte Aspekte: Gültigkeitsbereiche, Deklarationen und die verschiedenen Möglichkeiten der Typumwandlung in C++.
- Module* □ Der »traditionelle« Teil des Buchs wird durch einen Abschnitt über die modulbasierte Programmierung in C++ beschlossen (Kapitel 10). Das Kapitel beschreibt die Realisierung von Modulen in C++ und betont insbesondere die Bedeutung des Geheimnisprinzips.
- Klassen* □ Kapitel 11 ist das »Kernkapitel« des Buchs und erläutert das Klassenkonzept. Die Realisierung von Klassen und Themenbereichen wie Speichermanagement, Überladen von Operatoren, kanonische Form von Klassen etc. werden in diesem Kapitel anhand eines durchgängigen Beispiels demonstriert.
- Templates* □ Kapitel 12 beschreibt das Konzept der Generizität in C++. Besprochen werden generische Funktionen (Funktions-Templates) und Klassen (Klassen-Templates).
- Vererbung* □ Kapitel 13 zeigt die Realisierung von Vererbung in C++. Dabei stehen zunächst nur die »statischen« Aspekte der Vererbung im Vordergrund.
- Polymorphismus* □ Darauf aufbauend werden in Kapitel 14 die dynamischen Aspekte (Polymorphismus, abstrakte Basisklassen) sowie verschiedene andere weiterführende Themenbereiche (Laufzeit-Typinformation, Mehrfachvererbung) besprochen.

- Kapitel 15 beschließt das Buch mit einer Beschreibung des Konzepts der Ausnahmebehandlung. *Ausnahmebehandlung*

In den Anhängen sind verschiedene ergänzende Abschnitte angeführt, die vom Leser je nach Bedarf durchgearbeitet werden können: *Anhang*

- Anhang A stellt die Ein-/Ausgabe in C++ kurz und überblicksweise vor.
- Anhang B beschreibt den C++-Präprozessor.
- Anhang C behandelt die »Hauptfunktion« von C++-Programmen sowie die Parameterübergabe an diese.
- Anhang C führt in einer kurzen Übersicht eine Reihe von empfehlenswerten C++-Büchern an.

1.6 Schreibweisen und Konventionen

Schreibweisen wie zum Beispiel *der* Anwender werden in einer allgemeinen und geschlechtsneutralen Bedeutung verwendet.

Kursive Schreibweisen heben Begriffe hervor.

Englischsprachige Begriffe sind ebenfalls kursiv gedruckt. Setzen sie sich aus mehreren Wörtern zusammen, so werden sie nicht durch Bindestriche verbunden (*Call by Value* anstelle von *Call-by-Value*).

Codestücke und Schlüsselwörter sind im Zeichensatz `Typewriter` gesetzt.

Zitate erfolgen wie üblich unter Angabe der Quelle und Seitenzahl. Lediglich in den entsprechend aufbereiteten Quellen werden die Stellen über logische Namen angegeben (zum Beispiel [ISO 1995, dcl.dcl]), da diese unabhängig von den konkreten Seitenzahlen sind.

Zwei besondere Absatzformate sind zu erwähnen. Beide kennzeichnen spezielle Sachverhalte und sind durch eine Grafik am äußeren Rand vom restlichen Text abgegrenzt:

Wird im Text auf eine Tatsache besonders hingewiesen, so ist der entsprechende Textabschnitt durch ein Ausrufezeichen am Rand speziell gekennzeichnet.

!

Spezielle Tipps hingegen werden durch seitliche »Striche« markiert. Im Gegensatz zu den Anmerkungen sind diese Tipps »subjektive« Hinweise des Autors.



1.7 Übungsbeispiele und Entwicklungssysteme

Übung Programmieren erlernt man nicht durch das Studium von Büchern oder Handbüchern, sondern nur durch Übung. Dieses Buch ist lediglich ein »Hilfsmittel« und bereitet die Programmiersprache C++ für den Leser auf. Die Umsetzung der Konzepte bei konkreten Problemstellungen aber obliegt dem Leser.

Um Möglichkeiten der Umsetzung und die Anwendung der besprochenen Konzepte aufzuzeigen, werden daher Übungsbeispiele verwendet. Dabei handelt es sich weder um theoretische Fragen bezüglich der Syntax oder andere Problembereiche von C++ noch um umfangreiche Beispiele, deren Lösungen nur auszugsweise angeführt werden können. Vielmehr werden Beispiele verwendet, deren konkrete (und vor allem vollständige) Lösungen dem Leser die Anwendung der besprochenen Sprachkonzepte zeigen.

Übungsaufgaben Übungsaufgaben werden ab Kapitel 11 (Klassen) gestellt und sind durch kurze Angaben, die bewusst vage und offen gehalten sind, spezifiziert. Die Angaben geben dem Leser damit großen Freiraum in Bezug auf die Realisierung. Jedes einzelne Beispiel ist vollständig in dem Sinne, dass eine Klasse oder eine Reihe von Klassen inklusive Testprogramm zu erstellen ist.

Die Aufgabenstellungen stammen aus Programmierkursen im universitären und außeruniversitären Bereich und wurden in den abgebildeten Formen von Kursteilnehmern gelöst.

1.8 Lösungen und *online*-Beispiele

Die Lösungen der Beispiele können *online* bezogen werden. Sie können über die Webseiten des Verlags (<http://www.dpunkt.de>) oder direkt über die Seiten des Autors (<http://www.silbergrau.at/cplusplus/>) bezogen werden.

ANSI/ISO-Standard Die einzelnen Aufgaben sind bewusst allgemein gehalten und sollten auf jedem C++-Entwicklungssystem, das dem ANSI/ISO-Standard weitgehend entspricht, übersetzt werden können.

Zu beachten ist, dass die angeführten Lösungen keineswegs *die* idealen Lösungen der Beispiele darstellen. Sie sind vielmehr *mögliche* Lösungen, die dem Leser zeigen sollen, wie bestimmte Konzepte umgesetzt werden können. Die Lösungen enthalten eine allgemeine Lösungsidee sowie Auszüge aus den wesentlichen Passagen und – vor allem – den vollständigen Sourcecode.

Da es sich bei allen Aufgaben um vollständige Beispiele handelt, kann der Leser experimentieren: Er kann die Dateien übersetzen und abändern, er kann die Programme ausführen oder *debuggen* und so den genauen Ablauf verfolgen.

2 Einführung

Dieses Kapitel gibt einen kurzen Überblick über die Entwicklung von C++ und führt anschließend anhand von einigen kleinen Beispielen grob und unvollständig in die allgemeine Struktur von C++-Programmen ein. Dieser Abschnitt soll helfen, die verwendeten Programmbeispiele in den ersten Kapiteln besser verstehen zu können.

2.1 Die Programmiersprache C++

Die folgenden zwei Abschnitte beschreiben kurz und überblicksmäßig die Programmiersprache C++ sowie ihre Entwicklung.

2.1.1 Grundlagen von C++

C++ wurde von Bjarne Stroustrup [Stroustrup 2000] entwickelt und basiert auf der Programmiersprache C, die Anfang der siebziger Jahre (1971) für das Betriebssystem UNIX entwickelt wurde und Bestandteil dessen Programmierumgebung ist. Die klassische C-Sprachdefinition wurde 1978 von Brian Kernighan und Dennis Ritchie vorgelegt [Kernighan & Ritchie 1978], fünf Jahre später begannen das *American National Standards Institute* (ANSI) und die *International Standardization Organization* (ISO) einen neuen Standard festzulegen.

Die zweite Grundlage für C++ ist Simula67. Von dieser Sprache wurde das Klassenkonzept mit abgeleiteten Klassen und virtuellen Funktionen entliehen.

Simula67

Die Gründe, warum Stroustrup C als Basissprache für C++ wählte, sind:

- C ist vielseitig und prägnant.
- C ist effizient.
- C eignet sich für die meisten Aufgaben in der Systemprogrammierung.
- C ist auf praktisch allen Rechnersystemen verfügbar und einfach portierbar.
- C ist Bestandteil der UNIX-Programmierumgebung.

- Es existiert eine Vielzahl an Bibliotheksroutinen und Software-Utilities, die in C geschrieben sind und von C++ genutzt werden können.

C++ ist daher weitgehend kompatibel zu C. Die Kompatibilität zu C hat aber nicht nur Vorteile, wie Programm 2.1 zeigt.

Programm 2.1

Acht-Damen-
Problem
[Libes 1993]

```
v,i,j,k,l,s,a[99];
int main() {
    for(scanf("%d",&s);*a-s;v=a[j*=v]-a[i],k=i<s,j+=(v=j<
s&&(!k&&!printf(2+"\n\n%c"-(!l<<!j),"#Q"[l^v?(l^j)
&l:2])&&+1||a[i]<s&&v&&v-i+j&&v+i-j))&&(l%=s),v|
(i==j?a[i+=k]=0:++a[i])>=s*k&&+a[--i]);
}
```

Dieses Programm löst das Problem der acht Damen, funktioniert aber auch mit 4-99 Damen. Als »Damenproblem« wird die Aufgabe bezeichnet, acht Damen so auf einem Schachbrett zu platzieren, dass keine der Damen eine der anderen »schlagen kann«.

Das Programm ist ein durchaus legales C-Programm:

»The authors note, that they use 'no preprocessor statements and no ifs, breaks, cases, functions, gotos, structures ... In short, it contains no C language that might confuse the innocent'«. [Libes 1993, S. 301]

Tatsächlich ist das Programm 1990 als Sieger beim jährlichen *Obfuscated C Code Contest* hervorgegangen – Kategorie *Best Small Program*.

Stroustrup ist für die enge Anlehnung von C++ an C oft und hart kritisiert worden. Der Erfolg von C++ aber bestätigt diese Entscheidung.

2.1.2 Die Entwicklung von C++

1979: *C with Classes*

1979 begann Bjarne Stroustrup seine Arbeit an *C with Classes* als ein Resultat seiner Probleme im Zusammenhang mit einer großen Simulationsaufgabe, die in Simula entwickelt wurde. *C with Classes* war eine erweiterte Version von C, die zusätzlich folgende Sprachelemente umfasste:

- Klassen
- Vererbung ohne Polymorphismus
- Konstruktoren und Destruktoren
- Funktionen
- friend-Deklarationen
- Typprüfung

Die Implementierung von *C with Classes* bestand in einem Präprozessor, der *C with Classes*-Programme in C-Programme übersetzte.

Ab 1982 führte Stroustrup die Entwicklung von *C with Classes* fort. Der Entwicklungsprozess mündete schließlich in der Version 3.0 von C++, die den im *The Annotated C++ Reference Manual (ARM)* [Ellis & Stroustrup 1990] beschriebenen Sprachstandard implementierte. C++ umfasste in dieser Version folgende neue Features:

1982: C++

ARM

- Virtuelle Funktionen
- Überladen von Funktionen und Operatoren
- Referenzen
- Konstanten
- Speichermanagement
- Templates

Im Jahr 1991 fand das erste Treffen der *ISO Workgroup 21* statt. Die Arbeit des Standardisierungskomitees mündete 1995 in einem so genannten *Draft Standard* [ISO 1995]. Im Laufe des Jahres 1998 wurde der C++-Standard (ISO/IEC 14882) verabschiedet. Gegenüber der Version 3.0 wurde C++ im Wesentlichen um folgende Features erweitert:

1995: Draft Standard
C++

- Ausnahmebehandlung (*Exception Handling*)
- Laufzeit-Typinformationssystem (*Run-Time Typ Information*)
- Namensräume (*Name Spaces*)
- Neue Möglichkeiten der Typumwandlung

2.2 Einfache C++-Programme

Die im Folgenden angeführten einfachen C++-Programme sollen dem Leser über die »Tristheit« der Grundlagen in den ersten Kapiteln hinweghelfen. Sie sollen ihm ermöglichen, kleine und einfache Testprogramme zu erstellen. Dabei ist es nicht notwendig, dass jede Anweisung der Beispielprogramme bis ins letzte Detail verstanden wird.

Im einfachsten Fall besteht ein C++-Programm aus einer Textdatei, die den Programmcode (*Source*) enthält. Diese Datei muss vom Compiler übersetzt werden. Der Compiler erzeugt dabei ein so genanntes *Object File*. Ein *Object File* enthält die für den Computer »übersetzten« Informationen der Programmdatei. In der Regel ist der Name eines *Object Files* gleich dem Namen des *Source Files* mit der Endung `.obj` (MSDOS, Windows oder OS/2) beziehungsweise `.o` (UNIX).

Kompilieren

Object Files können noch nicht ausgeführt werden. Jedes Programm benötigt bestimmte spezielle Codeteile für die Ausführung, die nicht extra

vom Programmierer erstellt werden müssen. Außerdem wird in Programmen im Allgemeinen eine Reihe von bereits vordefinierten Funktionen und anderen Objekten verwendet, deren Definitionen nicht im *Source File* enthalten sind (zum Beispiel die Ein-/Ausgaben). Diese sind in bereits fertig übersetzten Teilen, den so genannten »Bibliotheken« (*Libraries*), abgelegt.

Begriff Objekt

Anmerkung: Der Begriff »Objekt« wird in den ersten Kapiteln *nicht* im objektorientierten Sinne verwendet. Vielmehr bezeichnet »Objekt« vorerst ganz allgemein ein »Ding« im Speicher.

Linken

Um aus einem *Object File* ein ausführbares Programm zu machen, ist es daher nötig, dieses mit den benötigten Bibliotheken zu *linken* (Deutsch: *binden*). Erst durch diesen Schritt wird ein ausführbares Programm erzeugt.

2.2.1 Das erste Programm: Hello world!

Wie bereits oben erwähnt, kann ein C++-Programm aus einer einzelnen Datei bestehen, die die »Hauptfunktion« `main` und gegebenenfalls weitere Funktionen enthält. `main` wird beim Start eines C++-Programms automatisch aktiviert.

Hello world!

Das traditionell erste zu erstellende C++-Programm (und zugleich eines der einfachsten) gibt den Text »Hello world!« aus:

Programm 2.2

Hello world!

```
//File: hello.cpp
// Hello World

// Schnittstelle von iostream inkludieren
// iostream ist ein vordefiniertes Modul, daher wird
// der Name mit den Zeichen <> begrenzt
#include <iostream>
using namespace std;

// "Hauptprogramm"
void main()
{
    cout << "Hello world!";
}
```

Einige Bemerkungen zum Programm:

Bibliotheken

- Die Ein-/Ausgabe ist in C++ in einer eigenen »Bibliothek« (einer Sammlung von bereits vorgefertigten Programmteilen) vereinbart. Um Daten einlesen beziehungsweise ausgeben zu können, muss dem Compiler mitgeteilt werden, dass diese Bibliothek verwendet wird. Dies erfolgt durch die Anweisung `#include <iostream>`. Die

Anweisung `using namespace std` stellt den *Namensraum* `std` zur Verfügung. Wird diese Anweisung nicht verwendet, müssen alle Elemente dieses Namensraumes *qualifiziert* werden (zum Beispiel `std::cout` anstelle von `cout`).

- `main` ist die »Hauptfunktion« eines jeden C/C++-Programms und wird vom C++-Laufzeitsystem beim Programmstart automatisch aktiviert. *main*
- `Hello world!` benutzt das in allen C++-Entwicklungssystemen vorhandene Modul `iostream`, um den Text auszugeben. `iostream` realisiert eine einfache textuelle Ein-/Ausgabe. *Ein-/Ausgabe*
Grundsätzlich erfolgen alle Eingaben im Programm über Anweisungen der folgenden Art: *Eingaben*

```
cin >> Argument;
```

Dabei wird der Wert des Arguments unter Verwendung des Operators `>>` (»lesen von«, *get from*) vom Standard-Eingabestrom `cin`, also von der Tastatur, eingelesen.

Die Ausgabe erfolgt mit Anweisungen der Art *Ausgabe*

```
cout << Argument;
```

Der Wert des Arguments wird mit Hilfe des Ausgabeoperators `<<` (»schreiben nach«, *put to*) auf den Standardausgabestrom `cout` (und damit auf den Bildschirm) geschrieben.

Sowohl Eingabeoperator `>>` wie auch Ausgabeoperator `<<` können »kaskadiert« werden:

```
cout << "Hello " << "world!"; cin >> i >> j;
```

2.2.2 Variablen und Kontrollstrukturen

Das folgende Programm löst eine Gleichung der Form $ax^2 + bx + c = 0$. Das Programm liest die Koeffizienten a , b und c ein und ermittelt anschließend das Ergebnis.

Die Lösung einer derartigen Gleichung kann nach der Formel

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

erfolgen, wobei sich p und q aus der umgeformten Gleichung ergeben ($x^2 + px + q = 0$):

$$p = \frac{b}{a}, q = \frac{c}{a}$$

Programm 2.3*Quadratische
Gleichung*

```
#include <iostream>
#include <cmath>
using namespace std;

void main() {
    float a, b, c;
    float help, disk;

    cout << "Loesen einer Gleichung der Form"
         << " a*x*x+b*x+c=0" << endl;
    cout << "a="; cin >> a;
    cout << "b="; cin >> b;
    cout << "c="; cin >> c;

    // Loesung wenn a!=0
    if (a!=0) {
        help=-0.5*b/a;
        disk=help*help-c/a;
        if (disk==0) {           // Eine (reelle) Loesung
            cout << "x = ";
            cout << help;
        } else if (disk>0) {    // Zwei reelle Loesungen
            cout << "\nx1=" << help+sqrt(disk);
            cout << "\nx2=" << help-sqrt(disk);
        } else {                // Zwei komplexe Loesungen
            cout << "\nx1=" << help << '+' << sqrt(-disk);
            cout << "\nx2=" << help << '-' << sqrt(-disk);
        }
    } else if (b!=0) {         // b=0: Division nicht erlaubt
        cout << "x = " << -c/b << endl;
    } else if (c!=0) {
        cout << "Keine Loesung!" << endl;
    } else {
        cout << "Unendlich viele Loesungen!" << endl;
    }
}
```

Einige Erklärungen zum Programm:

- ❑ `cmath` ist ein Modul, das verschiedene mathematische Operationen bereitstellt. Im Programm wird `sqrt` (Quadratwurzel) verwendet.
- ❑ Die Koeffizienten werden im Programm durch Variablen repräsentiert. Variablen sind – so wie in der Mathematik – Größen, deren Wert »variabel« ist und sich im Laufe des Programms verändern kann. Jeder Va-

riablen ist ein »Typ« zugeordnet, der den Wertebereich und die möglichen Operationen bestimmt. Die Variablen im Beispiel sind Fließkommazahlen und vom Typ `float`.

Beispiele für andere mögliche Typen sind etwa `char` (Zeichen) oder `int` (Ganze Zahlen):

```
int    tag, monat, jahr;
char   trennzeichen;

cin >> tag >> monat >> jahr;
cin >> trennzeichen;

cout << "Datum:" << tag << trennzeichen
      << monat << trennzeichen << jahr;

cout << "\n"; // Zeilenvorschub ausgeben
```

Bei der Eingabe von 17, 4, 1991 und / entsteht folgende Ausgabe:

```
17/4/1991
```

Anmerkung: `cin` und `cout` sind für alle vordefinierten Datentypen (`int`, `char`, `float` ...) definiert.

- ❑ Die `if`-Abfrage `if (a!=0) { ... }` bewirkt, dass die zwischen den geschwungenen Klammern angeführten Anweisungen dann ausgeführt werden, wenn die zugehörige Bedingung (`a!=0`) wahr ist, `a` also ungleich 0 ist.
- ❑ Ist nach der `if`-Anweisung ein `else` angeführt, so bedeutet dies, dass die dahinter angegebene Anweisung ausgeführt wird, wenn die Bedingung der `if`-Anweisung falsch war.

Die Anweisung `if (a!=0) { Anweisungsfolge1 } else if (b!=0) { Anweisungsfolge2 } ...` bewirkt, dass *Anweisungsfolge1* ausgeführt wird, wenn `a` ungleich 0 ist. Andernfalls (`else`) wird geprüft, ob `b` ungleich 0 ist. Wenn ja, wird *Anweisungsfolge2* ausgeführt, ansonsten ...

2.2.3 Funktionen

Programm 2.4 auf der nächsten Seite berechnet einen »Turm« und gibt ihn aus. Dabei wird eine Zahl eingelesen und anschließend mit zwei multipliziert. Das Ergebnis wird ausgegeben und mit drei multipliziert. Dies wird bis zur Zahl 9 wiederholt, anschließend erfolgen Divisionen durch 2 bis 9. Das Ergebnis ist wiederum die Ausgangszahl. Der Turm ist »formatiert« auszuge-

ben, die Eingabe der Zahl 0 soll das Programm beenden. Negative Eingaben sind nicht zulässig.

In diesem Programm werden bereits die wichtigsten Kontrollstrukturen verwendet: Bedingungsanweisungen (`if`) und Schleifen (`for`, `while`, `do`).

Die formatierte Ein-/Ausgabe erfolgt über die Module `iostream` und die Funktion `setw` des Moduls `iomanip`.

Programm 2.4

Programm *Turm*

```
#include <iostream>
using namespace std;

// iomanip-Modul erlaubt das Festsetzen der
// Feldlänge bei der Ausgabe
#include <iomanip>
using namespace std;

// Funktion berechneTurm, berechnet u. gibt einen
// "Turm" basierend auf der Zahl num aus
void berechneTurm(int num) {
    // Vereinbarung der lokalen Variablen
    long help = num;

    // Multiplikation der Zahlen, for-Schleife:
    // die Variable opnd wird mit 2 initialisiert.
    // Dann wird der Schleifenrumpf durchlaufen (alle
    // Anweisungen zwischen { und }) und opnd erhöht.
    // Anschliessend wird geprüft, ob opnd < 10 ist.
    // Falls ja, dann wird der Schleifenrumpf abermals
    // durchlaufen ...
    for (int opnd=2; opnd<10; opnd++) {

        cout << setw(10); // Feldlänge f. Ausgabe
        cout << help;     // Ausgabe von help
        cout << setw(0); // Löschen d. Ausgabefeldes

        // Kaskadierung von Ausgabeanweisungen ...
        // '\n' entspricht einem Zeilenvorschub
        cout << " * " << opnd << '\n';
        help *= opnd; // entspricht "help = help*opnd"
    }

    // Division der Zahlen, for-Schleife
    for (int opnd=2; opnd<10; opnd++) {
        // Feldlänge direkt im cout-Statement
        cout << setw(9) << help << setw(0) <<
            " / " << opnd << '\n';
        help /= opnd; // entspricht "help = help/opnd"
    }
}
```

```

    // endl anstelle von '\n' ...
    cout << help << endl << endl;
}

void main() {    // Hauptfunktion
    int    num;

    // C++-Repeat-Schleife, fuehrt Schleifenrumpf
    // (Anweisungen zwischen "{" und "}" while (num>0);"
    // aus, solange num > 0 ist. Die Pruefung der
    // Bedingung erfolgt *nach* dem Schleifenrumpf.
    do { // do
        cout << "Bitte geben Sie eine Zahl >= 0 ein!\n";
        cout << "0 = Ende!!" << "\n\n";
        cin >> num;

        // Solange num < 0: fuehre Schleifenrumpf
        // aus (Anweisungen zwischen { und }) solange aus,
        // als num<0 (die Pruefung der Bedingung erfolgt
        // vor dem Ausfuehren des Schleifenrumpfs)
        while (num<0) {
            cout << " Bitte geben Sie eine Zahl >= 0 ein!\n";
            cin >> num; cout << "\n";
        }
        // num > 0: Fuehre Rumpf aus
        if (num > 0) {
            berechneTurm (num);
        }
        cout << "\n\n";
    } while (num > 0); // do
}

```

berechneTurm ist eine »Funktion«. Eine Funktion ist vereinfacht gesagt eine Zusammenfassung von Anweisungen zu einer Einheit. Die zusammengefassten Anweisungen können dann wie eine einzelne Anweisung behandelt werden. Im Detail weist berechneTurm folgende Besonderheiten auf:

- ❑ Die for-Schleife ist eine »Zählschleife«, die die angegebene Variable opnd von 2 bis 9 zählt und für jeden Wert die enthaltenen Anweisungen ausführt.
- ❑ Die Anweisung `cout << setw(10)` setzt die Länge des Ausgabefeldes mit 10 fest. Das bedeutet, dass alle Ausgaben rechtsbündig in einem Feld von zehn Zeichen erfolgen (sofern die Feldlänge ausreicht). Beispiele für derartige Ausgaben:

```
0123456789
    Hallo!
        Wau
Test123456
Donaudampfschiffahrtsgesellschaft
```

Das Hauptprogramm `main` besteht aus einer Schleife (`do { ... }` `while (num > 0)`); die wiederum eine Schleife (`while (num<0) { ... }`) sowie eine Bedingungsanweisung (`if (num>0) { ... }`) enthält:

- ❑ Die erste Schleife (die »äußere« Schleife) enthält eine zweite Schleife (die »innere«) sowie eine `if`-Anweisung. Die äußere Schleife ist eine so genannte `do`-Schleife, die alle Anweisungen zwischen dem folgenden `{` und dem abschließenden `}` wiederholt, solange die dahinter angeführte Bedingung (`num > 0`) gilt. Die Prüfung der Bedingung erfolgt jeweils *nach* der Ausführung dieser Anweisungen.
- ❑ Die innere `while`-Schleife wiederholt die enthaltenen Anweisungen, solange die gelesene Zahl kleiner 0 ist.
- ❑ Die Bedingungsanweisung `if (num > 0) { ... }` führt die enthaltenen Anweisungen dann aus, wenn `num` größer 0 ist.