



## Leseprobe

Norbert Heiderich, Wolfgang Meyer

Technische Probleme lösen mit C/C++

Von der Analyse bis zur Dokumentation

ISBN (Buch): 978-3-446-44784-4

ISBN (E-Book): 978-3-446-44905-3

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-44784-4>

sowie im Buchhandel.

# Vorwort des Herausgebers

## Was können Sie mit diesem Buch lernen?

Wenn Sie mit diesem Lernbuch arbeiten, dann erwerben Sie umfassende Erkenntnisse, die Sie zur Problemlösungsfähigkeit beim Programmieren mit der Hochsprache C/C++ führen.

Der Umfang dessen, was wir Ihnen anbieten, orientiert sich an

- den Studienplänen der Fachhochschulen für technische Studiengänge,
- den Lehrplänen der Fachschulen für Technik,
- den Anforderungen der Programmierpraxis,
- dem Stand der einschlägigen, professionellen Softwareentwicklung.

Sie werden systematisch, schrittweise und an ausgewählten Beispielen mit der Entwicklungsumgebung Visual C++ (VC++) von Microsoft vertraut gemacht.

Dabei gehen Sie folgenden Strukturelementen und Verfahrensweisen nach:

- Wie stellt sich die Entwicklungsumgebung dar?
- Welche grundlegenden Sprach- und Steuerungswerkzeuge gilt es kennenzulernen und an einfachen Beispielen anzuwenden?
- Wie wird ein Problem strukturiert programmiert?
- Wie muss die Software dokumentiert und getestet werden?
- Was meint objektorientierte Programmierung?

## Wer kann mit diesem Buch lernen?

Jeder, der

- sich weiterbilden möchte,
- die Grundlagen der elektronischen Datenverarbeitung beherrscht,
- Kenntnisse in den Grundlagen der elementaren Mathematik besitzt,
- bereit ist, sich mit technischen, mathematischen und kommerziellen Fragestellungen auseinanderzusetzen.

Das können sein:

- Studenten an Fachhochschulen und Berufsakademien,
- Studenten an Fachschulen für Technik,
- Schüler an beruflichen Gymnasien und Berufsoberschulen,
- Schüler in der Assistentenausbildung,
- Meister, Facharbeiter und Gesellen während und nach der Ausbildung,
- Umschüler und Rehabilitanden,
- Teilnehmer an Fort- und Weiterbildungskursen,
- Autodidakten.

## Wie können Sie mit diesem Buch lernen?

Ganz gleich, ob Sie mit diesem Buch in Hochschule, Schule, Betrieb, Lehrgang oder zu Hause lernen, es wird Ihnen Freude machen!

*Warum?*

Ganz einfach, **weil wir Ihnen ein Buch empfehlen, das in seiner Gestaltung die Grundgesetze des menschlichen Lernens beachtet.**

- Ein Lernbuch also! -

Sie setzen sich kapitelweise mit den Lehr-, Lerninhalten auseinander. Diese sind in überschaubaren Lernsequenzen schrittweise dargestellt. Die zunächst verbal formulierten Lehr-, Lerninhalte werden danach in die softwarespezifische Darstellung umgesetzt. An ausgewählten Beispielen konkretisiert und veranschaulichen die Autoren diese Lehr- bzw. Lerninhalte.

- Also auch ein unterrichtsbegleitendes Lehr-/Lernbuch mit Beispielen! -

Für das Suchen bestimmter Inhalte steht Ihnen das Inhaltsverzeichnis am Anfang des Buches zur Verfügung. Sachwörter finden Sie am Ende des Buches. Bücher zur vertiefenden und erweiterten Anwendung sind im Literaturverzeichnis zusammengestellt.

- Selbstverständlich mit Sachwortregister, Inhalts- und Literaturverzeichnis! -

Sicherlich werden Sie durch intensives Arbeiten mit diesem Buch Ihre „Bemerkungen zur Sache“ unterbringen und es so zu Ihrem individuellen Arbeitsmittel ausweiten:

- So wird am Ende Ihr Buch entstanden sein! -

Möglich wurde dieses Buch für Sie durch die Bereitschaft der Autoren und die intensive Unterstützung des Verlages mit seinen Mitarbeitern. Ihnen sollten wir herzlich danken.

Beim Lernen wünsche ich Ihnen viel Freude und Erfolg!

Ihr Herausgeber

*Manfred Mettke*

# Inhalt

<b>Einleitung</b> .....	<b>13</b>
<b>1 Systematik der Problemlösung</b> .....	<b>17</b>
1.1 Phasen der Programmentwicklung .....	17
1.2 Software-Lebenszyklus .....	19
1.3 Software-Entwicklungsverfahren .....	21
<b>2 Erste Gehversuche mit C/C++</b> .....	<b>26</b>
2.1 Warum gerade C/C++? .....	26
2.2 Compiler und Interpreter .....	28
2.3 Übersetzen eines C/C++-Programms .....	30
2.4 Programmstart .....	31
<b>3 Die Entwicklungsumgebung Visual C++</b> .....	<b>32</b>
3.1 Installation von VC++ .....	32
3.2 Starten von VC++ .....	34
3.3 Erstellen eines neuen Projektes .....	36
3.3.1 Win32-Projekte .....	37
3.3.1.1 Variante 1 - VC++ leistet Vorarbeit .....	38
3.3.1.2 Variante 2 - leeres Projekt .....	39
3.3.2 CLR-Projekte .....	42
3.4 Übersetzen eines eigenen Programms .....	44
3.5 Ausführen eines eigenen Programms .....	47
3.6 Paradigmen der Projektorganisation .....	47
<b>4 Grundlegende Sprach- und Steuerungselemente</b> .....	<b>50</b>
4.1 Kommentare .....	50
4.2 Datentypen und Variablen .....	51
4.2.1 Variablenamen .....	52
4.2.2 Ganzzahlige Variablen .....	52
4.2.3 Fließkommazahlen .....	54
4.2.4 Zeichen .....	55
4.2.5 Felder .....	56

4.2.5.1	Eindimensionale Felder .....	56
4.2.5.2	Mehrdimensionale Felder .....	57
4.2.5.3	Zugriff auf die Elemente eines Feldes .....	59
4.2.5.4	Startwertzuweisung für ein- und mehrdimensionale Arrays .....	61
4.2.6	Zeichenketten .....	63
4.3	Konstanten .....	64
4.4	Operatoren .....	65
4.4.1	Vorzeichenoperatoren .....	65
4.4.2	Arithmetische Operatoren .....	65
4.4.2.1	Addition + .....	65
4.4.2.2	Subtraktion - .....	65
4.4.2.3	Multiplikation * .....	66
4.4.2.4	Division / .....	66
4.4.2.5	Modulo % .....	66
4.4.2.6	Zuweisung = .....	66
4.4.2.7	Kombinierte Zuweisungen .....	67
4.4.2.8	Inkrementierung ++ .....	67
4.4.2.9	Dekrementierung - .....	68
4.4.3	Vergleichsoperatoren .....	68
4.4.3.1	Gleichheit == .....	68
4.4.3.2	Ungleichheit != .....	68
4.4.3.3	Kleiner < .....	69
4.4.3.4	Größer > .....	69
4.4.3.5	Kleiner gleich <= .....	69
4.4.3.6	Größer gleich >= .....	70
4.4.4	Logische Operatoren .....	70
4.4.4.1	Logisches NICHT ! .....	70
4.4.4.2	Logisches UND && .....	70
4.4.4.3	Logisches ODER    .....	70
4.4.5	Typumwandlungsoperator .....	71
4.4.6	Speicherberechnungsoperator .....	71
4.4.7	Bedingungsoperator .....	72
4.4.8	Indizierungsoperator .....	73
4.4.9	Klammerungsoperator .....	73
4.5	Anweisungen und Blöcke .....	75
4.6	Alternationen .....	75
4.6.1	Einfache Abfragen (if - else) .....	75
4.6.2	Mehrfachabfragen (else - if) .....	76
4.6.3	Die switch-case-Anweisung .....	77
4.7	Iterationen .....	79
4.7.1	Zählergesteuerte Schleifen (for) .....	79
4.7.2	Kopfgesteuerte Schleifen (while) .....	83
4.7.3	Fußgesteuerte Schleifen (do - while) .....	84
4.7.4	Schleifenabbruch (continue) .....	85
4.7.5	Schleifenabbruch (break) .....	86
4.7.6	Schleifenumwandlungen .....	88

4.8	Funktionen	88
4.8.1	Formaler Aufbau einer Funktion	89
4.8.1.1	Der Funktionskopf	90
4.8.1.2	Der Funktionsrumpf	91
4.8.2	Datentyp und Deklaration einer Funktion - Prototyping	92
4.8.3	Das Prinzip der Parameterübergabe	97
4.8.3.1	Aufrufverfahren call by value	97
4.8.3.2	Aufrufverfahren call by reference	99
4.8.3.3	Adressoperator, Zeiger und Dereferenzierung	102
4.8.4	Regeln für ein erfolgreiches Prototyping	103
4.8.5	Die exit()-Funktion	104
4.8.6	Rekursive Funktionen	104
4.9	Ein- und Ausgabe	107
4.9.1	Formatierte Eingabe mit scanf()	107
4.9.2	Formatierte Ausgabe mit printf()	108
4.9.3	Arbeiten mit Dateien	109
4.9.3.1	Öffnen der Datei	110
4.9.3.2	Verarbeiten der Datensätze	110
4.9.3.3	Schließen der Datei	111
4.9.3.4	stdio.h	111
4.9.3.5	fflush() und stdin	113
<b>5</b>	<b>Strukturierte Programmierung</b>	<b>114</b>
5.1	Problemstellung	115
5.2	Problemanalyse	116
5.3	Struktogramm nach Nassi-Shneiderman	119
5.3.1	Sequenz	121
5.3.2	Alternation	123
5.3.3	Verschachtelung	124
5.3.4	Verzweigung	125
5.3.5	Schleifen	127
5.3.5.1	Zählergesteuerte Schleife	127
5.3.5.2	Kopfgesteuerte Schleife	131
5.3.5.3	Fußgesteuerte Schleifen	133
5.3.5.4	Endlosschleifen	134
5.3.5.5	Kriterien zur Schleifenauswahl	134
5.3.6	Programm- oder Funktionsaufruf	134
5.3.7	Aussprung	135
5.3.8	Rechnergestützte Erstellung von Struktogrammen	136
5.3.8.1	StruktEd	136
5.3.8.2	hus-Struktogrammer	143
5.4	Flussdiagramm nach DIN 66001	151
5.5	Programmerstellung	153
5.6	Programmtest	153
5.7	Programmlauf	154
5.8	Dokumentation nach DIN 66230	155

5.8.1	Funktion und Aufbau des Programms	155
5.8.2	Programmkenndaten	156
5.8.3	Betrieb des Programms	157
5.8.4	Ergänzungen	157
5.9	Aspekte des Qualitätsmanagements EN-ISO 9000	158
5.10	Algorithmus – was ist das?	159
5.11	EVA-Prinzip	165
5.12	Programmierung von Formelwerken	166
<b>6</b>	<b>Lösung einfacher Probleme</b>	<b>171</b>
6.1	Umrechnung von Temperatursystemen	171
6.2	Flächenberechnung geradlinig begrenzter Flächen (Polygone)	177
6.2.1	Erste Problemvariation: Berechnung der Schwerpunktkoordinaten $S(x_S; y_S)$ von polygonförmig begrenzten Flächen	184
6.2.2	Zweite Problemvariation: Suche nach einem „günstigen“ Treffpunkt	185
6.3	Berechnung einer Brückenkonstruktion	186
6.4	Schaltjahrüberprüfung	190
6.5	Ein Problem aus der Energiewirtschaft	196
6.6	Logarithmische Achsenteilung	206
<b>7</b>	<b>Objektorientierte Programmierung (OOP)</b>	<b>214</b>
7.1	Modellbildung mittels Abstraktion	214
7.2	Klassen und Objekte	215
7.3	Attribute und Methoden einer Klasse	218
7.4	Bruchrechnung mit OOP	219
7.5	Vererbung	228
7.6	Strings	235
7.7	Typumwandlungen	236
7.8	Strukturierte Programmierung vs. OOP	240
<b>8</b>	<b>Lösung fortgeschrittener Probleme</b>	<b>241</b>
8.1	Grafische Darstellung funktionaler Abhängigkeiten	241
8.1.1	Welt- und Screenkoordinaten	243
8.1.2	Koordinatentransformationen	245
8.1.3	Darstellung der Sinusfunktion	251
8.1.4	Darstellung quadratischer Parabeln	255
8.1.5	Spannungsteilerkennlinien	258
8.2	Lösung technisch-wissenschaftlicher Probleme	260
8.2.1	Widerstandsreihen E6 bis E96	260
8.2.2	Farbcodierung von Widerständen nach DIN 41429	263
8.2.3	Fourier-Synthese periodischer empirischer Funktionen	266
8.2.4	Fourier-Analyse empirischer Funktionen	274
8.3	Nullstellenbestimmung von Funktionen	279
8.3.1	Inkrementverfahren und Intervallhalbierung	279
8.3.2	Die regula falsi	284
8.3.3	Das Newton-Verfahren	286

8.4	Numerische Integration .....	289
8.4.1	Riemannsche Unter- und Obersummen .....	289
8.4.2	Trapezregel .....	293
8.4.3	Simpsonsche Regel .....	298
8.4.4	Effektivwertberechnungen .....	303
8.5	Einbindung eigener Klassen .....	305
8.5.1	Das „Platinenproblem“ als objektorientierte Konsolenanwendung ..	305
8.5.2	Das „Platinenproblem“ in der Erweiterung mit grafischer Benutzeroberfläche .....	310
<b>9</b>	<b>Lösung komplexer Probleme .....</b>	<b>314</b>
9.1	Kurvendiskussion und Funktionsplotter am Beispiel ganzzahliger Funktionen bis 3. Ordnung .....	314
9.2	Ausgleichsrechnung – Bestimmung der „besten“ Geraden in einer Messreihe .....	317
9.3	Digitaltechnik .....	327
<b>10</b>	<b>Tabellen und Übersichten .....</b>	<b>341</b>
10.1	Datentypen und ihre Wertebereiche .....	341
10.2	Vergleich der Symbole nach DIN 66 001 und der Nassi-Shneiderman-Darstellung .....	342
10.3	Schlüsselwörter ANSI C .....	343
10.4	Erweiterte Schlüsselwörter C++ .....	345
10.5	ASCII-Tabelle .....	348
10.6	Standardfunktionen und ihre Zuordnung zu den Header-Dateien (Include)	350
<b>Literatur</b>	.....	<b>354</b>
<b>Index</b>	.....	<b>355</b>



# 2

## Erste Gehversuche mit C/C++

Dieses Kapitel beschäftigt sich mit einigen grundlegenden Aspekten der Programmiersprache C/C++. Neben der Frage, warum es sinnvoll ist, gerade mit C/C++ zu arbeiten, werden Funktionsweisen der Komponenten der Entwicklungsumgebung betrachtet und erläutert. In den folgenden Kapiteln werden zunächst Beispiele in klassischem C als Konsolenanwendungen realisiert, bevor später objektorientiert mit C++ weitergearbeitet wird. Dann sind die Beispiele auch mit grafischen Oberflächen ausgestattet.

### ■ 2.1 Warum gerade C/C++?

Wer C/C++ erlernen will, hat sich für eine Programmiersprache entschieden, die auf fast allen Rechnertypen und unter fast allen Betriebssystemen verfügbar ist. Es steht Ihnen, anders als bei vielen anderen Programmiersprachen, auf den verschiedensten Entwicklungsplattformen eine genormte Standard-Bibliothek zur Verfügung. Damit gelingt eine einheitliche Implementierung der mit dieser Programmiersprache erstellten Programme mit sehr hoher Geschwindigkeit.

C wird auch als Highlevel-Assembler bezeichnet, also als Programmiersprache, die sehr nah an der Maschinensprache ist. Dies beruht auf der Tatsache, dass der Kern (bzw. Kernel) aller gängigen Betriebssysteme in C geschrieben wurde. Damit eignet sich C/C++ auch in besonderem Maße für die Systemprogrammierung, also für Programme, die für den Betrieb von Rechenanlagen erforderlich sind.

Dank der relativ einfachen Struktur und dem geringen Umfang der eigentlichen Sprache, d. h. der verfügbaren Schlüsselwörter der Programmiersprache, war es möglich, **C-Compiler**, also spezielle Programme zur Übersetzung des vom Programmierer erstellten Codes in eine maschinenverständliche Sprache, für alle Arten von Prozessorplattformen zu entwickeln, so dass die Programmiersprache C/C++ heute für die gesamte Leistungspalette vom Mikrocontroller bis zu High-End-Rechnern verfügbar ist. Für den Entwickler von Software bedeutet dies: Egal für welche Prozessorplattform programmiert wird, einen C-Compiler wird man für das relevante Zielsystem bekommen. Man braucht sich nicht um eine Programmierung zu kümmern, die spezifisch für den jeweiligen Zielprozessor ist. In den meis-

ten Fällen wird es möglich sein, die auf einer Plattform entwickelte Anwendung auf einer anderen Plattform auszuführen. Der erforderliche Anpassungsaufwand ist in aller Regel sehr überschaubar.



Das bedeutet nicht, dass man fertige Programme von einer Plattform auf eine andere übertragen kann (etwa von einem Windows-PC auf einen Linux-PC) und diese dann auf der neuen Plattform (also unter Linux) sofort wieder funktionieren. Vielmehr ist nur die problemlose Übertragung der Quelltexte auf ein neues System gemeint, auf dem diese dann mit dem entsprechenden Compiler und Linker (ein Linker oder Binder ist ein Programm, das einzelne Programmmodule zu einem ausführbaren Programm verbindet) in ein funktionierendes Programm umzuwandeln sind!

Die Tatsache, dass Programme, die in C/C++ geschrieben werden, sehr klein sind (nur in Assembler – also Maschinensprache – geschriebene Programme sind noch kleiner), macht C/C++ zu einer wichtigen Programmiersprache im Bereich Embedded Systems (also Systemen, die stark einschränkenden Randbedingungen unterliegen, wie geringe Kosten, Platz-, Energie- und Speicherverbrauch) und der Mikrocontroller-Programmierung, wo Speicherplatz ebenfalls sehr kostbar ist.

Ein C/C++-Programm wird mithilfe eines **Compilers** (dem Übersetzer des Quelltextes) aus einer oder mehreren einfachen Textdateien zu Objektcodedateien übersetzt. Diese Objektcodedateien werden anschließend von einem **Linker** (bzw. Linkage-Editor = Binder, Werkzeug für das Zusammenfügen übersetzter Programmteile) mit den erforderlichen Systembibliotheken zu einer ausführbaren Datei (der Executable – oder kurz EXE-Datei) zusammengebunden.



Jedes ausführbare C/C++-Programm besitzt eine Hauptfunktion. In C wird diese Hauptfunktion als `main` bezeichnet.

Damit das Betriebssystem erkennen kann, wo der Einstiegspunkt für den Ablauf eines C/C++-Programms zu finden ist, muss diese Namenskonvention unbedingt eingehalten werden. Auch wenn andere Entwicklungsumgebungen als das Visual Studio von Microsoft oder andere Compiler eingesetzt werden, ändert sich an diesem Einstiegspunkt nichts. Variieren kann allenfalls die Parametrisierung (also die Art, Anzahl oder der Datentyp der Übergabeparameter) dieser Hauptfunktion. Dieser Aspekt wird später in Kapitel 4.8, in dem es um Funktionen gehen wird, noch ausführlich erläutert.

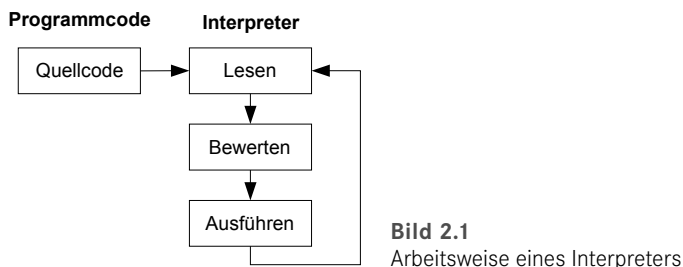
Darüber hinaus ist es natürlich auch möglich, eigene Programme und/oder Funktionen in eigenen Bibliotheken zusammenzufassen, um diese später erneut benutzen zu können. Diese Bibliotheken können bei einem späteren Bindevorgang durch den Linker wieder verwendet werden, damit diese dann zu einem neuen Programm hinzugebunden werden.

## ■ 2.2 Compiler und Interpreter

Die höheren Programmiersprachen (dazu zählen u. a. FORTRAN, ALGOL und C/C++) sind entwickelt worden, damit die Beschreibung eines Lösungsverfahrens, der Algorithmus, in einer für Menschen einfacher lesbaren Form erfolgen kann und nicht in Maschinencode vorliegen muss. Der Programmierer als der Anwender solcher Programmiersprachen soll sich also auf die Lösung seiner konkreten Aufgabenstellung konzentrieren können, ohne sich zu sehr mit den Interna des Rechners beschäftigen zu müssen. Ein weiterer, nicht zu vernachlässigender Vorteil ist, dass Programmiersprachen normalerweise unabhängig von der Maschine sind, auf der die Programme ausgeführt werden.

Die Tatsache, dass es zwei verschiedene Verfahren zur Erzeugung von Programmen gibt, deutet das Problem an, das sich hinter der Verwendung einer Programmiersprache verbirgt: die Programme können nicht mehr direkt und unmittelbar vom Computer gelesen und zur Ausführung gebracht werden, sondern sie müssen erst in eine vom Computer interpretierbare passende Darstellungsform gebracht werden.

Eine Möglichkeit Quellprogramme (also die vom Programmierer erstellten und für den Programmierer lesbaren Textdateien mit den Programmen) zu übersetzen, sind **Interpreter**. Bei ihnen wird das Programm Zeile für Zeile gelesen und bewertet, wobei Schreibfehler oder Verstöße gegen die Regeln der Programmiersprache, Syntaxfehler, festgestellt werden. Danach führt der Interpreter die mit den Anweisungen verbundenen Befehle und Aktionen aus. Die Arbeitsweise solcher Interpreter kann wie in Bild 2.1 dargestellt werden.



**Bild 2.1**  
Arbeitsweise eines Interpreters

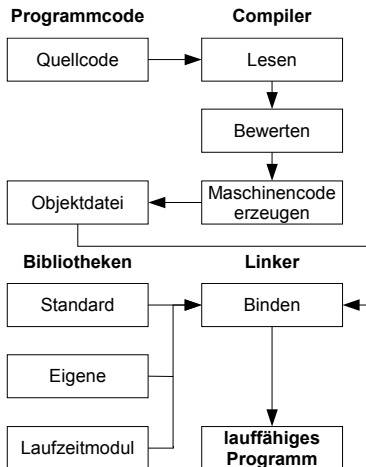
Der große Vorteil interpretierter Programme besteht darin, dass es möglich ist, schnell einmal etwas auszuprobieren oder während des Programmablaufs einzugreifen und Werte von Variablen zu betrachten oder zu verändern. Für professionelle Projekte dagegen sind Interpreter eher ungeeignet. Zuerst einmal ist da das Problem der mangelnden Geschwindigkeit zu nennen. Denn egal, ob ein Programm zum ersten oder tausendsten Mal ausgeführt wird, alle zugehörigen Programmzeilen durchlaufen immer wieder den Zyklus: *Lesen - Bewerten - Ausführen*.

Außerdem ist nur ein geringer Schutz des Quellcodes vor Eingriffen von außen gegeben. Der Anwender hat jederzeit die Möglichkeit den Code zu manipulieren, ohne sich mit dem Programmierer abzusprechen. Es entstehen so in der Praxis ziemlich schnell alternative Programmversionen, die bei einem neuen Release zu großen Problemen führen.

Der andere Lösungsansatz, um ein ausführbares Programm zu erzeugen, führt über den Einsatz eines Compilers. Ein **Compiler** ist ein Programm, das einen Quellcode einliest und

ihn zunächst auf syntaktische Fehler untersucht. Werden keine wirklichen Fehler festgestellt, so wird der Quellcode in eine maschinenlesbare Form übersetzt.

Nach dem abschließenden Schritt des Bindens (Linken) liegt das Programm in einer Form vor, die auf dem Rechner, auf dem es verarbeitet wurde, lauffähig (also ausführbar) ist (engl. executable, daher die Extension (Programmendung) EXE). Dieses Programm kann nun unabhängig von Quellcode und Compiler ausgeführt werden. Diese Arbeitsweise lässt sich wie in Bild 2.2 darstellen.



**Bild 2.2**  
Arbeitsweise eines Compilers und Linkers

Der große Vorteil dieser Vorgehensweise liegt darin, dass die Programme ohne Preisgabe des Quellcodes lauffähig sind. So können z. B. auch im Quellcode verwendete Betriebsgeheimnisse von keinem Anwender mehr eingesehen werden. Vor allen Dingen aber laufen solche Programme sehr viel schneller ab als interpretierte Programme, da ja die Interpretation jedes Programmstatements (also jeder Programmzeile) entfällt. Die Überprüfung hat bereits bei der Kompilierung (also der Übersetzung in die Object-Datei) stattgefunden. Ein gewisser Nachteil besteht zweifellos darin, dass nun bei jeder Programmänderung der komplette Zyklus *Editieren - Übersetzen - Starten* durchlaufen werden muss, bevor man etwas in einem Programm ausprobieren kann. Das kostet in der Testphase natürlich etwas mehr Zeit als bei interpretativen Programmen. Denn auch beim Auftreten eines syntaktischen Fehlers muss man immer erst wieder das Programm in den Editor laden, den Fehler beheben und die Übersetzung (Kompilierung) von neuem starten.

Deutlich entschärft wird dieses Problem dadurch, dass die meisten Compiler heutzutage mit einer sogenannten integrierten Entwicklungsumgebung ausgestattet sind, in der der Editor und der Compiler eine Einheit bilden. Damit verkürzen sich die Bearbeitungsschritte erheblich.

## ■ 2.3 Übersetzen eines C/C++-Programms

Nachdem der Quellcode eingegeben ist, muss dieser in eine für den Computer verständliche Sprache übersetzt werden. An dieser Stelle wird ein **Object** erzeugt. Dieser Prozess stellt einen Zwischenschritt bei der Herstellung eines ausführbaren Programms dar. Das Werkzeug, mit dem hier gearbeitet wird, ist ein Compiler. Das ist ein eigenständiges Programm zur Übersetzung des Quellcodes in ein Object.

1. Der erste Schritt ist der **Präprozessorlauf**. Bei dem Programm, das hier zum Einsatz kommt, handelt es sich um einen Textersetzer, der spezielle Dateien, die Headerdateien, in den Programmtext importiert. Zusätzlich vermag es der Präprozessor, Makros und symbolische Konstanten in die ihnen zugeordneten Anweisungsfolgen und Werte zu übersetzen und bestimmte Programmteile (Kommentare) vor der Übersetzung auszublenken. Vom Präprozessor wird eine temporäre Zwischendatei erzeugt, die aber nach dem Übersetzungsvorgang wieder zerstört wird und somit keinen dauerhaften zusätzlichen Speicherplatz auf der Festplatte beansprucht.
2. Im zweiten Schritt testet der Compiler, ob die Schreibweisen und der Aufbau des Programms den Regeln der Programmiersprache entsprechen. Hier spricht man von der **syntaktischen Analyse**. Ein dabei auftretender Fehler wird dementsprechend Syntaxfehler genannt. Moderne Compiler sammeln nun die Fehler und brechen die Übersetzung erst bei der Überschreitung einer gewissen Fehlerobergrenze ab. Anschließend kann der Anwender von der Fehlerliste aus zu den Programmstellen springen, die der Compiler moniert hat. Hier gilt es zu bedenken, dass es sich nicht immer nur um eine einzige Stelle handeln muss, an der der Fehler liegen kann. Evtl. ist schon einige Zeilen vorher ein Fehler programmiert worden, der erst jetzt zu einem Syntaxproblem führt und den Compiler zur Meldung veranlasst. Der Programmierer korrigiert die beanstandeten Zeilen und startet den Übersetzungslauf erneut. Je nach Komfort oder Aussagekraft der Fehlermeldungen ist die Analyse der Fehlerquellen aufwendig. In jedem Fall bedarf es einiger Übung, um aus den teilweise sehr kryptischen Fehlermeldungen die Fehlerquelle zu erkennen.
3. Nach einigen Durchgängen ist das Programm in der Regel frei von Syntaxfehlern. Der Compiler hat dann den **Objectcode** erzeugt. Dieser ist prinzipiell schon maschinenlesbar, kann allerdings noch nicht ausgeführt werden, da einerseits weitere Objectdateien evtl. hinzuzufügen sind, andererseits aber immer Bibliotheksdateien und das Laufzeitmodul hinzugebunden werden müssen.
4. Das **ausführbare Programm**, das dann auf der Plattform, auf der die Entwicklung durchgeführt wurde, lauffähig ist – und in aller Regel auch nur auf dieser Plattform, denn C/C++ ist eine plattformabhängige Programmiersprache –, erzeugt letztendlich der Linker. Hier werden nun alle Bestandteile, die zur Lauffähigkeit des Programms benötigt werden, also das Laufzeitmodul, die Bibliotheksfunktionen und Objectdateien, zu einer ausführbaren Programmdatei zusammengeführt (gebunden). Auch in diesem Schritt besteht noch die Möglichkeit, dass der Programmierer Fehler begeht. So sind Fehler, die in diesem Schritt entdeckt werden z. B. fehlende Funktionsdefinitionen, fehlende externe Variablen oder doppelte Funktionsdefinitionen. Auch in solchen Fällen muss der Programmierer wieder zurück bis in den Quellcode, um die Fehler zu korrigieren. Anschließend sind die aufgeführten Schritte erneut alle zu durchlaufen.

## ■ 2.4 Programmstart

Nachdem der Quellcode fehlerfrei übersetzt und gebunden wurde, liegt eine Datei vor, die ein ausführbares Programm-Modul darstellt. Sie ist gekennzeichnet durch die **Extension** (Dateierweiterung) EXE. Dieses Programm kann nun, wie jedes andere Programm auch, gestartet werden. An dieser Stelle beginnt standardmäßig die Testphase, in der der Programmierer versucht herauszubekommen, ob die von ihm realisierte Lösung immer (d.h. mit unterschiedlichen Eingabedaten) zu einem vernünftigen und erwarteten Ergebnis führt. Dazu gehören selbstverständlich auch die Auswahl von geeigneten Testdaten sowie die Dokumentation von erwarteten und erzielten Resultaten.



Selten werden die Programme auf Anhieb die an sie gestellten Aufgaben mit den erwarteten Ergebnissen liefern. Es ist also nun wieder der Quellcode heranzuziehen, um ihn so zu berichtigen, dass die Ergebnisse den Erwartungen entsprechen.

Die häufigsten Fehler, die in diesem Zusammenhang gemacht werden, sind:

▪ Designfehler:	Die Lösung ist nicht angemessen. Zur Realisierung des Programms sind zu viele Annahmen getroffen worden, die nicht in jedem (Beispiel-) Fall tatsächlich auch vorliegen, oder es ist eine falsche Schrittfolge der einzelnen durchzuführenden Arbeitsschritte innerhalb des Quellcodes umgesetzt worden.
▪ Logische Fehler:	Z. B. falsche Abfragebedingungen oder Reihenfolge der Programmschritte falsch angeordnet.
▪ Sprachbedingte Fehler:	Da C/C++ eine äußerst knapp gehaltene Sprache ist, können Laufzeitfehler daraus resultieren, dass man sich schlicht vertippt hat. Es kann also passieren, dass man einen syntaktisch korrekten Programmteil erstellt, der aber doch ein logisch falsches Konstrukt enthält, das der Compiler allerdings nicht identifizieren kann. Man spricht hier von semantischen Fehlern.

Durch die sequentielle Arbeitsweise (Korrektur von Fehlern – Test der neuen Version – Korrektur von Fehlern – Test der neuen Version ...) entstehen immer wieder neue Versionen des Programms, die man durch entsprechende Vergabe von Programmnamen oder Versionsnummern dokumentieren kann und sollte. Ist dann irgendwann der endgültige Programmstand erreicht, können diese Zwischenstände wieder entsorgt werden.

# Index

## A

Abfrage 75  
abgeleitete Klasse 228  
Ablauflinie 152  
abstrakte Klasse 217  
Abstraktion 214  
abweisende Schleife 132  
Achsenbeschriftung 259  
Addition 65  
Adressoperator 102 f.  
Aggregation 228  
Aggregatobjekt 228  
Algorithmus 159, 178, 218  
Alternation 75, 120, 123 f., 136, 141, 150  
Anpassungshinweise 157  
Anweisungen 75  
arithmetischer Operator 65  
Array 56, 63  
ASCII-Tabelle 348  
ASCII-Zeichensatz 55  
Assoziation 228  
Assoziativität 74  
Attribut 216, 218, 302  
Attributwert 217, 253  
Aufgabe 156  
Aufgabenlösung 155  
Aufgabenstellung 155  
Ausgabesymbol 152  
Ausgleichsrechnung 317  
Ausnahmefehler 253  
Außerbetriebnahme 19  
Aussprung 121, 135  
Auswahl 114, 138  
auto 343

## B

Basisklasse 228  
Batchdatei 104  
Bedienung 157  
Bedingungsoperator 72  
Bemerkung 152  
Betrieb 19, 21  
Bezeichnung 156  
Bibliotheksfunktionen 96  
Bildpunkt 244  
binäre Operatoren 65  
Blöcke 75  
bool 346  
Boole 327  
Boole, Georg 327  
break 78, 86, 127, 343  
Brückenkonstruktion 186

## C

call by reference 99  
call by value 97  
case 343  
case-sensitiv 52  
Cast 236  
catch 346  
char 52, 63, 343  
class 346  
Codierung 20, 25  
const 343  
const\_cast 346  
Container-Klasse 235  
continue 85, 343  
Copy-and-paste 89

## D

Datentyp 51  
 Datentypen und ihre Wertebereiche 341  
 Debugger 154  
 default 126, 343  
 Default-Werte 259  
 Definition der Funktion 92  
 Dekrementierung 68  
 delete 346  
 Dereferenzierung 102, 104  
 Designfehler 31  
 Deskriptoren 157  
 destruktives Schreiben 110  
 Destruktor 222  
 Detailkonzept 22, 24  
 deterministische Verfahren 160  
 Differenzialrechnung 314  
 Digitalschaltung 329  
 Digitaltechnik 327  
 DIN 66001 151  
 DIN 66230 155  
 Division 66  
 do 343  
 Dokumentation 115, 138, 155, 221  
 double 54, 343  
 do - while 84  
 Dualzahlen 330 f.  
 dynamic\_cast 346

## E

Effektivwertberechnungen 303  
 eindimensionale Felder 56  
 einfache Abfrage 75  
 Eingabesymbol 152  
 Elemente eines Feldes 59  
 Elementverweis-Operator 236  
 else 344  
 else - if 76  
 Endlosschleife 81, 127, 134  
 Energiewirtschaft 196  
 EN-ISO 9001 158  
 Entität 216  
 Entwurf 20 f., 24  
 enum 344  
 E-Reihen 260  
 erweiterte Schlüsselwörter C++ 345  
 euklidischer Algorithmus 164  
 EVA-Prinzip 165  
 Exception 253  
 Exemplar 216  
 explicit 346

explizite Typumwandlungen 236  
 extern 344, 346  
 externe Operation 219  
 Extremstellen 314

## F

Fakultätsberechnung 160  
 false 346  
 Farbcodierung nach DIN 41429 263  
 fclose() 111  
 Fehler 31  
 Fehlerbehandlung 157  
 Fehlerquadratsumme 318  
 Feld 56 ff.  
 fflush() 113  
 fgetc() 110  
 fgets() 110  
 FILE 109  
 Fließkommazahl 54  
 float 54, 60, 344  
 Flussdiagramm 151, 153  
 Font 257  
 fopen() 110  
 for 79, 344  
 formatierte Ausgabe 108  
 formatierte Eingabe 107  
 Formelwerk 166 f.  
 Fourier  
 - Analyse 274  
 - Koeffizient 274  
 - Reihen 267  
 - Synthese 266  
 fprintf() 111  
 fputc() 110  
 fputs() 110  
 fread() 111  
 friend 346  
 fscanff() 111  
 Funktion 88  
 Funktions  
 - aufruf 120, 134 f., 250  
 - graph 255  
 - kopf 89 f.  
 - rumpf 89, 91  
 fußgesteuerte Schleife 84, 127, 133 f.  
 fwrite() 111

## G

ganzzahlige Variablen 52  
 Geheimnisprinzip 217  
 Gerätebedarf 156



get-Methode 222  
Gleichheit 68  
goto 344  
Graphical User Interface (GUI) 235  
Grenzstelle 152  
Grobkonzept 22  
größer 69  
größer gleich 70  
GUI 36, 214, 235

## H

Hauptprojektdatei 94  
Header-Datei 94, 350  
Heißeleiter 206, 213

## I

if 344  
if - else 75  
Implementierung 153  
Implementierungsaufwand 89  
implizite Typumwandlung 236  
Include 350  
Indizierungsoperator 73  
Initialisierung 79  
Inkludierung 96  
Inkrementierung 67  
Inkrementverfahren 279  
inline 346  
Instanz 216, 235  
int 52f., 56f., 344  
IntelliSense 237f.  
Intervallhalbierung 280  
Iteration 79, 128, 150

## K

Kapselung 216, 221  
Kennlinie 258  
Kennlinienfeld 258  
Klammerungsoperator 73  
Klassen 215, 217, 314  
- beschreibung 217  
- diagramm 217  
- hierarchie 228  
- operation 219  
kleiner 69  
kleiner gleich 69  
kombinierte Zuweisung 67  
Kommentar 50  
Komponente 32, 228  
Komposition 228

Konstante 52, 57, 64  
Konstruktor 219, 222  
Kontrollstruktur 75  
Koordinatentransformation 245  
kopfgesteuerte Schleife 82f., 132, 134  
Kosinus-Koeffizient 274

## L

Leibniz 330  
logarithmische Achsenteilung 206f.  
logischer Fehler 153  
logisches NICHT 70  
logisches ODER 70  
logisches UND 70  
long 52f., 108, 344  
long int 52

## M

malloc() 181  
mehrdimensionale Felder 57  
Mehrfachabfrage 76  
Mehrfachauswahl 345  
Message-Box 253  
Messreihe 317  
Methode 216, 218, 257, 259  
Methode überladen 219  
mode 110  
Modellbildung 214  
Modul 89  
Modulo-Operator 66  
Multiplikation 66  
mutable 347

## N

nachprüfende Schleife 133  
namespace 347  
Nassi-Shneidermann 119, 153  
new 347  
Newton-Verfahren 286  
nicht-abweisende Schleife 133  
NTC 206  
Nullstellenbestimmung 279  
numerische Integration 289  
Nutzungsvereinbarungen 157

## O

Obersumme 291  
Objekt 215f., 253  
Objektoperation 219

objektorientierte Programmierung (OOP)  
 112, 114, 152, 214  
 OOP 112, 114, 152, 214  
 Operation 218  
 Operator 65, 347  
 Ordinalwert 55 f.  
 Overloading 219

## P

Parabel 255  
 Paradigma 114  
 Parameterübergabe 97  
 Planung 20  
 Pointer 102  
 Postfix 67  
 Potenzfunktion 125  
 pow( ) 125  
 Präfix 67  
 Präprozessoranweisung 96  
 printf() 107, 130  
 Priorität 74  
 private 218, 221, 229, 347  
 Problemanalyse 115 f., 166, 171  
 Problemstellung 115, 171  
 Programm 171  
 - ablauf 154, 156, 236  
 - ablaufplan 151  
 - aufbau 156  
 - aufruf 120  
 - bedarf 156  
 - erstellung 115, 153  
 - lauf 115, 142, 154, 171  
 - test 171, 181  
 Programmierparadigma 114  
 Programmiersprache 157  
 Projektmanagement 18 f.  
 Projektverwaltung 18  
 protected 218, 222, 347  
 Protokoll einer Klasse 218  
 Prototyp 93  
 Prozeduren 89, 114  
 public 218, 221, 229, 347

## Q

Qualitätsmanagement 158  
 Qualitätssicherung 18, 159

## R

Realisierung 21 f., 24  
 Rechteckfunktion 268, 276

Referenz 99, 101  
 register 344  
 regula falsi 284  
 Reinitialisierung 80  
 reinterpret\_cast 347  
 Rekursion 104  
 return 344  
 Riemannsche Unter- und Obersummen 289  
 Rundungsfehler 55

## S

scanf( ) 107, 132  
 Schaltjahrüberprüfung 190  
 Schaltnetz 328  
 Schleifen 114, 120, 127, 144, 149, 192  
 - abbruch 85 f.  
 - bedingung 82  
 - begrenzungssymbol 152  
 - kopf 82  
 - rumpf 82 ff.  
 - steuerungsvariable 81  
 - umwandlung 88  
 Schlüsselwörter ANSI C 343  
 Schnittstelle 156, 218  
 Schriftart 257  
 Schrittweitenwert 255  
 Schwerpunktkoordinaten 184  
 Screenkoordinaten 243  
 Sequenz 114, 120, 135, 139  
 set-Methode 222  
 Shannon, Claude E. 327  
 short 52 f., 108, 344  
 short int 52, 54, 328  
 signed 53 f., 344 f.  
 Signifikanz 54  
 Simpsonsche Regel 298  
 Sinusfunktion 241, 243, 251  
 Sinus-Koeffizient 274  
 sizeof 344  
 Skalierung 255  
 Softwareengineering 17  
 Softwarelebenszyklus 19  
 SolidBrush 257  
 Spannungsteiler 258  
 Spannungsteilerkennlinie 258  
 Spannungsteilerwiderstand 258  
 Speicherbedarf 156  
 Speicherberechnungsoperator 71  
 Spezifikation 20  
 sprachbedingte Fehler 31  
 sqrt( ) 125, 296  
 Standardeingabe 83

- Standardfunktionen 350
- Stapelverarbeitungsdatei 104
- Startbedingung 81
- Startwertzuweisung 61
- static 344
- static\_cast 347
- stdin 83
- stdio.h 109
- Stilllegung 21
- STL 235
- Stream 109
- string 194, 235
- String 235
- struct 344
- Struktogramm 119, 151, 171
- Struktur 109, 218, 320
- strukturierte Programmierung 14
- Subtraktion 65
- switch 345
- switch-case 77
- syntaktischen Fehler 153
- Syntax 50, 61
- Systemeinführung 23
- Systemtests 23

## T

- Tagesbelastungskurve 196
- Temperatursysteme 171
- template 347
- Template-Klasse 235
- Test 20, 115, 155, 157
- Test des Programms 115
- TextBox 236, 302
- Textstrom 109
- this 347
- throw 347
- Top-down-Verfahren 119
- Trapezregel 293
- Treffpunktkoordinaten 185
- true 348
- try 348
- try-catch 235
- typedef 71
- Typecasting 54
- typedef 345
- typeid 348
- typename 348
- Typkonvertierung 236
- Typmodifizierer 53
- Typumwandlung 236, 257
- Typumwandlungsoperator 71

## U

- Übergangsstelle 152
- überladene Methode 219, 238
- UML 217, 222
- unäre Operatoren 65
- Ungleichheit 68
- union 345
- unsigned 52, 54, 108
- Untersumme 290
- using 348

## V

- Variable 51f., 63
- Vererbung 228, 240
- Vergleichsoperatoren 68
- Verhalten einer Klasse 217
- Verschachtelung 120, 124, 139
- Verzweigung 114, 120, 125 f.
- Verzweigungssymbol 152
- virtual 348
- Visual C++ 13
- void 90, 345
- volatile 345
- Volladdierer 334
- vorprüfende Schleife 132
- Vorstudie 22
- Vorzeichenoperator 65

## W

- Wahrheitstabelle 333, 339
- Wartbarkeit 89
- wchar\_t 348
- Weltkoordinaten 243
- Wendestellen 314
- while 83, 345
- Whitespace 108
- Widerstandsreihe 260
- Wiederanlaufverfahren 157
- Wiederholung 79, 114, 129
- Wiederverwendbarkeit 89

## Z

- zählergesteuerte Schleife 79, 82, 127, 145
- Zeichen 55
- Zeiger 102
- Zuweisung 66
- Zwei-Punkte-Form 247