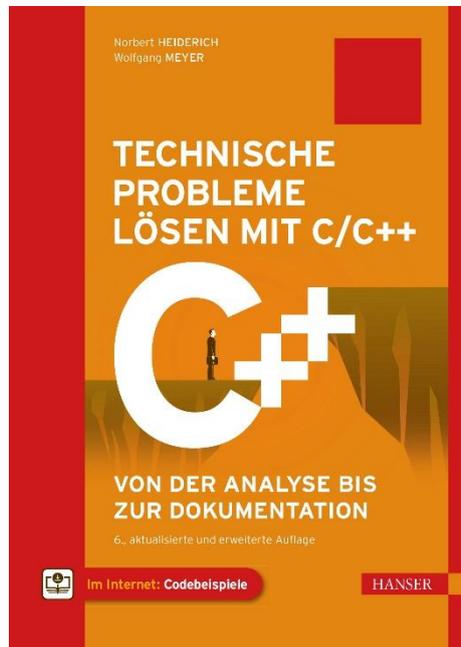


# HANSER



## Leseprobe

zu

## Technische Probleme lösen mit C/C++

von Norbert Heiderich und Wolfgang Meyer

Print-ISBN: 978-3-446-48020-9

E-Book-ISBN: 978-3-446-48090-2

Weitere Informationen und Bestellungen unter

<https://www.hanser-kundencenter.de/fachbuch/artikel/9783446480209>

sowie im Buchhandel

© Carl Hanser Verlag, München

# 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*

# Vorwort der Autoren

Die Zukunft unserer Arbeitswelt ist digital!

Programmieren ist der Einstieg in die digitale Welt und ist neben Lesen, Schreiben und Rechnen zu einer Schlüsselqualifikation des 21. Jahrhunderts geworden. Ohne Programmierkenntnisse ist ein beruflicher Ein-, Auf- oder Umstieg nicht mehr möglich.

C/C++ ist eine in der Wirtschaft sehr weit verbreitete Programmiersprache. Das Erlernen anhand konkreter Problemstellungen ist eine praxisorientierte Investition in die eigene berufliche Zukunft.

Der von den Autoren gewählte pädagogische Ansatz *so wenig Theorie wie nötig, so viel Praxis wie möglich* kommt bei den Leserinnen und Lesern offensichtlich gut an. Aber nichtsdestotrotz ist ein theoretischer Hintergrund für das Schreiben guter Programme und die spätere Fehleranalyse unverzichtbar.

In die sechste Auflage haben wir zu der umfangreichen Sammlung an Beispielen, Aufgaben und nach Schwierigkeit gestaffelten Problemstellungen zusätzlich Sortieralgorithmen aufgenommen. In einem Benchmark-Programm vergleichen wir Bubble-, Ripple-, Linear-, Quick- und Shell-Sort bezüglich ihrer Rechenzeit miteinander. Ferner lernen Sie, wie C/C++ Echtzeiten und Zeitdifferenzen ermittelt. Und zur Steigerung der Motivation können Sie sich einen Vorschlag für den wöchentlichen Lottoschein ausgeben lassen, natürlich „ohne Gewähr“!

Die Autoren wünschen Ihnen mit der sechsten Auflage viel Freude und gute Ergebnisse beim Programmieren mit C/C++.

*Norbert Heiderich*

*Wolfgang Meyer*



# Inhalt

<b>Vorwort des Herausgebers</b> .....	<b>5</b>
<b>Vorwort der Autoren</b> .....	<b>7</b>
<b>Einleitung</b> .....	<b>15</b>
<b>1 Systematik der Problemlösung</b> .....	<b>19</b>
1.1 Phasen der Programmentwicklung .....	19
1.2 Software-Lebenszyklus .....	21
1.3 Software-Entwicklungsverfahren .....	23
<b>2 Erste Gehversuche mit C/C++</b> .....	<b>28</b>
2.1 Warum gerade C/C++? .....	28
2.2 Compiler und Interpreter .....	30
2.3 Übersetzen eines C/C++-Programms .....	32
2.4 Programmstart .....	33
<b>3 Die Entwicklungsumgebung Visual C++</b> .....	<b>34</b>
3.1 Installation von VC++ .....	34
3.2 Starten von VC++ .....	36
3.3 Erstellen eines neuen Projektes .....	38
3.3.1 Win32-Projekte .....	39
3.3.1.1 Variante 1 - VC++ leistet Vorarbeit .....	40
3.3.1.2 Variante 2 - leeres Projekt .....	41
3.3.2 CLR-Projekte .....	44
3.4 Übersetzen eines eigenen Programms .....	46
3.5 Ausführen eines eigenen Programms .....	49
3.6 Paradigmen der Projektorganisation .....	49
3.7 Ein erstes Programm in C/C++ .....	51

<b>4</b>	<b>Grundlegende Sprach- und Steuerungselemente</b>	<b>54</b>
4.1	Kommentare	54
4.2	Datentypen und Variablen	55
4.2.1	Variablenamen	56
4.2.2	Ganzzahlige Variablen	56
4.2.3	Fließkommazahlen	58
4.2.4	Zeichen	59
4.2.5	Felder	60
4.2.5.1	Eindimensionale Felder	60
4.2.5.2	Mehrdimensionale Felder	61
4.2.5.3	Zugriff auf die Elemente eines Feldes	63
4.2.5.4	Startwertzuweisung für ein- und mehrdimensionale Arrays	65
4.2.6	Zeichenketten	67
4.3	Konstanten	68
4.4	Operatoren	69
4.4.1	Vorzeichenoperatoren	69
4.4.2	Arithmetische Operatoren	69
4.4.2.1	Addition +	69
4.4.2.2	Subtraktion -	69
4.4.2.3	Multiplikation *	70
4.4.2.4	Division /	70
4.4.2.5	Modulo %	70
4.4.2.6	Zuweisung =	70
4.4.2.7	Kombinierte Zuweisungen	71
4.4.2.8	Inkrementierung ++	71
4.4.2.9	Dekrementierung -	72
4.4.3	Vergleichsoperatoren	72
4.4.3.1	Gleichheit ==	72
4.4.3.2	Ungleichheit !=	72
4.4.3.3	Kleiner <	73
4.4.3.4	Größer >	73
4.4.3.5	Kleiner gleich <=	73
4.4.3.6	Größer gleich >=	74
4.4.4	Logische Operatoren	74
4.4.4.1	Logisches NICHT !	74
4.4.4.2	Logisches UND &&	74
4.4.4.3	Logisches ODER	74
4.4.5	Typumwandlungsoperator	75
4.4.6	Speicherberechnungsoperator	75
4.4.7	Bedingungsoperator	76
4.4.8	Indizierungsoperator	77
4.4.9	Klammerungsoperator	77
4.5	Anweisungen und Blöcke	79
4.6	Alternationen	79
4.6.1	Einfache Abfragen (if - else)	79

4.6.2	Mehrfachabfragen (else - if) .....	80
4.6.3	Die switch-case-Anweisung .....	81
4.7	Iterationen .....	83
4.7.1	Zählergesteuerte Schleifen (for) .....	83
4.7.2	Kopfgesteuerte Schleifen (while) .....	87
4.7.3	Fußgesteuerte Schleifen (do - while) .....	88
4.7.4	Schleifenabbruch (continue) .....	89
4.7.5	Schleifenabbruch (break) .....	90
4.7.6	Schleifenumwandlungen .....	92
4.8	Funktionen .....	92
4.8.1	Formaler Aufbau einer Funktion .....	93
4.8.1.1	Der Funktionskopf .....	94
4.8.1.2	Der Funktionsrumpf .....	95
4.8.2	Datentyp und Deklaration einer Funktion - Prototyping .....	96
4.8.3	Das Prinzip der Parameterübergabe .....	101
4.8.3.1	Aufrufverfahren call by value .....	101
4.8.3.2	Aufrufverfahren call by reference .....	103
4.8.3.3	Adressoperator, Zeiger und Dereferenzierung .....	106
4.8.4	Regeln für ein erfolgreiches Prototyping .....	107
4.8.5	Die exit()-Funktion .....	108
4.8.6	Rekursive Funktionen .....	108
4.9	Ein- und Ausgabe .....	111
4.9.1	Formatierte Eingabe mit scanf() .....	111
4.9.2	Formatierte Ausgabe mit printf() .....	112
4.9.3	Arbeiten mit Dateien .....	113
4.9.3.1	Öffnen der Datei .....	114
4.9.3.2	Verarbeiten der Datensätze .....	114
4.9.3.3	Schließen der Datei .....	115
4.9.3.4	stdio.h .....	115
4.9.3.5	fflush() und stdin .....	117
<b>5</b>	<b>Strukturierte Programmierung .....</b>	<b>118</b>
5.1	Problemstellung .....	119
5.2	Problemanalyse .....	120
5.3	Struktogramm nach Nassi-Shneiderman .....	123
5.3.1	Sequenz .....	125
5.3.2	Alternation .....	127
5.3.3	Verschachtelung .....	128
5.3.4	Verzweigung .....	129
5.3.5	Schleifen .....	131
5.3.5.1	Zählergesteuerte Schleife .....	131
5.3.5.2	Kopfgesteuerte Schleife .....	135
5.3.5.3	Fußgesteuerte Schleifen .....	137
5.3.5.4	Endlosschleifen .....	138
5.3.5.5	Kriterien zur Schleifenauswahl .....	138
5.3.6	Programm- oder Funktionsaufruf .....	138

5.3.7	Aussprung	139
5.3.8	Rechnergestützte Erstellung von Struktogrammen	140
5.3.8.1	StruktEd	140
5.3.8.2	hus-Struktogrammer	147
5.4	Flussdiagramm nach DIN 66001	155
5.5	Programmerstellung	157
5.6	Programmtest	157
5.7	Programmablauf	158
5.8	Dokumentation nach DIN 66230	159
5.8.1	Funktion und Aufbau des Programms	159
5.8.2	Programmkenndaten	160
5.8.3	Betrieb des Programms	161
5.8.4	Ergänzungen	161
5.9	Aspekte des Qualitätsmanagements EN-ISO 9000	162
5.10	Algorithmus – was ist das?	163
5.11	EVA-Prinzip	169
5.12	Programmierung von Formelwerken	170
<b>6</b>	<b>Lösung einfacher Probleme</b>	<b>175</b>
6.1	Umrechnung von Temperatursystemen	175
6.2	Flächenberechnung geradlinig begrenzter Flächen (Polygone)	181
6.2.1	Erste Problemvariation: Berechnung der Schwerpunkt- koordinaten $S(x_S; y_S)$ von polygonförmig begrenzten Flächen	188
6.2.2	Zweite Problemvariation: Suche nach einem „günstigen“ Treffpunkt	189
6.2.3	Eine Projektidee: Wohnflächenberechnung	190
6.3	Berechnung einer Brückenkonstruktion	191
6.4	Schaltjahrüberprüfung	195
6.5	Ein Problem aus der Energiewirtschaft	201
6.6	Logarithmische Achsenteilung	211
6.7	Berechnung der Kreiszahl $\pi$	218
6.7.1	Berechnung nach Archimedes (287 – 212 v. Chr.)	219
6.7.2	Berechnung mit der Monte-Carlo-Methode	221
6.7.3	$\pi$ in C/C++-Programmen	226
6.8	Primzahlen – Sieb des Eratosthenes	227
6.9	Lottozahlen	232
<b>7</b>	<b>Objektorientierte Programmierung (OOP)</b>	<b>235</b>
7.1	Modellbildung mittels Abstraktion	235
7.2	Klassen und Objekte	236
7.3	Attribute und Methoden einer Klasse	239
7.4	Bruchrechnung mit OOP	240
7.5	Vererbung	249
7.6	Strings	255
7.7	Typumwandlungen	257
7.8	Strukturierte Programmierung vs. OOP	260

<b>8</b>	<b>Lösung fortgeschrittener Probleme</b>	<b>262</b>
8.1	Grafische Darstellung funktionaler Abhängigkeiten	262
8.1.1	Welt- und Screenkoordinaten	264
8.1.2	Koordinatentransformationen	266
8.1.3	Darstellung der Sinusfunktion	272
8.1.4	Darstellung quadratischer Parabeln	276
8.1.5	Spannungsteilerkennlinien	279
8.2	Lösung technisch-wissenschaftlicher Probleme	281
8.2.1	Widerstandsreihen E6 bis E96	281
8.2.2	Farbcodierung von Widerständen nach DIN 41429	284
8.2.3	Fourier-Synthese periodischer empirischer Funktionen	287
8.2.4	Fourier-Analyse empirischer Funktionen	295
8.3	Nullstellenbestimmung von Funktionen	300
8.3.1	Inkrementverfahren und Intervallhalbierung	300
8.3.2	Die regula falsi	305
8.3.3	Das Newton-Verfahren	307
8.4	Numerische Integration	310
8.4.1	Riemannsches Unter- und Obersummen	310
8.4.2	Trapezregel	314
8.4.3	Simpsonsche Regel	319
8.4.4	Effektivwertberechnungen	324
8.4.5	Volumenberechnung	326
8.5	Einbindung eigener Klassen	334
8.5.1	Das „Platinenproblem“ als objektorientierte Konsolenanwendung	334
8.5.2	Das „Platinenproblem“ in der Erweiterung mit grafischer Benutzeroberfläche	339
<b>9</b>	<b>Lösung komplexer Probleme</b>	<b>343</b>
9.1	Kurvendiskussion und Funktionsplotter am Beispiel ganzrationaler Funktionen bis 3. Ordnung	343
9.2	Ausgleichsrechnung – Bestimmung der „besten“ Geraden in einer Messreihe	346
9.3	Digitaltechnik	356
9.4	Sortierverfahren	370
9.4.1	Bubble-Sort	370
9.4.2	Ripple-Sort	374
9.4.3	Sortieren alphanumerischer Daten mit Ripple-Sort	377
9.4.4	Sortieren von Listen nach vorgegebenen Kriterien	380
9.4.5	Benchmarkprogramm zur Beurteilung der Leistungsfähigkeit von Sortierprogrammen	386
9.4.5.1	Die Sortierprogramme Bubble-, Linear-, Ripple-, Quick- und Shell-Sort im Vergleich	386
9.4.5.2	Zeiterfassung in der Programmiersprache C/C++	387
9.4.5.3	Das Benchmarkprogramm	391
9.4.5.4	Interpretation der Vergleichsergebnisse	396
9.4.5.5	Die O-Notation	397
9.5	40 Jahre EDV mit dem PC – ein Blick zurück und nach vorn	398

<b>10 Tabellen und Übersichten</b> .....	<b>401</b>
10.1 Datentypen und ihre Wertebereiche .....	401
10.2 Vergleich der Symbole nach DIN 66 001 und der Nassi-Shneiderman-Darstellung .....	402
10.3 Schlüsselwörter ANSI C .....	403
10.4 Erweiterte Schlüsselwörter C++ .....	405
10.5 ASCII-Tabelle .....	408
10.6 Standardfunktionen und ihre Zuordnung zu den Header-Dateien (Include) .....	410
<b>Literatur</b> .....	<b>414</b>
<b>Index</b> .....	<b>415</b>

# Einleitung

Bücher, die sich mit Entwicklungsumgebungen beschäftigen, gibt es viele. Ebenso gibt es unzählige Werke, die die Programmiersprachen C und C++ beschreiben und sich mit den Vorteilen der einen gegenüber der anderen auseinandersetzen. Lehrbücher, die dem Leser den Weg vom konkreten Problem über Lösungsstrategien und Dokumentationshilfen bis hin zur rechnerunterstützten fertigen Lösung aufzeigen, sind hingegen Mangelware.

Niemand fängt bei dem Versuch, mit einer neuen Programmiersprache zu arbeiten, gerne ganz von vorne an. Daher werden Sie in diesem Buch Codebeispiele und exemplarische Vorgehensweisen finden, die Ihnen den Einstieg in die C/C++-Programmierung erleichtern. Das Buch möchte Sie als Leser bei der Lösung und Bearbeitung Ihrer Probleme unterstützen, Ihnen anhand vieler Beispiele unterschiedliche Einsatzmöglichkeiten von Programmier- und Dokumentationstechniken näherbringen, um Ihnen Schritt für Schritt den Weg in die professionelle Softwareentwicklung aufzuzeigen.

Zur Realisierung der Beispiele haben wir uns entschieden, mit Visual C++ (kurz: VC++) der Firma Microsoft zu arbeiten. Zum einen liegt mit VC++ ein hochmodernes Werkzeug zur effizienten Erstellung von Software jeden Anwendungs- und Komplexitätsgrades vor. Zum anderen ist die Verbreitung von VC++ so groß, dass die Wahrscheinlichkeit, dass Sie im professionellen Einsatz mit dieser Entwicklungsumgebung umgehen werden, sehr hoch ist. Darüber hinaus ist die Verfügbarkeit von VC++ als kostenloser Download der Firma Microsoft (siehe <https://www.visualstudio.com/downloads>) ein nicht zu unterschätzender Vorteil.

Grundsätzlich ist es natürlich auch möglich, eine andere Entwicklungsumgebung zu nutzen (z. B.: Code::Blocks, Eclipse, NetBeans IDE u. a.), allerdings können dann die von uns bereit gestellten Beispiele nicht ohne Anpassungen genutzt werden. Sie müssen in solchen Fällen die jeweiligen Quellcodes in Ihre Entwicklungsumgebung übertragen. Die Nutzung unserer Beispiele mit grafischen Oberflächen entfällt vollständig. Die entsprechenden Layouts müssen dann in Ihrer Entwicklungsumgebung nachempfunden werden. Alle relevanten Ideen können natürlich aus unseren Quellcodes ausgelesen werden.

## *Aufbau des Buches*

Das Buch ist in zehn Kapitel gegliedert, die jeweils ein spezielles Thema zum Inhalt haben. Bei der Auswahl der umgesetzten Beispiele haben wir versucht, sowohl technisch-wissenschaftliche, mathematische als auch kommerzielle Probleme zu thematisieren, um ein mög-

lichst großes Spektrum von Fragestellungen und damit verbundenen Problemlösungen aufzuzeigen.

Die Kapitel werden im Folgenden kurz skizziert:

### *Kapitel 1*

Im ersten Kapitel stehen Gesichtspunkte, die in einer hochvernetzten und globalisierten Arbeitswelt den Umgang mit zu lösenden Problemen diktieren, im Fokus der Betrachtung. Längst sind die Zeiten vorbei, in denen sich ein Softwareentwickler so intensiv mit seinem Problem auseinandersetzen konnte, bis er sicher war, eine optimale und anwendungsbreite Lösung gefunden zu haben. Die Maxime heute lautet: in kürzester Zeit ein brauchbares Ergebnis zu liefern.

### *Kapitel 2*

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.

### *Kapitel 3*

Im dritten Kapitel wird die Arbeitsweise von VC++ und die Arbeit mit dieser Entwicklungsumgebung beschrieben. Der Leser erfährt, wie ein komplexeres Projekt organisiert wird.

 Auf Installationshinweise auf zur jeweils aktuellen Version von VC++ haben wir absichtlich verzichtet, da Microsoft in sehr kurzen Zyklen neue Versionen zur Verfügung stellt, deren Installationen keinerlei Probleme darstellen. Die Kompatibilität von eingesetztem Betriebssystem und verwendeter Version der Entwicklungsumgebung von VC++ wird auf den Internet-Seiten von Microsoft dargestellt (s. a. <https://msdn.microsoft.com/de-de>).

### *Kapitel 4*

In diesem Kapitel werden die grundlegenden Sprach- und Steuerungselemente der Syntax der Programmiersprache C/C++ an einfachen Beispielen dargestellt.

### *Kapitel 5*

Dieses Kapitel enthält eine Einführung in die strukturierte Programmierung und ihre Darstellungsformen. Der Leser lernt rechnergestützte Systeme zur Erstellung von Struktogrammen kennen, die an Beispielen zunehmender Komplexität beschrieben werden. Außerdem werden die Bestandteile einer Software-Dokumentation beschrieben und die Frage beantwortet, was Software mit Qualität zu tun haben sollte.

### *Kapitel 6*

Im sechsten Kapitel werden die Kenntnisse der strukturierten Programmierung zunächst an einfachen Problemen angewendet. Von der Problemanalyse bis zur Ergebnisausgabe der Programme sind die Beispiele durchgängig dokumentiert.

### *Kapitel 7*

Dieses Kapitel gibt eine Einführung in die objektorientierte Programmierung. Der Leser lernt das erweiterte Vokabular und die Techniken der OOP kennen.

### *Kapitel 8*

Mit den Grundkenntnissen der OOP können in diesem Kapitel fortgeschrittene Probleme unter Zuhilfenahme von grafischen Oberflächen gelöst werden. Die Darstellung der Ergebnisse geschieht hier teilweise auch in zeichnerischer Form.

### *Kapitel 9*

Im neunten Kapitel werden komplexe Fragestellungen durch konsequente Anwendung aller erlernten Techniken bearbeitet. Die erzielten Lösungen sind dabei nach entsprechend gründlicher Problemanalyse verblüffend einfach.

### *Kapitel 10*

In Kapitel 10 finden Sie Tabellen und Übersichten, es stellt eine hilfreiche Zusammenfassung der für die Problemlösung erforderlichen Teilgebiete dar. Neben Datentypen, Symbolen für die Erstellung von Struktogrammen und Programmablaufplänen enthält es ein Glossar für den Sprachumfang von C und C++ und eine Tabelle oft genutzter Standardfunktionen. Außerdem finden Sie im Internet alle Code-Beispiele der Kapitel 6 bis 9 unter: *plus.hanserfachbuch.de*. Die Beispiele sind hier sowohl als PDF-Dateien mit den durchnummerierten Zeilen, auf die in den Erläuterungen verwiesen ist, als auch als Quelltexte, die Sie ohne mühsames Abtippen sofort in eigene Programme integrieren können, vorhanden. So lassen sich alle Beispiele unmittelbar erzeugen und ausführen. Einige Beispiele der Kapitel 8 und 9 finden Sie aufgrund ihrer Länge nur im Internet.

### *Zielgruppe des Buchs*

Das Buch wendet sich in erster Linie an Studierende an Fachschulen, Studenten technischer Studiengänge sowie an Auszubildende in den IT-Berufen. Aber auch Schülerinnen und Schüler in technisch orientierten Gymnasien und Fachoberschulen werden an der Vielfalt der Problemstellungen und der Herangehensweise an die Lösung – vom Problem über die Problemanalyse mit Struktogramm bis zum Testing und zur Dokumentation – sicherlich Gefallen finden.

Es liegt ein zeitgemäßes Buch vor. Zeitgemäß deshalb, weil wir den wichtigsten Schritt in der Programmentwicklung, die Problemanalyse, in den Mittelpunkt der Entwicklungsarbeit stellen und erst danach die Umsetzung mit VC++ besprechen. Denn ohne eine gründliche Problemanalyse haben Sie später keine Chance, logische Programmfehler zu finden. Übrigens verdient man in der Wirtschaft in diesem Bereich das meiste Geld.

Zeitgemäß ist das Buch auch dadurch, dass Sie ellenlange Quellcodes nicht mehr abtippen müssen, sondern einfach aus dem Internet herunterladen können. Das vereinfacht den Programmtest und die individuelle Fortentwicklung der Programme ganz gewaltig.

Um Ihnen die Orientierung im Buch zu erleichtern, haben wir Icons verwendet, die folgende Bedeutung haben:

	Bei den <b>Beispielen</b> im Buch finden Sie dieses Symbol.
	Wichtige <b>Hinweise</b> werden durch dieses Icon kenntlich gemacht.
	Dieses Symbol kennzeichnet <b>Aufgaben</b> .
	Die Darstellungen von <b>Lösungen</b> sind an diesem Icon zu erkennen.
	Hinweis auf einen Teil im <b>Internet</b> .

# 1

# Systematik der Problemlösung

Einst löste Alexander der Große den Gordischen Knoten sehr unkonventionell mit dem Schlag seines Schwertes. An den kunstvoll geknoteten Stricken, die einen Streitwagen untrennbar mit seinem Zugjoch verbinden sollten, waren zuvor die Gelehrten gescheitert. Sie versuchten, ihn ohne Beschädigung zu entfernen, quasi die Verknotungen umzukehren. Dies zeigt deutlich, dass ein Problem komplex und damit sogar unlösbar werden kann, wenn man nicht fähig ist, es unvoreingenommen zu betrachten, wenn man sich nicht von unvermeidbar erscheinenden Lösungswegen trennen kann. Die Lösung des Problems soll das Ziel sein – aber auch der Weg dorthin!

Zur Lösung eines Problems mithilfe eines Rechners geht man üblicherweise in mehreren Einzelschritten vor. Diese Vorgehensweise ist sinnvoll, weil die in jedem Schritt anfallenden Probleme häufig so speziell sind, dass Fachleute des jeweiligen Gebietes sie lösen müssen. So muss z. B. ein Betriebsführer, der eine Problemstellung sehr genau aus der Sicht des Betriebsablaufes beschreiben und sicherlich aus dieser Sicht auch erste Strategien entwickeln kann, nicht notwendigerweise auch derjenige sein, der mögliche Auswirkungen auf die Buchführung und Abrechnung des Unternehmens beurteilen, oder zur Auswahl geeigneter Programmiererelemente und einzusetzender Hardware einen Beitrag leisten kann.

## ■ 1.1 Phasen der Programmentwicklung

In den Anfängen der Datenverarbeitung waren Systemanalyse und methodisches Vorgehen bei der Entwicklung von Software beinahe bedeutungslos und der heute gebräuchliche Begriff **Softwareengineering** war noch nicht geprägt. Die erste Phase des Softwareerstellungsprozesses ist die Systemanalyse. Der Systemanalytiker beschreibt hier die für seine Fragestellung relevanten Elemente und deren Beziehungen zueinander.

Die ersten Rechner waren von den Abmessungen her groß und von der Leistungsfähigkeit aus heutiger Sicht sehr bescheiden. Hardware war so teuer, dass kleinere Unternehmen in der Regel die Verarbeitung ihrer Daten Service-Rechenzentren übergaben. Diese Rechenzentren entwickelten und warteten auch die individuellen Programme ihrer Kunden. Die eigene Datenverarbeitung im Hause bedeutete immense Investitionen, und die Software wurde dann mehr oder weniger individuell um die vorhandene Hardware „gestrickt“.

Die steigende Leistungsfähigkeit und der Preisverfall mit jeder neuen Generation von Rechnern eröffneten nach und nach immer neue Einsatzgebiete. So konnte man zunehmend integrierte Systeme entwickeln. Allerdings wurden mit dem wachsenden Integrationsgrad der Software die Programme und Programmsysteme komplexer.

Betrachtet man zu den Anfängen der Datenverarbeitung in mittleren bis großen Unternehmen das Verhältnis der Kosten von Hard- zur Software, so lag die bei etwa 85:15. Die gleiche Bewertung liefert heute ein Verhältnis von 10:90. Vergleicht man das Kostenverhältnis der Hard- zur Software im PC-Bereich, so ergibt sich für einen normalen Anwender in einem kleinen bis mittleren Betrieb ein ganz anderes Bild. Hier liegt das Verhältnis nahezu bei 50:50.

Der Einsatz von Datenverarbeitung in neuen Anwendungsgebieten ist primär ein Problem der Qualität, Funktionalität und Verfügbarkeit der Software zum richtigen Zeitpunkt und zu einem vertretbaren Preis. Damit wird deutlich, dass die Entwicklung von Software ein hochkomplexes Unterfangen ist und ein abgestimmtes, methodisches Verfahren und organisatorisches Vorgehen verlangt. Zusammengefasst wird dies unter dem Begriff **Softwareengineering**.

Softwareengineering wurde als Vorgehensweise zur Verbesserung der bis dahin unbefriedigenden Situation bei der Softwareentwicklung und -wartung betrachtet. Software sollte produziert werden können wie Produkte aus der industriellen Fertigung: solide, zuverlässig und kontrollierbar. Aus diesen Anfängen entwickelte sich die heutige Definition:



Unter **Softwareengineering** versteht man die Anwendung von Strategien, Methoden, Werkzeugen und Kontrollinstrumenten im gesamten Prozess der Softwareentwicklung und -wartung einschließlich des Managements.

Die Beschäftigung mit Softwareengineering setzt nun einen gewissen Erfahrungsschatz in der Softwareentwicklung voraus. Bei der **Softwareentwicklung im Kleinen** geht es um die Umsetzung überschaubarer Problemstellungen in rechnergestützte Lösungen. Dem Anwender der fertigen Software sollen möglichst viele, von ihm bisher evtl. mit anderen Hilfsmitteln erledigte Arbeitsschritte durch einen Rechner abgenommen werden. Dabei stehen die Auswahl und das Design einzelner Konstrukte im Vordergrund, was für die korrekte Funktionsweise und das spätere Verständnis eines Bausteins absolut wesentlich ist. Bei der **Softwareentwicklung im Großen** geht es um die zweckmäßige, fast generalstabsmäßige Organisation eines Arbeitsvolumens von vielen Mann-Jahren. (In der Informatik wird der Begriff Mann-Tage, Mann-Monate oder Mann-Jahre als Aufwandsmaß eines abstrakten Wesens verwendet, das während seiner Arbeitszeit weder männlich noch weiblich ist.)

In manchem ist das Softwareengineering mit der Arbeitsorganisation in herkömmlichen Produktions- und Konstruktionsprozessen vergleichbar. Softwareengineering beschäftigt sich mit Arbeitsabläufen in und um die Softwareentwicklung herum. Neben dem eigentlichen Entwicklungsprozess sind dies:

- Projektmanagement,
- Qualitätssicherung und
- Projektverwaltung.



Unter **Projektmanagement** versteht man die Gesamtheit von Führungsaufgaben bei der Abwicklung eines Projekts, z. B. Fragen der Projektorganisation.

Bei der **Qualitätssicherung** geht es einerseits um formelle, konstruktive und analytische Kontrollmaßnahmen während des gesamten Entwicklungsprozesses, andererseits um interpersonelle Techniken, also darum, dafür Sorge zu tragen, dass alle Aufgaben von möglichst geeigneten Mitarbeitern erledigt werden.

Die **Projektverwaltung** (auch: Konfigurationsmanagement) beschäftigt sich mit der Bereitstellung und Verwaltung aller Ressourcen für den Softwareentwicklungsprozess sowie mit allen nebengelagerten Prozessen. Dazu gehören u. a. die Organisation der Speicherung aller Programmvarianten einschließlich der Dokumentationen sowie die notwendigen Update-Dienste.

## ■ 1.2 Software-Lebenszyklus

Der Software-Lebenszyklus ist ein abstraktes Modell für den Lebenslauf einer jeden Software und die Grundlage für alle weiteren Betrachtungen zur Softwaretechnologie. Die meisten Aktivitäten, Methoden und Werkzeuge der Softwaretechnologie lassen sich anhand dieses Modells ein- und zuordnen. Für den konkreten Ablauf der Arbeit ist das Projektmanagement verantwortlich.

Der **Software-Lebenszyklus** stellt ein Modell für alle Aktivitäten während der Existenz einer Software dar. Man kann im Wesentlichen drei Teile unterscheiden:

- die eigentliche **Softwareentwicklung**, bei der das neue System aufgebaut wird;
- den **laufenden Betrieb**, währenddessen das System produktiv arbeitet, und
- die **Außerbetriebnahme** des Systems mit der Sicherstellung der Datenbestände für Nachfolgesysteme und der Entsorgung von Altdaten.

Während des laufenden Betriebs werden immer wieder ungeplante und geplante Unterbrechungen durch Wartung der eigentlich verschleißfreien Software erfolgen. Diese Wartungsarbeiten sind notwendig, um während des laufenden Betriebs festgestellte Fehler oder Effizienzverluste in den Programmen zu beheben oder die Software an geänderte Bedingungen des Umfeldes, in dem sie abläuft, anzupassen. Die Außerbetriebnahme einer Software erfolgt ebenso in der Regel aus dem laufenden Betrieb heraus. Schematisch lässt sich der **Software-Lebenszyklus** darstellen wie in Bild 1.1.

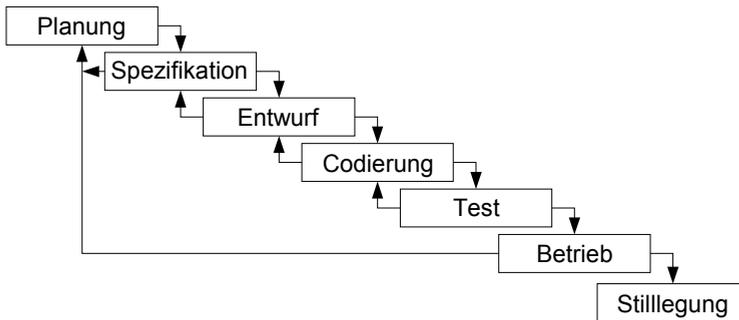


Bild 1.1 Software-Lebenszyklus

Bei der Entwicklung eines Systems werden die Zyklen Planung bis Test als Abfolge von einzelnen Phasen durchlaufen. In jeder Phase können unterschiedliche Mitarbeiter an der Realisierung des Projektes beteiligt sein, die ihre Ergebnisse jeweils für die nächste Phase zur Verfügung stellen. Der Betriebszyklus umfasst während der gesamten Lebensdauer des Systems dessen Unterhalt und Weiterentwicklung bis zur Außerbetriebnahme des Systems. Betrachtet man nun die Kostenseite, so verursachen die ersten vier Zyklen etwa 40 % der Gesamtsystemkosten; die restlichen 60 % der Kosten entfallen auf den Betrieb des Systems.

Die einzelnen Zyklen lassen sich inhaltlich folgendermaßen beschreiben:

- Die **Planung** umfasst eine Voruntersuchung des künftigen Systems mit den entsprechenden Wirtschaftlichkeitsberechnungen und bildet die Entscheidungsgrundlage die Rechtfertigung und somit die Freigabe zur Entwicklung des neuen Systems. In der Praxis wird dazu zunächst eine Studie beauftragt, über deren Ergebnis ein sog. Lenkungsausschuss befindet.
- Bei der **Spezifikation** werden die wesentlichen Anforderungen und Leistungsparameter des neuen Systems festgelegt. Dies ist gleichzeitig der Zeitpunkt der Erstellung eines sog. Pflichtenheftes, das eine exakte Beschreibung des zu erstellenden Systems liefert und die Basis bildet für die Programmdokumentation und das Anwenderhandbuch.
- Der **Entwurf** des Systems schlüsselt die Anforderungen und Leistungsparameter schrittweise auf bis ein Detaillierungsgrad erreicht ist, bei dem die fachlichen Anforderungen und der fachliche Lösungsweg in Form von Elementarprozessen umfassend beschrieben sind. Am Ende müssen alle fachlichen und datenverarbeitungstechnischen (kurz: DV-technischen) Anforderungen festgelegt sein. Zu diesem Zeitpunkt ist eine umfassende Problemanalyse abgeschlossen, das Pflichtenheft liegt in seiner endgültigen Form vor und alle an der Erstellung der neuen Software beteiligten Personen verfügen über ausreichende Fachkenntnis, um den nächsten Schritt angehen zu können.
- Die **Codierung** umfasst die eigentliche Programmkonstruktion mit der Programmierung der neu zu erstellenden Software.
- Der **Test** dient der Aufdeckung von Entwurfs- und Codierungsfehlern. Werden Fehler entdeckt, so wird die Software zur Korrektur an die Codierungsphase zurückgewiesen. Lassen sich Fehler nicht lokal beheben, z. B. weil ihre Ursache bereits im Entwurf liegt, so wird die Software bis in die Entwurfsphase zurückverwiesen. Diese Testphase blockiert die weitere Entwicklung, bis eine sachlich und fachlich richtige Ausführung der einzelnen Programmkomponenten sowie des Gesamtsystems gesichert werden kann. Dabei

sollten Testhilfen eingesetzt werden, die sicherstellen, dass alle möglichen Fälle, die auftreten können, auch tatsächlich einmal durchlaufen worden sind.

- Der **Betrieb** einer Software wird bis zur Außerbetriebnahme immer wieder durch korrigierende oder geplante Wartung der Software unterbrochen. Das reicht von Eingriffen in die Konfigurationsdateien über das selektive Einspielen neuer Systemkomponenten (sog. Patches) bis hin zur Modifikation oder Neuentwicklung ganzer Systemteile. Besonders kritisch wird der Betrieb, wenn aus Sicherheitsgründen eine alte und eine neue Softwareversion parallel gefahren werden müssen.
- Bei der **Stilllegung** einer Software kommt es schließlich darauf an, wesentliche Nutzdaten sicherzustellen, die für die Konfiguration und Initialisierung von Nachfolgesystemen sonst erst aufwendig akquiriert werden müssten, möglicherweise datenschutzrelevante Daten zuverlässig aus dem System zu entfernen und alle Arten von Datenmüll zu beseitigen. Dies ist nicht nur eine Frage der vorbeugenden Hygiene im Rechnersystem, sondern wegen möglicher Fernwirkungen auf später zu installierende Software dringend notwendig.

## ■ 1.3 Software-Entwicklungsverfahren

Alle EDV-Projekte (EDV = elektronische Datenverarbeitung) haben einen typischen und gleichartigen im Software-Lebenszyklus bezeichneten Ablauf, der in einzelne Abschnitte unterteilt werden kann. Diese einzelnen Abschnitte oder Phasen lassen sich in einer sehr stark standardisierten Form darstellen und führen zu den Phasenmodellen. Prinzipiell kann jedes EDV-Projekt in zwei große Bearbeitungsbereiche, Entwurf und Realisierung, zerlegt werden. Jeder dieser beiden Blöcke muss für die weitere Bearbeitung in einzelne Abschnitte aufgesplittet werden. Ein Phasenmodell entsteht im Prinzip durch genaue Definition und Abgrenzung der einzelnen Abschnitte des Software-Lebenszyklus.

Eine zu grobe Unterteilung der einzelnen Phasen lässt einen großen Spielraum innerhalb der einzelnen Phase zu und erhöht damit die Fehlerwahrscheinlichkeit. Eine zu feine Unterteilung der Phasen verzögert die Bearbeitung wegen der häufigen Unterbrechungen durch externe Entscheidungen. Sinnvolle Phasenmodelle unterscheiden zwischen drei und sechs Phasen, in Abhängigkeit vom Projektumfang. Hier soll von einem 6-Phasenmodell wie in Bild 1.2 ausgegangen werden.

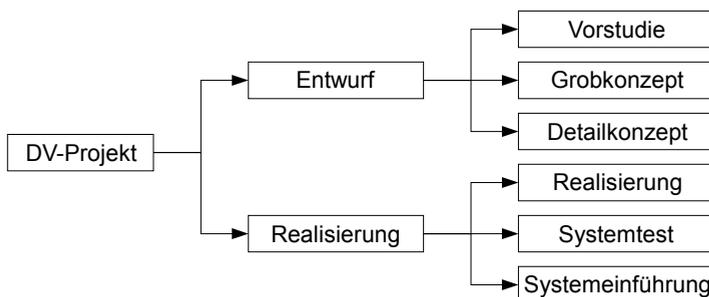


Bild 1.2 Das 6-Phasenmodell

Die einzelnen Phasen lassen sich wie folgt beschreiben:

- Die **Vorstudie** ist ein Abklärungsprozess, dem unmittelbar eine Entscheidung bezüglich der möglichen Lösungsvarianten folgt. Dabei wird die Zielrichtung für die Gestaltung des neuen Projektes festgelegt. Folgende Punkte müssen in einer Vorstudie enthalten sein:
  - Beschreibung der Ausgangslage und Begründung für die Entwicklung einer neuen Lösung
  - Konkrete Zielvorstellung
  - Vollständige Beschreibung des Ist-Zustandes und Schwachstellenanalyse
  - Vor- und Nachteile der heutigen Lösung mit Schwachstellenbeschreibung
  - Gestellte Anforderungen und Wünsche an die neue Lösung
  - Beschreibung der Lösung mit möglichen Alternativen
  - Bewertung der Lösung und der möglichen Alternativen
  - Wirtschaftlichkeitsüberlegungen
  - Planung und Freigabe der nächsten Phase
- Auf der Basis der in der Vorstudie favorisierten Lösungsmöglichkeit muss eine generelle Lösung mit den möglichen Varianten in einem betrieblichen und DV-technischen **Grobkonzept** erarbeitet werden. Die Lösung muss hier so detailliert sein, dass eine fachliche und sachliche Beurteilung und Bewertung möglich ist. Inhalt dieser Phase ist:
  - EDV-technische Konzeption der Funktionen, Abläufe, Transaktionen, Datenstrukturen, Festlegung der Verarbeitungsmodalitäten und des weiteren Vorgehens
  - Betriebliche Konzeption der Funktionen, Abläufe, Transaktionen, Layouts, Ausfallverfahren, Verarbeitungsmodalitäten und weiteres Vorgehen
  - Definition der betrieblichen Einführungsstufen
  - Test- und Einführungskonzeption
  - Überprüfung der Lösung
  - Wirtschaftlichkeitsberechnungen
  - Planung und Freigabe für die nächste Phase
- In der Phase **Detaillkonzept** ist das komplette fachliche und technische Systemdesign definitiv und abschließend zu erarbeiten. Ungelöste Probleme sind in dieser Phase nicht mehr zulässig. Die EDV-technische und betriebliche Machbarkeit muss sichergestellt sein.
  - Detaillierung und Komplettierung der EDV-technischen Konzeption
  - Detaillierung und Komplettierung der betrieblichen Konzeption
  - Detaillierung und Komplettierung der betrieblichen Einführungsstufen
  - Detaillierung und Komplettierung der Test- und Einführungskonzeption
  - Überprüfung aller Konzeptionen
  - Wirtschaftlichkeitsberechnungen
  - Planung und Freigabe für die nächste Phase
- Die Phase **Realisierung** stellt die reine Umsetzung der erstellten Konzeption in Programme dar. Zu diesem Zeitpunkt muss die komplette Dokumentation, wie Benutzerhandbücher und Operatorhandbuch, vorliegen.

# Index

## Symbole

<chrono>- 387  
<ctime> 387  
6 aus 49 232

## A

Abfrage 79  
abgeleitete Klasse 249  
Ablauflinie 156  
abstrakte Klasse 238  
Abstraktion 235  
abweisende Schleife 136  
Achsenbeschriftung 280  
Addition 69  
Adressoperator 106f.  
Aggregation 249  
Aggregatobjekt 249  
Algorithmus 163, 182, 239  
Alternation 79, 124, 127f., 140, 145, 154  
Anpassungshinweise 161  
Anweisungen 79  
Archimedes 219  
arithmetischer Operator 69  
Array 60, 67  
- einer Struktur 377  
ASCII-Tabelle 408  
ASCII-Zeichensatz 59  
Assoziation 249  
Assoziativität 78  
Attribut 237 ff., 323  
Attributwert 238, 274  
Aufgabe 160  
Aufgabenlösung 159  
Aufgabenstellung 159

Ausgabesymbol 156  
Ausgleichsrechnung 346  
Ausnahmefehler 274  
Außerbetriebnahme 21  
Ausprung 125, 139  
Auswahl 118, 142  
auto 403

## B

Bachmann-Landau-Notation 397  
Basisklasse 249  
Batchdatei 108  
Bedienung 161  
Bedingungsoperator 76  
Bemerkung 156  
Benchmarkprogramm 386  
Betrieb 21 ff.  
Bezeichnung 160  
Bibliotheksfunktionen 100  
Bildpunkt 265  
binäre Operatoren 69  
Blöcke 79  
bool 406  
Boole, Georg 356  
break 82, 90, 131, 403  
Brückenkonstruktion 191  
Bubble-Sort 370

## C

call by reference 103  
call by value 101  
case 403  
case-sensitiv 56  
Cast 257

- catch 406
- char 56, 67, 403
- class 406
- clock\_gettime 388
- Codierung 22, 27
- const 403
- const\_cast 406
- Container-Klasse 255
- continue 89, 403
- Copy-and-paste 93
  
- D**
- Datentyp 55
- Datentypen und ihre Wertebereiche 401
- Debugger 158
- default 130, 403
- Default-Werte 280
- Definition der Funktion 96
- Dekade 211
- Dekrementierung 72
- delete 406
- Dereferenzierung 106 ff.
- Designfehler 33
- Deskriptoren 161
- destruktives Schreiben 114
- Destruktor 243
- Detailkonzept 24 ff.
- deterministische Verfahren 164
- Differenzialrechnung 343
- Digitalschaltung 358
- Digitaltechnik 356
- DIN 66001 155
- DIN 66230 159
- Division 70
- do 403
- Dokumentation 119, 142, 159, 242
- double 58, 403
- do - while 88
- Dualzahlen 359 f.
- dynamic\_cast 406
- dynamisches Array 225
- dynamische Speicherzuweisung 231
  
- E**
- Eckpunktkoordinate 182
- Effektivwertberechnungen 324
- eindimensionale Felder 60
- einfache Abfrage 79
  
- Eingabesymbol 156
- Elemente eines Feldes 63
- Elementverweis-Operator 256
- else 404
- else - if 80
- Endlosschleife 85, 131, 138
- Energiewirtschaft 201
- EN-ISO 9001 162
- Entität 237
- Entwurf 22 f., 26
- enum 404
- Eratosthenes 227
- E-Reihen 281
- Erstes Programm in C 51
- erweiterte Schlüsselwörter C++ 405
- euklidischer Algorithmus 168
- EVA-Prinzip 169
- Exception 274
- Exemplar 237
- explicit 406
- explizite Typumwandlungen 257
- extern 404 ff.
- externe Operation 240
- Extremstellen 343
  
- F**
- Fakultätsberechnung 164
- false 406
- Farbcodierung nach DIN 41429 284
- fclose() 115
- Fehler 33
- Fehlerbehandlung 161
- Fehlerquadratsumme 347
- Feld 60 ff.
- Feld, Felder 60
- fflush() 117
- fgetc() 114
- fgets() 114
- FILE 113
- Flächenberechnung nach Gauß 182
- Fließkommazahl 58
- float 58, 64, 404
- Flussdiagramm 155 ff.
- Font 278
- fopen() 114
- for 83, 404
- formatierte Ausgabe 112
- formatierte Eingabe 111
- Formelwerk 170 f.

Fourier  
- Analyse 295  
- Koeffizient 295  
- Reihen 288  
- Synthese 287  
fprintf() 115  
fputc() 114  
fputs() 114  
fread() 115  
friend 406  
fscanf() 115  
Funktion 92  
Funktions  
- aufruf 124, 138 f., 271  
- graph 276  
- kopf 93 f.  
- rumpf 93 ff.  
fußgesteuerte Schleife 88, 131, 137 f.  
fwrite() 115

## G

ganzzahlige Variablen 56  
Gaußscher Integralsatz 182  
Geheimnisprinzip 238  
Gerätebedarf 160  
get-Methode 243  
Gleichheit 72  
goto 404  
Graphical User Interface (GUI) 255  
Grenzstelle 156  
Grobkonzept 24  
größer 73  
größer gleich 74  
GUI 38, 235, 255

## H

Hauptprojektdatei 98  
Header-Datei 98, 410  
Heißeleiter 211, 218

## I

if 404  
if - else 79  
Implementierung 157  
Implementierungsaufwand 93  
implizite Typumwandlung 257  
Include 410

Indizierungsoperator 77  
Initialisierung 83  
Inkludierung 100  
Inkrementierung 71  
Inkrementverfahren 300  
inline 406  
Instanz 237, 255  
int 56 f., 60 f., 404  
IntelliSense 258 f.  
Intervallhalbierung 301  
Iteration 83, 132, 154

## K

Kapselung 237, 242  
Kennlinie 279  
Kennlinienfeld 279  
Klammerungsoperator 77  
Klassen 236 ff., 343  
- beschreibung 238  
- diagramm 238  
- hierarchie 249  
- operation 240  
kleiner 73  
kleiner gleich 73  
kombinierte Zuweisung 71  
Kommentar 54  
Komponente 34, 249  
Komposition 249  
Konstante 56, 61, 68  
Konstruktor 240 ff.  
Kontrollstruktur 79  
Koordinatentransformation 266  
kopfgesteuerte Schleife 86 f., 136 ff.  
Kosinus-Koeffizient 295  
Kreiszahl  $\pi$  218

## L

LARGE\_INTEGER 390  
least square fit 347  
Leibniz 359  
Linear-Sort 386  
logarithmische Achsenteilung 211 f.  
logischer Fehler 157  
logisches NICHT 74  
logisches ODER 74  
logisches UND 74  
long 56 f., 112, 404  
long int 56

## M

malloc() 185  
 math.h 227  
 mehrdimensionale Felder 61  
 Mehrfachabfrage 80  
 Mehrfachauswahl 405  
 Message-Box 274  
 Messreihe 346  
 Methode 237 ff., 278 ff.  
 Methode überladen 240  
 mode 114  
 Modellbildung 235  
 Modul 93  
 Modulo-Operator 70  
 Monte-Carlo-Methode 221  
 Multiplikation 70  
 mutable 407

## N

nachprüfende Schleife 137  
 namespace 407  
 Nassi-Shneidermann 123, 157  
 new 407  
 Newton-Verfahren 307  
 nicht-abweisende Schleife 137  
 NTC 211  
 Nullstellenbestimmung 300  
 numerische Integration 310  
 Nutzungsvereinbarungen 161

## O

Obersumme 312, 327  
 Objekt 236 f., 274  
 Objektoperation 240  
 objektorientierte Programmierung (OOP)  
   118, 156, 235  
 O-Notation 397  
 OOP 116  
 Operation 239  
 Operator 69, 407  
 Ordinalwert 59 f.  
 Overloading 240

## P

Parabel 276  
 Paradigma 118

Parameterübergabe 101  
 Pivotelement 386  
 Planung 22  
 Pointer 106  
 Polygone 181  
 Postfix 71  
 Potenzfunktion 129  
 pow() 129  
 Präfix 71  
 Präprozessoranweisung 100  
 Primzahlen 227  
 printf() 111, 134  
 Priorität 78  
 private 239, 242, 250, 407  
 Problemanalyse 119 f., 170, 175  
 Problemstellung 119, 175  
 Programm 175  
   - ablauf 158 ff., 257  
   - ablaufplan 155  
   - aufbau 160  
   - aufruf 124  
   - bedarf 160  
   - erstellung 119, 157  
   - lauf 119, 146, 158, 175  
   - test 175, 185  
 Programmierparadigma 118  
 Programmiersprache 161  
 Projektmanagement 20 f.  
 Projektverwaltung 20  
 protected 239, 243, 407  
 Protokoll einer Klasse 239  
 Prototyp 97  
 Prozeduren 93, 118  
 Pseudozufallszahlen 223  
 public 239, 242, 250, 407

## Q

Qualitätsmanagement 162  
 Qualitätssicherung 20, 163  
 QueryPerformanceCounter 389, 392  
 QueryPerformanceFrequency 390 ff.  
 Quick-Sort 386

## R

rand() 223  
 Realisierung 23 ff.  
 Rechenzeiten 398  
 Rechteckfunktion 289, 297

Referenz 103 ff.  
register 404  
regula falsi 305  
Reinitialisierung 84  
reinterpret\_cast 407  
Rekursion 108  
return 404  
Riemannsche Unter- und Obersummen 310  
Ripple-Sort 374  
Rotationskörper 326  
RSA-Verfahren 227  
Rundungsfehler 59

## S

scanf() 111, 136  
Schaltjahrüberprüfung 195  
Schaltnetz 357  
Schleifen 118, 124, 131, 148, 153, 197  
- abbruch 89 f.  
- bedingung 86  
- begrenzungssymbol 156  
- kopf 86  
- rumpf 86 ff.  
- steuerungsvariable 85  
- umwandlung 92  
Schlüsselwörter ANSI C 403  
Schnittstelle 160, 239  
Schriftart 278  
Schrittweitenwert 276  
Schwerpunktkoordinaten 188  
Screenkoordinaten 264  
Sequenz 118, 124, 139, 143  
set-Methode 243  
Shannon, Claude E. 356  
Shell-Sort 386  
short 56 f., 112, 404  
short int 56 ff., 357  
Sieb des Eratosthenes 227  
signed 57 f., 404 f.  
Signifikanz 58  
Simpsonsche Regel 319  
Sinusfunktion 262 ff., 272  
Sinus-Koeffizient 295  
sizeof 404  
Skalierung 276  
Softwareengineering 19  
Softwarelebenszyklus 21  
SolidBrush 278  
Sortieralgorithmen 370

Sortierverfahren 370  
Spannungsteiler 279  
Spannungsteilerkennlinie 279  
Spannungsteilerwiderstand 279  
Speicherbedarf 160  
Speicherberechnungsoperator 75  
Spezifikation 22  
sprachbedingte Fehler 33  
sqrt() 129, 317  
srand() 223  
Standardeingabe 87  
Standardfunktionen 410  
Stapelverarbeitungsdatei 108  
Startbedingung 85  
Startwertzuweisung 65  
static 404  
static\_cast 407  
stdin 87  
stdio.h 113  
Stilllegung 23  
STL 255 f.  
strcmp 378  
strcpy 378  
Stream 113  
string 199, 256  
String 255  
struct 404  
Struktogramm 123, 155, 175  
Struktur 113, 239, 349, 377, 381  
strukturierte Programmierung 16, 118  
Subtraktion 69  
switch 405  
switch-case 81  
syntaktischer Fehler 157  
Syntax 54, 65  
Systemeinführung 25  
Systemtests 25  
SYSTEMTIME 388

## T

Tagesbelastungskurve 201  
Temperatursysteme 175  
template 407  
Template-Klasse 255  
Test 22, 119, 159 ff.  
Test des Programms 119  
TextBox 256, 323  
Textstrom 113  
this 407

throw 407  
 Top-down-Verfahren 123  
 Trapezregel 314  
 Treffpunktkoordinaten 189  
 true 408  
 try 408  
 try-catch 256  
 typecast 75  
 Typecasting 58  
 typedef 405  
 typeid 408  
 typename 408  
 Typkonvertierung 257  
 Typmodifizierer 57  
 Typumwandlung 257, 278  
 Typumwandlungsoperator 75

## U

Übergangsstelle 156  
 überladene Methode 240, 259  
 UML 238, 243  
 unäre Operatoren 69  
 Ungleichheit 72  
 union 405  
 unsigned 56 ff., 112  
 Untersumme 311, 327  
 using 408

## V

Variable 55 f., 67  
 Vererbung 249, 261  
 Vergleichsoperatoren 72  
 Verhalten einer Klasse 238  
 Verschachtelung 124, 128, 143

Verzweigung 118, 124, 129 f.  
 Verzweigungssymbol 156  
 virtual 408  
 Visual C++ 15  
 void 94, 405  
 volatile 405  
 Volladdierer 363  
 Volumenberechnung 326  
 vorprüfende Schleife 136  
 Vorstudie 24  
 Vorzeichenoperator 69

## W

Wahrheitstabelle 362, 368  
 Wartbarkeit 93  
 wchar\_t 408  
 Weltkoordinaten 264  
 Wendestellen 343  
 while 87, 405  
 Whitespace 112  
 Widerstandsreihe 281  
 Wiederanlaufverfahren 161  
 Wiederholung 83, 118, 133  
 Wiederverwendbarkeit 93  
 Wohnflächenberechnung 190

## Z

zählergesteuerte Schleife 83, 86, 131, 149  
 Zeichen 59  
 Zeiger 106  
 Zeit-Messung 392  
 Zufallszahlen 221  
 Zuweisung 70  
 Zwei-Punkte-Form 268