

C programmieren

Das umfassende Handbuch

DAS INHALTS- VERZEICHNIS



» Hier geht's
direkt
zum Buch

Inhalt

Materialien zum Buch	31
Vorwort	33
1 Die Philosophie von C	41
1.1 Was ist C, und was ist es nicht?	41
1.1.1 Ursprung und Motivation	41
1.1.2 Was C ausmacht	43
1.1.3 Was C nicht ist	43
1.1.4 C im Vergleich zu C++	44
1.2 Der C-Standard	44
1.2.1 Versionen von C	45
1.2.2 Die C-Standardbibliothek und ihre Header	47
1.2.3 Ein Kommen und Gehen	50
1.2.4 Drafts	50
1.2.5 Gemeinsame Wurzeln und unterschiedliche Entwicklung von C und C++	50
1.2.6 Abseits des Standards	51
1.3 Die Bedeutung von C heute	52
1.3.1 Warum C immer noch so wichtig ist	53
1.3.2 C als Fundament der digitalen Welt	53
1.3.3 Aktuelle Popularität	54
1.4 Rückblick und Ausblick	55
2 Werkzeugkette und das erste C-Programm	57
2.1 Minimales Beispiel main.c	57
2.1.1 Ein kleiner Exkurs zur Programmsyntax	59
2.1.2 Host-Umgebungen und Freestanding-Umgebungen	60
2.2 Compiler im Alltag: GCC und Clang	60
2.2.1 GNU Compiler Collection (GCC)	62
2.2.2 LLVM und Clang	65
2.2.3 Compiler in Containern	66
2.2.4 Der Microsoft-Compiler (MSVC/cl)	66
2.3 C-Programm auf der Kommandozeile kompilieren und ausführen	68
2.3.1 Sprachunterstützung der Compiler	69
2.3.2 Sprachstandard explizit festlegen (ISO C)	71

2.3.3	Warnungen einschalten	71
2.3.4	GNU-Standard	72
2.4	Präprozessor, Compiler und Linker	73
2.4.1	Übersetzungsphasen explizit ausführen (GCC/Clang)	74
2.5	Editoren, IDEs	74
2.5.1	IDEs und Editoren im Überblick	76
2.5.2	Webbasierte IDEs	77
2.5.3	Compiler Explorer	78
2.6	Debugger und Laufzeitanalyse	80
2.6.1	Debug vs. Release	80
2.6.2	Breakpoints, Stepping, Watchpoints	81
2.6.3	Remote Debugging	82
2.7	Dokumentation zu C-Funktionen finden	82
2.8	Aufgabe	82
2.9	Rückblick und Ausblick	83
3	Syntax und Struktur von C-Programmen	85
3.1	Vom Quelltext zum Programm	85
3.1.1	Translation Units und Header-Dateien	85
3.2	Lexikalische Grundlagen	86
3.2.1	Schlüsselwörter	86
3.2.2	Bezeichner	87
3.2.3	Unicode und UTF-8	87
3.2.4	Konstanten und String-Literale	87
3.2.5	Punctuators und Operatoren	88
3.2.6	Kommentare	88
3.2.7	Aufgabe	90
3.3	Lesbarkeit und Stil	90
3.4	Grammatik und Programmbausteine	91
3.4.1	Deklarationen und Definitionen	91
3.4.2	Anweisungen und Blöcke	92
3.4.3	Das Semikolon	92
3.5	Rückblick und Ausblick	93

4	Variablen, Datentypen und Typsicherheit	95
4.1	Konstanten und String-Literale: feste Angaben im Quelltext	95
4.1.1	Ganzzahlkonstanten, Gleitkommakonstanten, Zeichenkonstanten und String-Literale	95
4.1.2	Literalverkettung	96
4.2	Werte mit der Funktion zur formatierten Ausgabe printf() ausgeben	97
4.2.1	Nicht passende Konvertierungsspezifikationen	98
4.2.2	Dezimale Gleitkommakonstanten mit Exponent	99
4.2.3	Aufgabe	99
4.3	Grundtypen	99
4.4	Variablen	100
4.4.1	Deklaration von Variablen	100
4.4.2	Deklaration mit Initialisierung	104
4.4.3	Werte einlesen (scanf)	105
4.4.4	Typableitung (Type Inference) (auto)	110
4.5	Konstante Werte	112
4.5.1	Konstante Werte (const)	112
4.5.2	Konstant berechenbare Ausdrücke (constexpr)	114
4.5.3	West gegen Ost const	115
4.5.4	Gültigkeitsbereich und Lebensdauer	115
4.6	Ganzzahlen (int, short, long, long long)	117
4.6.1	Ganzzahltypen mit implementierungsdefinierter Bitbreite	117
4.6.2	Datenmodelle	119
4.6.3	Ganzzahltypen mit fester, exakt definierter Breite	120
4.7	Gleitkommatypen (float, double, long double)	120
4.7.1	Spezielle Werte: $\pm\infty$, NaN und vorzeichenbehaftete Nullen	121
4.7.2	Stillschweigend gerundet	123
4.8	Zeichentyp (char)	123
4.8.1	Escape-Sequenzen	125
4.8.2	Hexadezimale Escape-Sequenzen	126
4.8.3	Mehrzeichen-Zeichenkonstante	127
4.8.4	Wie der C-Standard ein Byte bestimmt	127
4.9	Wahrheitswerte (bool)	128
4.9.1	Wie C lernte, wahr und falsch zu unterscheiden	129
4.10	Der Nicht-Wertetyp (void)	129
4.11	Vorzeichenbehaftete und vorzeichenlose Ganzzahlen	130
4.11.1	Einführung in vorzeichenbehaftete und vorzeichenlose Typen	130
4.11.2	Sonderfall char	132

4.11.3	Wertebereiche und Darstellung	133
4.11.4	Zweierkomplementdarstellung	135
4.11.5	Überlaufverhalten	136
4.11.6	Mischen von vorzeichenbehafteten und -losen Datentypen	138
4.12	Zahlendarstellungen und Schreibweisen	140
4.12.1	Oktal, Hexadezimal und Binär	140
4.12.2	Suffixe für Ganzzahl- und Gleitkommakonstanten (U, L, f ...)	141
4.12.3	Zifferntrenner	144
4.12.4	Aufgabe	145
4.13	Typkategorien und Grundbegriffe	145
4.14	Rückblick und Ausblick	145
5	Operatoren und Ausdrücke	147
5.1	Primäre Ausdrücke	148
5.2	Zuweisungsoperatoren	149
5.2.1	Einfache Zuweisung	149
5.2.2	L-Values und R-Values	150
5.2.3	Zuweisung als Ausdruck	151
5.3	Arithmetische Operatoren (+, -, *, /, %)	152
5.3.1	Unäre und binäre Operatoren	152
5.3.2	Grundlegende arithmetische Operatoren	153
5.3.3	Restwertberechnung	155
5.3.4	Aufgaben	157
5.4	Kurzschreibweisen für Zuweisungen und Änderungen	157
5.4.1	Erweiterte Zuweisungen	157
5.4.2	Präfixformen und Postfixformen	158
5.5	Vergleichsoperatoren	160
5.5.1	Gleichheitsoperatoren (==, !=)	161
5.5.2	Relationale Operatoren (<, <=, >, >=)	164
5.5.3	Vergleiche mit Gleitkommazahlen	165
5.6	Logische Operatoren (!, &&,)	165
5.6.1	Der Negationsoperator (!)	166
5.6.2	Der UND-Operator (&&)	167
5.6.3	Der ODER-Operator ()	167
5.6.4	Zusammengesetzte logische Ausdrücke	168
5.6.5	Abgeleitete logische Operatoren	169
5.6.6	Kurzschlussverhalten	170

5.6.7	Arithmetische Tricks mit booleschen Werten	172
5.6.8	Aufgabe	173
5.7	Typumwandlungen (Conversions) und ihre Regeln	174
5.7.1	Integer-Promotion (Ganzzahlpromotion)	174
5.7.2	Implizite Konvertierungen (Automatic Conversions)	175
5.7.3	Explizite Konvertierungen (Casts)	178
5.7.4	Das Signed/Unsigned-Problem	179
5.7.5	Aufgaben	181
5.8	Kommaoperator, Vorrang, Assoziativität und Auswertungs- reihenfolge	181
5.8.1	Kommaoperator	182
5.8.2	Operatorpräzedenz	183
5.8.3	Assoziativität	183
5.8.4	Auswertungsreihenfolge (Sequenzierung)	185
5.9	Mathematisches aus den Standard-Bibliotheken	186
5.9.1	Mathematische Funktionen in <stdlib.h>	187
5.10	Rückblick und Ausblick	188
6	Kontrollstrukturen	189
6.1	Bedingte Anweisungen (Fallunterscheidungen: if, else)	189
6.1.1	Komplexe Bedingungen	191
6.1.2	Typische Fehler	192
6.1.3	if-else-Anweisung	197
6.1.4	else-if-Ketten für mehrere Bedingungen	199
6.1.5	Aufgaben	200
6.2	Bedingungsoperator (?:)	201
6.2.1	Verschachtelter Bedingungsoperator	203
6.2.2	Beispiel: Werte einklemmen	204
6.2.3	Bedingte Ausdrücke mit ausgelassenem Mittelteil (GCC)	205
6.3	switch-Anweisung	205
6.3.1	Default-Label	208
6.3.2	switch oder if?	209
6.3.3	Fall-through	209
6.3.4	Gültigkeitsbereich von Variablen	214
6.3.5	Aufgabe	215
6.4	while-Schleife	216
6.4.1	Endlosschleifen	217

6.4.2	Zuweisungen in der Bedingung	217
6.4.3	Aufgaben	221
6.5	do-while-Schleife	222
6.5.1	Konvertierung zwischen while und do-while	223
6.5.2	Aufgaben	224
6.6	for-Schleife	225
6.6.1	Varianten und leere Teile der for-Schleife	227
6.6.2	Mehrere Operationen mit dem Kommaoperator	228
6.6.3	Geschachtelte Schleifen	229
6.6.4	Aufgabe	231
6.7	Schleifen mit break und continue steuern	231
6.7.1	break	232
6.7.2	continue	232
6.7.3	Grenzen von break und continue	233
6.7.4	JSON-Dokumente formatieren (Pretty Printing)	234
6.7.5	Aufgabe	238
6.8	Best Practices, Tipps und Fehlerquellen bei Schleifen	239
6.8.1	Defensive Programmierung	239
6.8.2	Häufiger Fehler: Off-by-one-Fehler	240
6.8.3	Wahl der Schleifenart	242
6.8.4	Schleifenvariablen und Invarianten	242
6.8.5	Performance-Überlegungen	243
6.8.6	Abschlusstipps	245
6.9	Programmsprünge (goto)	246
6.9.1	Warum man bei goto aufpassen muss	247
6.9.2	goto zur Fehlerbehandlung	249
6.9.3	Aus verschachtelten Schleifen ausbrechen	249
6.9.4	Doppeldeutigkeit von break bei switch und Schleifen	250
6.9.5	Zustandsautomaten	251
6.10	Rückblick und Ausblick	253
7	Funktionen	255
7.1	Warum Funktionen?	255
7.1.1	Vorteile eigener Funktionen	256
7.2	Funktionsgrundlagen: Definition, Deklaration und Aufruf	257
7.2.1	Erstes Beispiel: eine Banner-Funktion	257
7.2.2	Benennung von Funktionen	258

7.2.3	Definition, Deklaration und Funktionsprototyp	259
7.2.4	Eine Definition pro Funktion	262
7.3	Parameterübergabe	263
7.3.1	Parameter und Argumente	264
7.3.2	Call by Value	266
7.3.3	Übliche Typkonvertierungen	266
7.4	Funktionen mit return verlassen	267
7.5	Funktionen mit Rückgabewerten	267
7.5.1	Kompakte Rümpfe bei boolean-Rückgaben	269
7.5.2	Was passiert, wenn ein Rückgabewert fehlt?	269
7.6	Überladen von Funktionen in C?	270
7.7	Rekursive Funktionen	272
7.7.1	Das Stack-Modell	273
7.7.2	Endlosrekursion	274
7.7.3	Weitere Rekursionsprobleme	274
7.7.4	Endrekursion (Tail Recursion) und Optimierung	275
7.7.5	Rekursion oder Iteration?	275
7.7.6	Aufgaben	275
7.8	Variadische Funktionen (variable Argumentlisten)	276
7.8.1	Syntax mit	277
7.8.2	Makros aus <stdarg.h>	277
7.8.3	Typinformation und Default Promotions	279
7.8.4	Anzahl und Typen der variablen Argumente erkennen	279
7.8.5	Stellenwert variadischer Funktionen	279
7.8.6	Aufgabe	279
7.9	Inline-Funktionen	280
7.9.1	Das inline-Schlüsselwort als Compiler-Hinweis	280
7.9.2	Lokale Hilfsfunktionen mit static inline	281
7.10	Attribute	282
7.10.1	Attribut-Syntax	282
7.10.2	Standard-Attribute	283
7.11	Fehlerbehandlung in Funktionen	287
7.11.1	Rückgabewerte als Fehlerindikatoren	288
7.11.2	Programm mit Fehlercode beenden	290
7.11.3	Fehlerbehandlung in C mit errno	290
7.11.4	Ein-/Ausgabe-Kanäle und Fehlermeldungen	292
7.11.5	Fehlermeldungen mit perror() und strerror()	293
7.11.6	Das Makro assert	295
7.12	Rückblick und Ausblick	298

8	Scope, Linkage, Storage Duration und Modularisierung	299
8.1	Scope, Linkage und Storage Duration von Variablen und Funktionen ...	300
8.1.1	Globale Variablen	301
8.1.2	Statische globale Variablen	305
8.1.3	Statische lokale Variablen	306
8.1.4	Globale Funktionen	307
8.1.5	Zusammenfassung: Sichtbarkeit (Scope) und Linkage	309
8.1.6	Storage Duration	310
8.1.7	Aufgabe	311
8.2	Storage Classes im Überblick	312
8.2.1	extern bei Funktionen	313
8.2.2	extern bei Variablen	315
8.2.3	auto	318
8.2.4	register	319
8.2.5	Aufgabe	321
8.3	Header-Dateien und Modularisierung	321
8.3.1	Module bilden	322
8.3.2	Funktionsprototyp im Header, Implementierung in .c-Dateien	323
8.3.3	#include und Präprozessor	325
8.3.4	#include <...> vs. #include "..."	327
8.3.5	Include-Pfade und -I	328
8.3.6	Header Best Practices	329
8.3.7	Aufgabe	343
8.4	Strukturierung größerer C-Anwendungen	344
8.4.1	Vom Problem zur Struktur	344
8.4.2	Module und Kapselung	344
8.4.3	Datenorientierter Entwurf	344
8.4.4	Abhängigkeiten und Architektur	345
8.4.5	Funktionen und Benennung	346
8.4.6	Evolution der Struktur	346
8.5	Rückblick und Ausblick	347
9	Der C-Präprozessor	349
9.1	Historische Entwicklung und Motivation	349
9.1.1	Rolle in den Übersetzungsphasen	350
9.1.2	Erstes Makro	350

9.1.3	Verwendung außerhalb von C	355
9.1.4	C++ und der Präprozessor	355
9.2	Makros definieren: #define	356
9.2.1	Wenn möglich, dann constexpr	356
9.2.2	Syntax-Makros	357
9.2.3	Konstanten-Makros	360
9.2.4	Funktionsmakros	361
9.2.5	Funktionsmakros mit Parametern	366
9.2.6	Makros entfernen: #undef	382
9.2.7	Funktionen per Makro umleiten	383
9.2.8	X-Makros: Listen einmal definieren, mehrfach verwenden	387
9.3	Bedingte Kompilierung	388
9.3.1	#if, #elif, #else und #endif	388
9.3.2	Der defined-Operator	390
9.3.3	#ifdef und #ifndef	391
9.3.4	#elifdef und #elifndef	392
9.4	Vordefinierte Makros	394
9.4.1	Datei- und Zeileninformationen	394
9.4.2	Compile-Datum und -Zeit	396
9.4.3	Standard- und Umgebungsinformationen	397
9.4.4	Unterstützung optionaler Sprach- und Bibliotheks- erweiterungen	398
9.5	Compilerabhängige Erweiterungen	399
9.5.1	Statement-Expressions	399
9.5.2	Compilerspezifische und plattformabhängige Makros	401
9.5.3	Beispiel: portabler Einsatz von Attributen mit bedingter Kompilation	402
9.5.4	Plattformspezifische Kompilierung	403
9.5.5	Unterscheidung zwischen C und C++	404
9.6	Rückblick und Ausblick	405
10	Benutzerdefinierte und erweiterte Datentypen	407
10.1	Typ-Alias definieren (typedef)	407
10.2	Ganzzahltypen aus <stdint.h>	409
10.2.1	Ganzzahlen mit fester Breite (intN_t und uintN_t)	410
10.2.2	Minimalbreiten-Ganzzahlen (int_leastN_t und uint_leastN_t)	415
10.2.3	Schnellste Ganzzahlen (int_fastN_t und uint_fastN_t)	417
10.2.4	Ganzzahltypen größter Breite (intmax_t und uintmax_t)	418

10.2.5	Portable Format-Makros für Ein- und Ausgabe	419
10.2.6	Zusammenfassung	423
10.3	Typen aus <stddef.h>	424
10.3.1	size_t	424
10.4	Optionale Gleitkomma-Datentypen	425
10.4.1	Warum es optionale Datentypen gibt	426
10.4.2	FloatN-Typen: feste IEEE-754-Formate für Portabilität	429
10.4.3	_FloatNx-Typen: erweiterte Formate für höhere Präzision	432
10.4.4	Dezimale Gleitkommazahlen (_Decimal32, _Decimal64, _Decimal128)	433
10.4.5	Fallback-Strategien	434
10.4.6	Gibt es Float-Aliase (<stdfloat.h>) in C23?	434
10.5	Größen, Grenzen und Eigenschaften der Typen	434
10.5.1	Messwerkzeuge zur Ermittlung von Typ-Eigenschaften	435
10.5.2	Grenzen für Ganzzahltypen	436
10.5.3	Portabel definierte Ganzzahltypen	437
10.6	Rückblick und Ausblick	438
11	Komplexe Datentypen	439
11.1	Aufzählungen (enum)	439
11.1.1	Definition von Aufzählungen	439
11.1.2	Variablen eines Aufzählungstyps deklarieren und verwenden	440
11.1.3	Aufzählungen in switch-case	441
11.1.4	Namensräume und Sichtbarkeit von Aufzählungskonstanten	441
11.1.5	Verwendung als Parameter und Rückgabewert	443
11.1.6	Automatische Wertezuweisung und Vergleich	444
11.1.7	Explizite Wertzuweisungen	446
11.1.8	Sentinel-Werte in Aufzählungen	448
11.1.9	Anonyme enum-Deklarationen	450
11.1.10	Aufzählungskonstanten als konstante Ausdrücke und frühere Nutzungsmuster	451
11.1.11	Implizite und feste Basistypen	452
11.1.12	Vorwärtsdeklaration von Aufzählungen	454
11.1.13	Vereinfachung von enum-Nutzung durch typedef	455
11.2	Strukturen (struct)	457
11.2.1	Deklaration von Strukturen	458
11.2.2	Anlegen von Variablen vom Typ einer Struktur	460
11.2.3	Zugriff auf Strukturelemente	463
11.2.4	Wertweise Kopiersemantik	463

11.2.5	Initialisierung von Strukturen	469
11.2.6	Anonyme Strukturen und verschachtelte Typen	474
11.2.7	Vereinfachung von Strukturnutzung durch typedef	476
11.2.8	Strukturtypen in verschiedenen Translation Units	482
11.2.9	Aufgabe	483
11.3	Unions (union)	484
11.3.1	Aufbau und Definition einer Union	485
11.3.2	Anlegen von Variablen vom Typ einer Union	486
11.3.3	Zugriff und Übergabe an Funktionen	486
11.3.4	Anonyme Union	488
11.3.5	Union mit typedef	488
11.3.6	Variantentypen	489
11.3.7	Strukturen und Unions kombinieren	492
11.3.8	Tagged Unions	496
11.3.9	Variantentypen für Rückgabewerte (Result-Muster)	499
11.3.10	Type Punning mit Strukturen und Unions	502
11.3.11	Anonyme Unions und Kombination mit Strukturen	506
11.3.12	Aufgabe	510
11.4	Rückblick und Ausblick	511
12	Arrays	513
12.1	Einführung und Motivation für Arrays	513
12.2	Deklaration von Arrays	513
12.2.1	Arrays mit fester Größe (Compile-Time-Größe)	514
12.2.2	Arrays mit variabler Größe (Variable-Length Arrays, VLAs)	516
12.3	Zugriff auf Array-Elemente	519
12.3.1	Indexnotation	519
12.3.2	Über Arrays iterieren	520
12.3.3	Out-of-Bounds-Zugriffe und Undefined Behavior	521
12.3.4	Sichere Praktiken und Debugging	522
12.3.5	Grenzchecks mit AddressSanitizer	522
12.4	Initialisierung von Arrays	523
12.4.1	Listeninitialisierung	524
12.4.2	Teilinitialisierung und implizite Nullen	529
12.4.3	Standardinitialisierung mit {0}	530
12.4.4	Zeichenketten als char-Arrays	532
12.4.5	Designated Initializers	534
12.4.6	const Arrays	537

12.5	Arrays bei der Parameterübergabe und Rückgabe von Funktionen	540
12.5.1	Arrays an Funktionen übergeben	540
12.5.2	Variable Length Arrays in Funktionsparametern	541
12.5.3	Grenzen von sizeof bei Funktionsparametern	542
12.5.4	Mit const qualifizierte Parameter	543
12.5.5	Minimale Größengarantie mit static	543
12.5.6	Arrays als Rückgabewert: Einschränkungen und Alternativen	547
12.6	Aufgaben	550
12.7	Mehrdimensionale Arrays	554
12.7.1	Deklaration mehrdimensionaler Arrays	555
12.7.2	Variable Length Arrays (VLA) mit mehreren Dimensionen	555
12.7.3	Zugriff mit mehreren Indizes	555
12.7.4	Mehrdimensionale Arrays sind Arrays von Arrays	556
12.7.5	Mehrdimensionale Arrays liegen im Speicher hintereinander	557
12.7.6	sizeof bei mehrdimensionalen Arrays	558
12.7.7	Initialisierung von mehrdimensionalen Arrays	561
12.7.8	Mehrdimensionale Arrays an Funktionen übergeben	566
12.7.9	Praktische Beispiele mit mehrdimensionalen Arrays: Tabellen, Brettspiele und mehr	569
12.8	Einbinden externer Inhalte	576
12.9	Rückblick und Ausblick	578
13	Zeiger	579
13.1	Datenzeiger und Adressen	579
13.1.1	Speicher und Variablen	579
13.1.2	Adressoperator (&)	582
13.1.3	Zeigervariablen: Adressen speichern	583
13.1.4	Weitere zulässige Anwendungen des Adressoperators	585
13.1.5	Dereferenzierungsoperator (*)	586
13.1.6	Beispiel: Menüsystem mit Umschalten per Zeiger	588
13.1.7	Deklaration mehrerer Zeigervariablen	589
13.1.8	Typumwandlungen zwischen Zeigertypen	591
13.1.9	Das Schlüsselwort volatile	593
13.1.10	Zugriff auf Speicherstellen: Memory-Mapped I/O	593
13.1.11	Zeiger als Ganzzahl interpretieren sowie der Typ uintptr_t	596
13.1.12	Pointer Tagging	597
13.1.13	Aufgabe	599
13.2	Der Null-Pointer nullptr	599
13.2.1	Dereferenzierung einer Null-Pointer-Variablen	600

13.2.2	Zeigervariablen auf nullptr prüfen	600
13.2.3	Typ nullptr_t	601
13.2.4	Das Makro NULL	602
13.3	Zeiger und const: Unveränderlichkeit deklarieren	602
13.3.1	Zeiger auf konstante Daten	603
13.3.2	Konstanter Zeiger	605
13.3.3	Konstanter Zeiger auf konstante Daten	605
13.3.4	Zusammenfassung und Merkregel: const und Zeiger	606
13.4	Zeiger in Funktionen	607
13.4.1	Wertübergabe und Adressübergabe	607
13.4.2	Beispiel: swap()-Funktion mit Zeigern	608
13.4.3	Mehrere Rückgabewerte über Out-Parameter	609
13.4.4	Terminologie: In-, Out- und In-Out-Parameter	610
13.4.5	Ergebnisse über Out-Parameter und Status über Rückgabewerte	610
13.4.6	Grund für Zeigerübergabe bei scanf()	611
13.4.7	Zeiger als Rückgabewert	612
13.4.8	Häufige Fehler: Dangling-Pointers	614
13.4.9	Zeiger auf Funktionsparameter zurückgeben	615
13.4.10	Der Qualifizierer restrict: Aliasfreiheit und Optimierung	616
13.4.11	Exkurs: Referenzen in C++	618
13.4.12	Best Practices für Zeigerparameter	619
13.5	Void-Zeiger: typenloses Zeigerwerkzeug	620
13.5.1	Die Idee des void-Zeigers	621
13.5.2	Implizite und explizite Konvertierung	622
13.5.3	Hexadezimale Ausgabe beliebiger Datentypen	623
13.5.4	Einschränkungen von void *	625
13.6	Zeigerbeziehungen und -operationen	626
13.6.1	Vergleichsarten: Gleichheit und Ordnung	627
13.6.2	Gleichheitsvergleich: identische Adressen	627
13.6.3	Vergleich über unterschiedliche Zeigertypen und void *	629
13.6.4	Ordnungsvergleiche: relative Position im Speicher	630
13.6.5	Zeigerdifferenzen und ptrdiff_t	632
13.6.6	Zeigerprüfungen in Bedingungen	632
13.7	Array-Decay und Zeigerarithmetik	634
13.7.1	Array-Decay: implizite Umwandlung	634
13.7.2	Unterschiede zwischen Array-Namen und echten Zeigern	636
13.7.3	Zeigerarithmetik: Rechnen mit Adressen	637
13.7.4	Index-Notation vs. Zeiger-Notation	639
13.7.5	Iteration mit Zeigern	641

13.7.6	Array-Größe in Funktionen und der Effekt des Array-Decay	645
13.7.7	Mehrdimensionale Arrays und Zeiger	646
13.7.8	Aufgabe	652
13.8	Zugriff auf Strukturen über Zeiger	652
13.8.1	Der Pfeiloperator (->)	654
13.8.2	Funktionen mit Strukturzeigern	656
13.8.3	Verschachtelte Strukturen und mehrfache Pfeile	657
13.8.4	Zeiger auf Strukturen und verkettete Listen	658
13.8.5	Doppelt verkettete Listen	660
13.8.6	Aufgabe	661
13.9	Arrays von Zeigern	661
13.9.1	Deklaration und Initialisierung	662
13.9.2	Arrays von Zeichenketten	664
13.9.3	Verwendung mit Null-Pointer als Sentinel	666
13.9.4	Beispiel: Nachrichtenverwaltung mit Arrays von Zeigern auf Strukturen	668
13.9.5	Beispiel: Verlaufsliste mit Array von Zeigern	669
13.10	Mehrstufige Indirektion mit Zeigern auf Zeigern	671
13.10.1	Zeiger auf Zeiger deklarieren	671
13.10.2	Konstantheit bei mehrstufiger Indirektion	673
13.10.3	Warum mehrstufige Indirektion?	674
13.10.4	Unterschied: Array von Zeigern versus Zeiger auf Zeiger	675
13.10.5	Funktionen mit Arrays von Zeigern	676
13.10.6	Darstellung von Matrizen: zweidimensionale Arrays und Zeiger auf Zeiger	677
13.10.7	Zeiger in Funktionen verändern	679
13.10.8	Beispiel: verkettete Listen und Zeiger auf Zeiger	681
13.10.9	Mehrdimensionale Zeiger-Strukturen	683
13.11	Kommandozeilenargumente als Array von Zeigern	685
13.11.1	Programmargumente auswerten	688
13.12	Funktionszeiger	688
13.12.1	Deklaration und Syntax	690
13.12.2	Adresse einer Funktion speichern	692
13.12.3	Aufruf über Funktionszeiger	693
13.12.4	Vereinfachte Syntax durch Function-to-Pointer-Decay	695
13.12.5	Typedef für bessere Lesbarkeit	696
13.12.6	Namensräume durch Strukturen	697
13.12.7	Funktion höherer Ordnung	702
13.12.8	Arrays von Funktionszeigern für Sprungtabellen	707

13.12.9	Vergleichsfunktionen	709
13.12.10	Suchen und Sortieren über die Standardbibliothek	714
13.13	Rückblick und Ausblick	718
14	Speicherverwaltung und Fehlersicherheit	721
14.1	Einführung in die dynamische Speicherverwaltung	721
14.1.1	Stack vs. Heap: Eigenschaften, Lebensdauer und Anwendungsfälle	722
14.2	Grundlegende Allokationsfunktionen und Anwendungen	723
14.2.1	Speicher allozieren (malloc())	724
14.2.2	Freigabe von Speicher (free())	725
14.2.3	Speicher für Arrays allozieren	728
14.2.4	Speicher für Strukturen allozieren	729
14.2.5	Opake Typen: Strukturen hinter einer Schnittstelle verbergen	732
14.2.6	Speicher für Arrays von Strukturen allozieren	735
14.2.7	Dynamisch verkettete Listen implementieren	736
14.2.8	Allokation mit Initialisierung auf null (calloc())	748
14.2.9	Erweiterung und Verkleinerung von Speicher- bereichen (realloc())	750
14.2.10	Sichere Größenberechnung	756
14.3	Arbeiten mit Speicherbereichen	757
14.3.1	Speicher kopieren	758
14.3.2	Speicher vergleichen	760
14.3.3	Strukturen vergleichen	761
14.3.4	Speicher setzen	763
14.3.5	Einzelne Bytes suchen (memchr())	766
14.3.6	Aufgabe	768
14.4	Flexible Array-Mitglieder (FAM)	771
14.4.1	Syntax und Verwendung flexibler Array-Mitglieder	772
14.4.2	Fazit und Zusammenfassung	773
14.5	Speicherlebenszyklen und Ownership-Modelle	773
14.5.1	Ownership in C: Verantwortlichkeiten und Konventionen	774
14.5.2	Referenzzählung	776
14.5.3	Scope-basierte Ressourcenverwaltung mit Cleanup-Attributen	779
14.6	Typische Speicherfehler und ihre Vermeidung	781
14.6.1	Null-Pointer-Checks	781
14.6.2	Buffer-Overflows und Underflows bei dynamischem Speicher	786
14.6.3	Memory Leaks (Speicherlecks)	789
14.6.4	Dangling-Pointers und Use-after-free	796

14.6.5	Double-free und Invalid-free	800
14.6.6	Übersicht und Zusammenfassung über Undefined Behavior im Speichermanagement	803
14.7	Fragmentierung und Out-of-Memory-Handling	804
14.8	Alignment und Padding	804
14.9	Debugging und Analyse von Speicherfehlern	807
14.9.1	AddressSanitizer	808
14.9.2	MemorySanitizer (MSan)	810
14.9.3	Vertrauen gut, Valgrind ist besser	811
14.9.4	Core Dumps und Post-Mortem-Debugging	813
14.9.5	Wann welches Tool einsetzen?	813
14.9.6	Weitere Tools	814
14.9.7	Entwicklung und heutiger Stand	815
14.10	Allokatoren	816
14.10.1	Wie malloc() sich Speicher holt	816
14.10.2	Implementierungen von malloc()/free() in modernen Systemen	818
14.10.3	Eigene Allokatoren schreiben	820
14.10.4	Custom Allocators parametrisieren	826
14.10.5	Laufzeitüberlegungen: Heap vermeiden (Stack-Allokatoren, VLAs)	827
14.11	Rückblick und Ausblick	828
15	Metaprogrammierung	831
15.1	Das Problem: generischer Code ohne zusätzlichen Laufzeitaufwand	831
15.1.1	Was ist Metaprogrammierung?	832
15.1.2	Die drei Phasen der Kompilierung	833
15.2	Präprozessor-Metaprogrammierung	833
15.2.1	Portable Konfiguration und Plattformerkenung	834
15.2.2	Feature-Erkennung	835
15.2.3	Standard-Makros und Metainformationen	836
15.3	Compilerbasierte Metaprogrammierung	837
15.3.1	Typprüfungen mit static_assert	837
15.3.2	Generische Makros mit typeof und typeof_unqual	841
15.3.3	Das container_of-Idiom	845
15.3.4	Typabhängige Auswahl mit _Generic	848
15.3.5	Optimierungshinweis unreachable()	860
15.4	Zusammenfassung und Ausblick	862

16	Zeichen, Zeichenketten und Textverarbeitung	865
16.1	Historische Grundlagen der Textrepräsentation	865
16.1.1	char, signed char, unsigned char	865
16.1.2	Codepages und das klassische C-Zeichenmodell	867
16.2	Von einzelnen Zeichen zu Zeichenketten	867
16.2.1	Null-Terminierung von Strings	867
16.2.2	Klassische String-Literale	869
16.2.3	Die Transformation von String-Literalen in C	869
16.2.4	Zeiger vs. Array	871
16.2.5	Speicherorte für nicht-literale Strings	873
16.2.6	Umgang mit konstanten Strings	873
16.3	Moderne Textrepräsentation	874
16.3.1	Unicode als abstraktes Zeichensystem	874
16.3.2	Codierungen	874
16.3.3	Anzeigeebene: wie Zeichen erscheinen	876
16.4	Moderne Texttypen und Literale	877
16.4.1	char32_t	877
16.4.2	char8_t	878
16.4.3	char16_t	878
16.4.4	Historischer Typ wchar_t	879
16.4.5	Fazit	879
16.4.6	Unicode-Präfixe für String-Literale und Zeichenkonstanten	879
16.4.7	Universal Character Names (UCN)	882
16.5	Moderne String-Repräsentationen	884
16.5.1	String mit explizit gespeicherter Länge	884
16.5.2	Small-String-Konzepte	886
16.5.3	String-View	887
16.6	Memory-Management-Techniken	889
16.6.1	String-Pools	889
16.6.2	Globales Interning	891
16.6.3	Flyweight-Pattern	892
16.7	Dynamische Strings	893
16.7.1	String mit Kapazität	893
16.7.2	Wachstumsstrategien und Kostenmodell	897
16.7.3	Arena-basierte dynamische Strings	897
16.7.4	Vergleich mit *sprintf()-Funktionen	898
16.8	Locale: Einfluss auf Zeichen- und String-Funktionen	898
16.8.1	Locale-Kategorien	899
16.8.2	Locale setzen	900

16.8.3	Probleme des globalen Locale-Modells	900
16.8.4	Formatierungsdaten abfragen	901
16.8.5	Praxis: Locale sparsam einsetzen	901
16.9	Formatierte Ausgabe in Strings	902
16.9.1	Formatfunktionen [v]snprintf()/s[n]printf_s()	902
16.9.2	sprintf() und vsprintf()	904
16.9.3	String-Konkatenation mit Formatfunktionen	905
16.9.4	Automatische Speicherallokation mit GNU-Funktionen asprintf() und vasprintf()	907
16.9.5	Format-Spezifizierer	908
16.9.6	Einzelne Werte in Strings konvertieren	910
16.9.7	Compilerprüfungen bei Formatstrings	910
16.9.8	Formatstring-Prüfung für eigene Funktionen in GCC/Clang	913
16.9.9	Formatstring-Schwachstellen	914
16.9.10	Sichere Nutzung der printf()-Familie	915
16.10	Parsing und Konvertierung	916
16.10.1	Tokenisierung: Text in Einheiten zerlegen	916
16.10.2	Tokenisierung mit Standardfunktionen (strtok*())	916
16.10.3	Parser selbst entwickeln	918
16.10.4	Zeichenketten in Zahlen umwandeln (strto*())	918
16.10.5	Verkürzte Varianten atoi(), atol(), atoll() vermeiden	922
16.10.6	Strukturierte Eingaben auslesen (sscanf*())	923
16.11	Klassische Charakterklassifikation (is*()) und weitere)	926
16.12	Unicode in C23	927
16.12.1	UTF-8 selbst verarbeiten	928
16.12.2	Die Scheinlösung: Wide Characters	930
16.12.3	Externe Bibliotheken	930
16.13	Standard-Stringfunktionen für nullterminierte Zeichenketten	931
16.13.1	Die zwei grundlegenden Schwächen der klassischen C-Strings	932
16.13.2	Nur lesende Stringfunktionen	933
16.13.3	Schreibende Funktionen ohne Längenparameter	935
16.13.4	Schreibende Funktionen mit Längenparameter	935
16.13.5	Dynamische Stringfunktionen	937
16.13.6	Eigene Wrapper mit __builtin_object_size()	937
16.13.7	Beispiel: Dateiname und Dateiendung aus einem Pfad extrahieren	938
16.13.8	mem*()-Funktionen für String-Verarbeitung nutzen	940
16.13.9	Stringfunktionen mit verschiedenen String-Literal-Typen?	942
16.14	Vergleiche und Hashing	944
16.14.1	Grundformen des Stringvergleichs	944

16.14.2	Case-insensitive Vergleiche	945
16.14.3	Locale-abhängige Vergleiche	945
16.14.4	Natürliche Sortierung	947
16.14.5	Hashfunktionen für schnelle String-Vergleiche	948
16.15	Testing, Fuzzing und Verifikation von C-String-Code	951
16.15.1	Stringbezogene Fehler erkennen	951
16.15.2	Speicherbezogene Fehler erkennen	951
16.16	Aufgabe	951
16.17	Rückblick und Ausblick	956
17	Ein-/Ausgabe (I/O)	957
17.1	Grundlagen von Streams	957
17.2	Dateien öffnen und schließen	958
17.2.1	Eine Datei öffnen und schließen (fopen() und fclose())	958
17.3	Textdateien schreiben und lesen	960
17.3.1	Text schreiben (fprintf(), fputs(), fputc() usw.)	960
17.3.2	Text einlesen (fscanf(), fgets(), fgetc() usw.)	961
17.4	Standardstreams	962
17.4.1	Kurzformen für Ein- und Ausgabe	963
17.4.2	Einen Stream neu zuordnen (freopen())	963
17.5	Rohdaten verarbeiten	964
17.5.1	Strukturen binär schreiben und lesen	967
17.5.2	Aufgabe	970
17.6	Position im Stream steuern (fseek(), ftell() usw.)	972
17.7	Fehler erkennen und behandeln (ferror(), feof(), clearerr())	975
17.8	Puffer und Streameigenschaften steuern	976
17.8.1	Puffer einstellen (setvbuf())	976
17.8.2	Puffer leeren (fflush())	978
17.9	Dateien löschen und umbenennen	979
17.9.1	Eine Datei löschen (remove())	979
17.9.2	Eine Datei umbenennen (rename())	979
17.10	Temporäre Dateien	980
17.10.1	Eine temporäre Datei öffnen (tmpfile())	980
17.10.2	Sicheren temporären Dateinamen erzeugen (mkstemp())	981
17.11	Plattform- und POSIX-Erweiterungen	982
17.11.1	Umwandlung zwischen Deskriptoren und FILE *-Streams	982

17.12 POSIX-Erweiterungen rund um Datei- und Verzeichnis-I/O	983
17.12.1 Low-Level-I/O mit Dateideskriptoren	983
17.12.2 Dateisystem-Metadaten und Dateitypen	983
17.12.3 Rechte, Besitzer und Erstellungsmaske	984
17.12.4 Verzeichnisse lesen und verwalten	984
17.12.5 Links, Umbenennen und Löschen	985
17.12.6 Dateigröße ändern und Synchronisation	985
17.12.7 Dateideskriptor-Steuerung und Flags	985
17.12.8 Multiplexing und nicht-blockierendes I/O	986
17.12.9 Speicherabbildung von Dateien	986
17.12.10 Speicherbasierte Streams	986
17.12.11 Zeitstempel von Dateien setzen	986
17.13 Zusammenfassung und Ausblick	987
 18 Bitoperationen	 989
18.1 Bit- und Schiebeoperatoren	990
18.1.1 Bitweises UND (&)	991
18.1.2 Bitweises ODER ()	991
18.1.3 Bitweises XOR (^)	992
18.1.4 Bitweises NICHT (~)	992
18.1.5 Schiebeoperatoren (<<, >>)	993
18.1.6 Häufige Fehler bei bitweisen Operatoren	994
18.1.7 Aufgaben	995
18.2 Bitmasken, Flags und typische Muster	996
18.2.1 Bitmasken und ihre Konstruktion	996
18.2.2 Setzen einzelner Bits	997
18.2.3 Löschen einzelner Bits	998
18.2.4 Umschalten einzelner Bits	999
18.2.5 Prüfen einzelner Bits	999
18.2.6 Mehrere Bits gleichzeitig prüfen	1000
18.2.7 Aufzählungskonstanten als Bitmasken	1001
18.2.8 Häufige Fehler und Best Practices	1004
18.2.9 Zusammenfassung des Musters	1005
18.3 Standardisierte Bitfunktionen (<stdbit.h>)	1006
18.3.1 Übersicht der Funktionen	1006
18.4 Ganzzahlen mit fester Bitbreite (_BitInt)	1008
18.4.1 Beispiel und Überlaufverhalten	1008
18.4.2 printf() und Typecast	1010
18.4.3 _BitInt in Strukturen	1011

18.5	Bitfelder	1012
18.5.1	Bitfelder definieren	1012
18.5.2	Einfaches Flags-Beispiel	1013
18.5.3	IPv4-Beispiel	1015
18.5.4	Abbildung und Padding von Bitfeldern	1017
18.6	Rückblick und Ausblick	1019
19	Numerik und Mathematik	1021
19.1	Sichere Arithmetik (<stdckdint.h>)	1021
19.1.1	Wertebereiche und Überlaufprobleme	1021
19.1.2	Strategien zur Überlauferkennung	1022
19.1.3	Checked Integer Arithmetic	1023
19.1.4	Überlauf bei Gleitkommazahlen	1025
19.2	Besondere Gleitkommazahlen	1025
19.2.1	NaN	1026
19.2.2	Vorzeichenbehaftete Null (+0.0 und -0.0)	1026
19.2.3	Unendlichkeiten und Division durch null	1027
19.2.4	Subnormale Zahlen	1028
19.2.5	Klassifikation von Gleitkommazahlen	1028
19.2.6	Grenzen und Epsilon	1029
19.3	Konvertierungen zwischen Ganz- und Gleitkommazahlen	1032
19.3.1	Cast von Ganzzahlen in Gleitkommazahlen	1033
19.3.2	Casts von Gleitkommazahlen in Ganzzahltypen	1033
19.3.3	Rundung beim Cast von Gleitkommazahlen in Ganzzahlen	1034
19.4	Mathematische Funktionen (<math.h>)	1036
19.4.1	Demonstration ausgewählter Funktionen	1036
19.4.2	Übersicht der wichtigsten Funktionen	1037
19.4.3	Linken der Mathematik-Bibliothek (-lm)	1039
19.4.4	Aufgabe	1040
19.5	Floating-Point-Umgebung und Rundungsmodi (<fenv.h>)	1040
19.5.1	IEEE-754- und IEC-60559-Konformität in C23	1040
19.5.2	fesetround() und ihre Auswirkungen	1041
19.5.3	Floating-Point-Exceptions und Status-Flags	1044
19.5.4	Steuerung der FP-Umgebung in der Praxis	1045
19.6	Generische Mathematik (<tgmath.h>)	1046
19.6.1	Beispielnutzung mit Sinus	1046
19.6.2	Übersicht der typgenerischen Makros	1047

19.7	Zufallsfunktionen in C	1047
19.8	Komplexe Zahlen	1049
19.8.1	Datentypen float complex, double complex und long double complex	1049
19.8.2	Operatoren	1050
19.8.3	Funktionen	1050
19.8.4	Beispiel: Mandelbrot-Menge	1050
19.9	Rückblick und Ausblick	1051
20	Weitere Teile der C-Standardbibliothek	1053
20.1	Standard Utilities Header (<stdlib.h>)	1053
20.1.1	Überblick: bereits bekannte Teile	1053
20.1.2	Programm beenden	1054
20.1.3	Umgebungsvariablen lesen (getenv())	1055
20.1.4	POSIX-Erweiterungen für Umgebungsvariablen	1056
20.1.5	Externe Befehle ausführen (system())	1056
20.2	Signale (<signal.h>)	1058
20.3	Nichtlokale Sprünge (<setjmp.h>)	1059
20.3.1	Kontext speichern (jmp_buf)	1059
20.3.2	Rücksprungpunkt setzen (setjmp())	1059
20.3.3	Sprung ausführen (longjmp())	1060
20.3.4	Regeln und Fallstricke	1062
20.3.5	Try/Catch per Makros (setjmp()/longjmp())	1062
20.4	Zeit und Datum (<time.h>)	1067
20.4.1	Aktuelle Zeit abfragen: time_t und time()	1067
20.4.2	Zeitwerte in Kalenderzeit umwandeln	1069
20.4.3	Kalenderzeit formatieren: strftime()	1071
20.4.4	Kalenderdaten zurückwandeln: mktime() und timegm()	1074
20.4.5	Zeitspannen berechnen: difftime()	1076
20.4.6	Genauere Wandzeit: struct timespec und timespec_get()	1077
20.4.7	Auflösung abfragen: timespec_getres()	1079
20.4.8	POSIX am Rand: clock_gettime()	1079
20.4.9	CPU-Zeit messen: clock()	1080
20.4.10	Zeitzonen und Sommerzeit: tzset() als POSIX-Erweiterung	1081
20.5	Rückblick und Ausblick	1082

21 Nebenläufigkeit und Parallelität	1083
21.1 Einführung in Nebenläufigkeit	1083
21.1.1 Warum Nebenläufigkeit? Herausforderungen und Anwendungsfälle	1083
21.1.2 CPU-bound vs. I/O-bound Tasks	1085
21.1.3 Amdahl's Law und die Grenzen der Parallelisierung	1086
21.1.4 Threads vs. Prozesse	1088
21.2 Thread-Lebenszyklus: Erstellen, Ausführen, Beenden	1090
21.2.1 Thread-Bibliothek <threads.h>	1090
21.2.2 Erstellung eines Threads	1091
21.2.3 Freiwillig Rechenzeit abgeben	1094
21.2.4 Wer bin ich? Aktuellen Thread ermitteln	1095
21.2.5 Threads schlafen legen	1096
21.2.6 Natürliches und unnatürliches Ende eines Threads	1097
21.2.7 Der Hauptthread und Programmende	1101
21.2.8 Detached Threads	1104
21.2.9 Kooperatives Beenden von Threads durch Signalisierung	1106
21.2.10 Best Practices für den Thread-Lebenszyklus	1106
21.3 Das C-Memory-Modell und atomare Operationen	1106
21.3.1 Grundidee: Synchronisation schafft Ordnung	1107
21.3.2 Atomare und nicht-atomare Speicherzugriffe	1107
21.3.3 Data Race	1108
21.3.4 Ordnung und Korrektheit von Speicherzugriffen im C-Memory-Model	1110
21.4 Fortgeschrittene Konzepte kurz vorgestellt	1111
21.4.1 Atomare Typen und grundlegende Operationen (<stdatomic.h>) ...	1111
21.4.2 Synchronisation mit Mutex	1111
21.4.3 Condition Variables (cnd_t)	1111
21.4.4 Thread-lokaler Speicher (thread_local)	1112
21.5 Alternativen zu <threads.h>	1112
21.5.1 pthreads als faktischer Referenzpunkt	1112
21.5.2 Ereignisschleifen statt Threads	1115
21.5.3 Konsequenzen für den Entwurf	1116
21.6 Rückblick und Ausblick	1116

22	Standardbibliothek, Implementierungen und Erweiterungen	1119
22.1	Implementierungen der Standardbibliothek	1119
22.1.1	Systemabhängige Implementierungen der libc	1120
22.1.2	Schlanke Implementierungen für eingebettete Systeme	1120
22.2	POSIX	1121
22.2.1	POSIX-Funktionsbereiche	1122
22.2.2	POSIX-Header	1124
22.2.3	POSIX aktivieren	1126
22.3	Plattformunabhängige Bibliotheken	1127
22.4	Plattformabstrahierende Bibliotheken	1128
22.4.1	Konkrete Beispiele plattformabstrahierender Bibliotheken	1129
22.5	Plattformspezifische APIs jenseits von C und POSIX	1130
22.5.1	Typische Funktionsbereiche plattformspezifischer APIs	1131
22.5.2	Header, Bibliotheken und ABI	1131
22.5.3	Konsequenzen für C-Programme	1131
22.6	Compiler- und Implementierungserweiterungen	1132
22.6.1	Typische Erweiterungen	1132
22.7	Zusammenfassung und Ausblick	1133
23	Build und Auslieferung	1135
23.1	Make	1136
23.1.1	Aufbau eines Makefiles	1136
23.1.2	Erstes Beispiel	1136
23.1.3	Ziele über die Kommandozeile ansprechen	1137
23.1.4	Abhängigkeiten	1137
23.1.5	Variablen und automatische Variablen	1138
23.1.6	Variablen beim Aufruf überschreiben	1140
23.1.7	Ausblick: pkg-config	1141
23.1.8	Ausblick: Autotools	1141
23.2	Projektstruktur in C	1144
23.2.1	Makefile unter Berücksichtigung der Verzeichnisstruktur	1145
23.3	CMake	1147
23.3.1	Ein erstes Beispiel	1148
23.4	Cross-Compiler	1149

23.5	Packaging und Distribution	1151
23.5.1	Statisches vs. dynamisches Linken	1151
23.5.2	Verteilungsformen	1151
23.5.3	Versionierung	1152
23.6	Continuous Integration	1152
23.7	Zusammenfassung und Ausblick	1154
24	Testen und Qualitätssicherung	1155
24.1	Compiler-Warnungen	1155
24.2	Statische Analyse	1156
24.3	Code-Formatter	1158
24.3.1	clang-format	1158
24.4	Tests	1159
24.4.1	Unity	1159
24.4.2	Regressionstests	1161
24.5	Code-Coverage	1161
24.6	Fuzzing	1162
24.7	Review-Prozesse	1163
24.8	Dokumentation	1164
24.8.1	Doxygen	1164
24.8.2	Was nicht dokumentiert wird	1165
24.9	Zusammenfassung und Ausblick	1166
25	Profiler und Optimierung	1167
25.1	Compiler-Optimierungen	1167
25.1.1	Optimierungsstufen: Steuerung durch Compiler-Schalter	1168
25.1.2	Optimierungen nach Reichweite	1169
25.1.3	Speicher und Aliasing als Optimierungsgrenze	1170
25.1.4	Fazit: Was bedeutet das für die Praxis?	1171
25.2	Grundlagen und Kennzahlen der Leistungsanalyse	1172
25.2.1	Laufzeit, Durchsatz, Latenz	1173
25.2.2	Messfehler, Störeinflüsse	1173
25.2.3	Warmup, Cache, Branch Prediction	1174
25.2.4	Messbare Kennzahlen und Messmethodik	1175
25.3	Werkzeuge	1176
25.3.1	Profiler	1176

25.3.2	Tracer	1177
25.3.3	Counter-basierte Werkzeuge	1178
25.3.4	Einordnung und Arbeitsweise	1179
25.4	Benchmarking und Reproduzierbarkeit	1180
25.4.1	Benchmarking als Experiment	1180
25.4.2	Micro- und Macro-Benchmarks als Messlinse	1180
25.4.3	Reproduzierbarkeit als Protokollfrage	1181
25.4.4	Vergleichbarkeit und Versuchsbedingungen	1181
25.4.5	Messreihen und Driftkontrolle	1182
25.4.6	Typische Benchmark-Artefakte	1182
25.4.7	Benchmark-Frameworks	1183
25.5	Optimierungstechniken im Code	1184
25.5.1	Algorithmische Optimierung	1184
25.5.2	Datenlayout und Cache-Lokalität	1185
25.5.3	Parallelisierung und SIMD	1187
25.5.4	Kontrollfluss und Mikrooptimierungen	1189
25.5.5	Die goldene Reihenfolge	1191
25.6	Methodik: systematisches Vorgehen bei Leistungsproblemen	1191
25.7	Rückblick und Ausblick	1193
26	Nachwort: der »C-Weg« – Philosophie und Kernprinzipien	1195
Anhang		1199
A	Standard-Header	1201
B	Die 8 kostbaren Übersetzungsphasen	1205
C	Operatorpräzedenz	1211
D	Konvertierungsspezifikationen sowie PRI- und SCN-Makros	1213
Index		1229