Herold/Arndt: *C-Programmierung unter Linux/UNIX/Windows*

Dr. Helmut Herold
Jörg Arndt

C-Programmierung unter Linux / UNIX / Windows

Beispiele, Anwendungen, Programmiertechniken

Alle in diesem Buch enthaltenen Programme, Darstellungen und Informationen wurden nach bestem Wissen erstellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund ist das in dem vorliegenden Buch enthaltene Programm-Material mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autoren und die SUSE LINUX AG übernehmen infolgedessen keine Verantwortung und werden keine daraus folgende Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programm-Materials, oder Teilen davon, oder durch Rechtsverletzungen Dritter entsteht.
Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann verwendet werden dürften.
Alle Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt und sind möglicherweise eingetragene Warenzeichen. Die SUSE LINUX AG richtet sich im Wesentlichen nach den Schreibweisen der Hersteller. Andere hier genannte Produkte können Warenzeichen des jeweiligen Herstellers sein.
Dieses Werk ist urheberrechtlich geschützt.
Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Druck, Fotokopie, Microfilm oder einem anderen Verfahren), auch nicht für Zwecke der Unterrichtsgestaltung, reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Bibliografische Information Der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über `http://dnb.ddb.de` abrufbar.
ISBN 3-89990-123-1

© 2004 SuSE Linux AG, Nürnberg (http://www.suse.de)
Umschlaggestaltung: Fritz Design GmbH, Erlangen
Gesamtlektorat: Nicolaus Millin
Fachlektorat: Dieter Bloms, Jörg Dippel, Klaas Freitag, Bruno Gerz, Bernhard Hoelcker, Björn Jacke, Andreas Jaeger, Dirk Pankonin, Wolfgang Rosenauer, Christian Steinrücken, Peter Varkoly
Satz: LaTeX
Druck: Kösel, Kempten
Printed in Germany on acid free paper.

Inhaltsverzeichnis

0 Einleitung **3**
0.1 Ein paar Worte zu C und seiner Geschichte 3
0.1.1 Die Entstehung von C 3
0.1.2 C als Vorstufe zu C++ 3
0.1.3 Die Standardisierung von C 4
0.2 Hinweise zu diesem Buch und zum begleitenden Übungs- bzw. Lösungsbuch 5
0.3 Downloads zu diesem Buch 6

1 Einführendes Beispiel **7**
1.1 Erste wesentliche C-Regeln 7
1.2 Erstellen eines Programms (Bedienung) 8
1.3 Kompilieren eines Programms (Bedienung) 9
1.4 Starten eines kompilierten Programms (Bedienung) . . 10
1.5 Das Steuerzeichen `\n` 11
1.6 Zeilen-Kommentare mit `//` (neu in C99) 12
1.7 Tipps 12
1.8 Übungen 14
1.8.1 Ausgabe eines Menüs 14
1.8.2 Es weihnachtet sehr 14
1.8.3 Kommentar und Anführungszeichen 14

2 Elementare Datentypen **15**
2.1 Positionssysteme 15
2.2 Negation von Zahlen durch Zweier-Komplement . . 16
2.3 Die Grunddatentypen in C 19
2.4 Wertebereiche für die einzelnen Datentypen . . . 21
2.5 Fallgrube: Verlust von Bits bei zu grossen Zahlen . . 22
2.6 Übungen 23
2.6.1 Umwandlung von Dualzahlen in Dezimalzahlen . 23
2.6.2 Umwandlung von Dezimalzahlen in Dualzahlen . 23
2.6.3 Bereichsüberläufe beim Datentyp `short` . . . 23

3 Konstanten **25**

3.1 Oktale und Hexadezimale Zahlen 25
3.2 Verschiedene Arten von C-Konstanten 26
3.2.1 char-Konstanten 26
3.2.2 Ganzzahlige Konstanten 27
3.2.3 Gleitpunktkonstanten 27
3.3 Übungen 28
3.3.1 Bitmuster von Zeichen und Zahlen beim Datentyp `char` 28
3.3.2 Bitmuster für Oktal- und Hexazahlen beim Datentyp `short` 28
3.3.3 Erlaubte und unerlaubte Gleitpunktkonstanten . 28

4 Variablen **29**
4.1 Variablen und die C-Regeln für Variablennamen . . . 29
4.2 Tipps zur Wahl der Variablennamen 31
4.2.1 Selbsterklärende Variablennamen 31
4.2.2 Unterstrich verbessert die Lesbarkeit 32
4.3 Deklaration von Variablen 32
4.4 Tipp: Variablen bereits bei Deklaration dokumentieren . 34
4.5 Übung: Erlaubte und unerlaubte Variablennamen . . 34

5 Ausdrücke und Operatoren **35**
5.1 Der einfache Zuweisungsoperator 35
5.1.1 Allgemeines zum einfachen Zuweisungsoperator . 35
5.1.2 Initialisierung von Variablen 38
5.1.3 Übung: Zuweisen von unterschiedlichen Konstanten 38
5.2 Arithmetische Operatoren 39
5.2.1 Die arithmetischen Operatoren 39
5.2.2 Die C-Begriffe Ausdruck und Anweisung . . . 40
5.2.3 Ausgabe von `int`-Variablen und -Ausdrücken . . 40
5.2.4 Ausgabe von Gleitpunkt-Variablen und -Ausdrücken 41
5.2.5 Fallgrube: Ganzzahl- statt Gleitpunktdivision . . 42
5.2.6 Übungen 43
5.3 Vergleichsoperatoren 44
5.3.1 Die unterschiedlichen Vergleichsoperatoren . . 44
5.3.2 Die zwei Wahrheitswerte von Vergleichen . . . 44
5.3.3 Prioritäten der Vergleichsoperatoren 45
5.3.4 Übung: Prioritäten aller bisherigen Operatoren . . 45
5.4 Logische Operatoren 46
5.4.1 TRUE und FALSE in C 46
5.4.2 Der Datentyp `_Bool` (neu in C99) 46
5.4.3 Die C-Operatoren für NOT, AND und OR im Überblick 47
5.4.4 Der Negations-Operator `!` 47
5.4.5 Der AND-Operator `&&` 48
5.4.6 Der OR-Operator || 48
5.4.7 Beispiel zum neuen C99-Datentyp `_Bool` bzw. `bool` 49
5.4.8 Die Priorität der logischen Operatoren . . . 49
5.4.9 Keine unnötige Auswertung rechts von `&&` und || . 50

5.4.10 Übung: Überprüfungen mit logischen Operatoren . 51
5.5 Bit-Operatoren 52
5.5.1 Die Bit-Operatoren im Überblick 52
5.5.2 Bitweise Invertierung mit ~ 52
5.5.3 Bitweise AND-Verknüpfung mit & 53
5.5.4 Bitweise OR-Verknüpfung mit | 55
5.5.5 Bitweise XOR-Verknüpfung mit ^ 57
5.5.6 Bit-Operatoren nur für ganzzahlige Datentypen erlaubt 58
5.5.7 Fallgruben 59
5.5.8 Übung: Überprüfungen mit Bit-Operatoren . . 61
5.6 Shift-Operatoren 61
5.6.1 Die beiden Shift-Operatoren << und >> . . . 61
5.6.2 Shift-Operatoren nur für ganzzahlige Datentypen erlaubt 63
5.6.3 Priorität der Shift-Operatoren 64
5.6.4 Übungen 64
5.7 Zusammengesetzte Zuweisungsoperatoren 65
5.7.1 Die zusammengesetzten Zuweisungsoperatoren . 65
5.7.2 Übung zu den zusammengesetzten Operatoren . 67
5.8 Inkrement- und Dekrement-Operatoren 68
5.8.1 Inkrementieren und Dekrementieren mit ++ und -- 68
5.8.2 Präfix- und Postfix-Schreibweise für ++ und -- . 68
5.8.3 ++ und -- ist nur für Variablen erlaubt . . . 70
5.8.4 ++ und -- ist nicht auf linken Seite einer Zuweisung erlaubt 70
5.8.5 Übung zu den Inkrement- und Dekrement-Operatoren 71
5.9 Priorität und Anwendbarkeit der Operatoren . . . 71
5.9.1 Prioritätstabelle für Operatoren 71
5.9.2 Assoziativität der Operatoren 72
5.9.3 Erlaubte und unerlaubte Operationen für die C-Datentypen 73
5.9.4 Priorität und Auswertungszeitpunkt bei ++ und -- . 74
5.9.5 Fallgrube: Zugriff auf nicht vorbesetzte Variablen . 75
5.9.6 Übungen 76

6 Symbolische Konstanten 77
6.1 Konstanten-Definition mit `#define` 77
6.1.1 Die Direktive `#define` 77
6.1.2 Regeln für Konstanten-Namen bei `#define` . . 79
6.1.3 Konstanten machen Programm leicht änderbar . . 79
6.2 Konstanten-Definition mit `const` 79
6.3 Übungen 80
6.3.1 Volumen und Oberfläche einer Kugel 80
6.3.2 Das Phänomen der entfesselten Erde 81
6.3.3 Benzinverbrauch und Durchschnitts-Geschwindigkeit 81
6.3.4 Geldscheine stapeln 82

7 Ein- und Ausgabe 83
7.1 Headerdateien und `#include` 83

7.1.1 Bibliotheken und Headerdateien 83
7.1.2 Eigene Headerdateien 85
7.2 Ein- und Ausgabe eines Zeichens 85
7.2.1 `getchar()` und `putchar()` 85
7.2.2 Gepufferte Eingabe bei `getchar()` 86
7.2.3 Puffer-Bereinigung mit Dummy-`getchar()` . . 89
7.2.4 Puffer-Bereinigung ist nicht immer notwendig . . 91
7.2.5 Fallgrube: Zahlen nicht mit `getchar()` einlesen . 92
7.2.6 Die Headerdatei `ctype.h` 93
7.2.7 Einfache Makros 95
7.2.8 Übung: Umrechnung von Geschwindigkeiten . . 100
7.3 Die Ausgabe mit `printf()` 101
7.3.1 Die Funktion `printf()` 101
7.3.2 Fallgruben 108
7.3.3 Tipps 109
7.3.4 Übung: Die Capture-Recapture Methode . . . 112
7.4 Die Eingabe mit `scanf()` 113
7.4.1 Die Funktion `scanf()` 113
7.4.2 Fallgruben 118
7.4.3 Die Headerdatei `math.h` 121
7.4.4 Fallgrube: Vergessen von `#include <math.h>` . 127
7.4.5 Übungen 128

8 Datentypumwandlungen 131
8.1 Implizite Datentypumwandlungen 131
8.1.1 Der `sizeof`-Operator 131
8.1.2 Implizite Datentypumwandlungen 133
8.1.3 Fallgrube: Zuweisen von Ganzzahlausdrücken an Gleitpunktvariablen 140
8.1.4 Übung: Sparschweininhalt aufaddieren . . . 141
8.2 Explizite Datentypumwandlungen 142
8.2.1 Explizite Datentypumwandlungen mit cast-Operator 142
8.2.2 Fallgruben 142
8.2.3 Übung: Prozentzahlen für die Kandidaten bei einer Wahl 145

9 Die Headerdateien `limits.h` und `float.h` 147
9.1 <limits.h> - Grenzwerte von Ganzzahltypen . . . 147
9.1.1 Die Headerdatei `limits.h` 147
9.1.2 Übung: Wertebereiche der ganzzahligen Datentypen 148
9.2 `<float.h>` – Grenzwerte von Gleitpunkt-Datentypen . 149
9.2.1 Das IEEE-Format 149
9.2.2 Die Headerdatei `float.h` 151
9.2.3 Umwandlung gebrochener Dezimalzahlen in das Dualsystem 152
9.2.4 Hexadezimale Aus-/Eingabe von Gleitpunktzahlen im IEEE-Format (neu in C99) 153

9.2.5 Übung: Eigenschaften von Gleitpunkttypen . . 155

10 Anweisungen und Blöcke 157

11 Die `if`-Anweisung 159
11.1 Die zweiseitige `if`-Anweisung 159
11.2 Die einseitige `if`-Anweisung 166
11.3 Verschachtelte `if`-Anweisungen 168
11.4 Tipp: Einrücken untergeordneter Programmteile . . 174
11.5 Fallgruben 174
11.5.1 Falsche Gleichheitsüberprüfung 174
11.5.2 Keine unnötige Auswertung rechts von `&&` und `||` . 175
11.5.3 Vergleiche von negativen Zahlen mit `unsigned`-Variablen 176
11.5.4 Hohe Priorität des Negations-Operators `!` 177
11.6 Programmiertechniken 178
11.6.1 `if`-Kaskaden 178
11.6.2 Pseudocode 180
11.7 Übungen 181
11.7.1 Schaltjahre (Struktogramm in C-Programm umformen) 181
11.7.2 Rechnungen erstellen 181

12 Die bedingte Bewertung 183
12.1 Die bedingte Bewertung `?:` 183
12.2 Priorität des bedingten Operators 186
12.3 Tipp: Alternative Ausgaben bei `printf()` 187
12.4 Übung: Idealgewicht 187

13 Die `switch`-Anweisung 189
13.1 Die `switch`-Anweisung 189
13.2 Fallgrube: `case`-Marken müssen ganzzahlige Konstanten sein 196
13.3 Tipps 197
13.3.1 Alle `case`-Marken (auch letzte) mit `break` abschliessen 197
13.3.2 `default` immer angeben 197
13.4 Übung: Fläche, Umfang und Radius eines Kreises . . 198

14 Der Komma-Operator 199
14.1 Der Komma-Operator 199
14.2 Komma-Operator hat die niedrigste Priorität 200
14.3 Übung: Zusammenfassen mehrerer Anweisungen zu einer 200

15 Die `for`-Anweisung 201
15.1 Die `for`-Anweisung 201
15.2 Die `for`-Schleife und der Komma-Operator 207
15.3 Fallgrube: Semikolon am Ende des `for`-Schleifenkopfs . 210
15.4 Geschachtelte Schleifen 211
15.5 Eine endlose `for`-Schleife 219
15.6 `for` bei Durchläufen mit festen Schrittweiten 219

15.7 Variablendeklaration im for-Schleifenkopf (neu in C99) . 222
15.8 Programmiertechniken 223
15.8.1 Anhalten einer Bildschirmausgabe 223
15.8.2 Zeilenvorschübe bei geschachtelten Schleifen . . 226
15.8.3 Kombinieren mit for-Schleifen 230
15.8.4 Zwischeninformationen bei rechenintensiven Programmen 233
15.8.5 Merker in for-Schleifen bei Eintreten von Ereignissen 235
15.9 Fallgruben 236
15.9.1 Niemals die Laufvariable im Schleifenkörper ändern 236
15.9.2 Gleitpunktzahlen niemals auf Gleichheit prüfen . 240
15.9.3 Laufvariable einer for-Schleife läuft über Endwert hinaus 242
15.10 Übungen 243
15.10.1 Berechnung der harmonischen Reihe 243
15.10.2 Ausgabe der Dominosteine 243

16 Die while-Anweisung 245
16.1 Die while-Anweisung 245
16.2 Programmiertechniken 249
16.2.1 while bei unbekannter Zahl von Schleifendurchläufen 249
16.2.2 Konsistenzprüfungen bei Eingaben 251
16.2.3 Die Konstante EOF 252
16.2.4 Minimum und Maximum in einer Zahlenfolge . . 253
16.3 Zufallszahlen in C 254
16.4 Übungen 264
16.4.1 Fritz und Hans essen Äpfel 264
16.4.2 Primfaktor-Zerlegung 264

17 Die do...while-Anweisung 267
17.1 Die do...while-Anweisung 267
17.2 Programmiertechniken 270
17.2.1 do...while-Schleifen nicht so oft wie while-Schleifen 270
17.2.2 Abschließendes } while immer in einer Zeile 270
17.3 Übungen 271
17.3.1 Zahlen raten 271
17.3.2 Armstrong-Zahlen 272

18 Die break-Anweisung 273
18.1 Die break-Anweisung 273
18.2 break bewirkt Verlassen einer Schleifenebene . . . 274
18.3 Programmiertechniken 275
18.3.1 Sofortiges Verlassen von Schleifen und switch . 275
18.3.2 Endlosschleifen und break 277
18.4 Übungen 278
18.4.1 Würfelspiel bis 100 278
18.4.2 Primzahlen 278

19 Die continue-Anweisung 279

19.1 Die continue-Anweisung 279
19.2 Programmiertechniken 282
19.2.1 continue nur im äußersten Notfall 282
19.2.2 Korrekte Programme müssen auch schnell sein . . 283
19.3 Datums- und Zeitangaben (<time.h>) 286
19.3.1 Konstanten 286
19.3.2 Datentypen 287
19.3.3 Funktionen 287
19.3.4 Einige Funktionen aus time.h im Überblick . . 290
19.4 Übung: Drei Zahlen zu einer Summe finden . . . 294

20 Marken und die goto-Anweisung 297
20.1 Marken und die goto-Anweisung 297
20.2 Programmiertechniken 298
20.2.1 goto nur im äußersten Notfall 298
20.2.2 Lesbarere und schnellere Programme mit goto . 298

21 Graphikprogrammierung unter Linux 301
21.1 Bezug und Installation von LCGI 301
21.1.1 Bezug von LCGI 301
21.1.2 Installation von LCGI 302
21.2 Benutzung von LCGI 302
21.2.1 Inkludieren von <graphics.h> 302
21.2.2 Angabe von main() mit Parametern 302
21.2.3 Kompilieren und Linken von Graphikprogrammen . 302
21.3 Einige für Graphik benötigte C-Konstrukte 303
21.3.1 Dynamisches Erstellen von Zeichenketten . . . 303
21.3.2 Kurze Beschreibung von Arrays 304
21.4 Graphikmodus ein- und ausschalten 305
21.5 Eingaben im Graphikmodus 305
21.6 Bildschirm-, Farben- und Pixel-Operationen . . . 310
21.7 Positionieren, Linien zeichnen und Farbe einstellen . . 314
21.8 Figuren zeichnen und ausfüllen 317
21.9 Einstellungen für Textausgaben 324
21.10 Externe Bilder laden, Bildteile speichern und einblenden . 327
21.11 Kuchenstücke malen 330
21.12 Graphikpaket neu bzw. anders einrichten 332
21.13 Arbeiten mit mehreren Zeichenfenstern 333
21.14 Programmierung der Maus 334
21.15 Transformation mathematischer Koordinaten in das Graphikfenster 337
21.16 Kurzer Einblick in Fraktale und Chaos 341
21.17 Übungen 344
21.17.1 Ermitteln der Zahl PI mit Regentropfen . . . 344
21.17.2 Basketball spielen 345
21.17.3 Die Kochsche Schneeflocke 346

21.17.4 Zeichnen und Füllen von Quadraten mit der Maus . 347

22 Funktionen 349

22.1 Allgemeines zu Funktionen 349
22.1.1 Allgemeines Beispiel zu Funktionen 349
22.1.2 Die Begriffe Parameter und Argumente . . . 351
22.1.3 Bibliotheken und Headerdateien 351
22.2 Erstellen eigener Funktionen 353
22.2.1 Definition von Funktionen in C89/C99 . . . 353
22.2.2 Definition von Funktionen in Alt-C 356
22.2.3 Die `return`-Anweisung 357
22.2.4 Funktionen ohne Rückgabewert 357
22.2.5 Forward-Deklarationen 359
22.2.6 Funktions-Prototypen 361
22.2.7 Implizite Datentypumwandlung beim Funktionsaufruf 366
22.2.8 Typische Anwendungsgebiete von Funktionen . . 369
22.2.9 Übungen 374
22.3 Die Parameter von Funktionen 376
22.3.1 Leere Parameterliste durch Angabe von `void` . . 376
22.3.2 Bei Funktionsaufrufen findet nur Wertübergabe statt 377
22.3.3 Call by reference 381
22.3.4 Auswertung der Argumente findet vor Funktionsaufruf statt 385
22.3.5 Fallgruben 386
22.3.6 Übung: Zeiten-Taschenrechner 388
22.4 Ellipsen-Prototypen für Funktionen mit variabler Argumentzahl 389
22.4.1 Reihenfolge der Argument-Ablage im Stack . . 389
22.4.2 Ellipsen-Prototypen 389
22.4.3 Abarbeiten variabel langer Argumentlisten . . 389
22.4.4 Verfahren zum Abarbeiten variabel langer Argumentlisten 390
22.4.5 Fallgruben 395
22.4.6 Übungen 396
22.5 Neuheiten in C99 396
22.5.1 Inline-Funktionen 396
22.5.2 Der vordefinierte Name `__func__` 398
22.5.3 Keine Unterstützung von implizitem `int` . . . 398
22.5.4 Keine impliziten Funktionsdeklarationen . . . 399
22.5.5 Einschränkungen bei `return` 399
22.6 Rekursive Funktionen 399
22.6.1 Allgemeines zu rekursiven Funktionen . . . 399
22.6.2 Einige typische Anwendungen für die Rekursion . 404
22.6.3 Übungen 412
22.7 Zeiger auf Funktionen 415
22.7.1 Zeiger auf Funktionen 415
22.7.2 Typische Anwendungen 418

23 Speicherklassen und Modultechnik 423

23.1 Gültigkeitsbereich, Lebensdauer, Speicherort . . . 423
23.1.1 Gültigkeitsbereich 423
23.1.2 Lebensdauer 431
23.1.3 Speicherort 431
23.1.4 Gültigkeit, Lebensdauer und Speicherort im Überblick 431
23.1.5 Übung: Ausgabe des Programms `block3.c` . . 432
23.2 Schlüsselwörter `extern, auto, static` und `register` 432
23.2.1 Das Schlüsselwort `extern` 432
23.2.2 Das Schlüsselwort `auto` 436
23.2.3 Fallgrube: Niemals Adressen von `auto`-Variablen zurückgeben 444
23.2.4 Das Schlüsselwort `static` 445
23.2.5 Das Schlüsselwort `register` 455
23.2.6 Übung: Ausgabe des Programms `speikla1.c` . 456
23.3 Die Schlüsselwörter `const` und `volatile` 457
23.3.1 Das Schlüsselwort `const` 457
23.3.2 Das Schlüsselwort `volatile` 459
23.3.3 Kombination von `const` und `volatile` . . . 460
23.3.4 Übung: Konstante Zeiger und Zeiger auf Konstanten 460
23.4 Die Modultechnik 461
23.4.1 Von der Maschinensprache bis zur Modultechnik . 461
23.4.2 Modultechnik und Information Hiding . . . 466
23.4.3 Software-Technik: Linker und Compiler . . . 471
23.4.4 Beispiel: Simulation von Turingmaschinen . . 472
23.4.5 Übung: Turingmaschine zur binären Addition von 1 485

24 Präprozessor-Direktiven 487
24.1 Bedingte Kompilierung 488
24.1.1 Präprozessor-Direktiven zur bedingten Kompilierung 488
24.1.2 Typische Anwendungen 491
24.1.3 Testen mit Makro `assert()` aus Headerdatei `<assert.h>` 495
24.2 Einkopieren von anderen Headerdateien 497
24.2.1 Die Präprozessor-Direktive `#include` . . . 497
24.2.2 Typische Anwendungen 498
24.3 Definition von Makros (`#define` und `#undef`) . . . 499
24.3.1 Definition von Konstanten mit `#define` . . . 500
24.3.2 Definition von Funktionsmakros mit `#define` . . 501
24.3.3 Operator #: Ersetzung von Makroparametern durch String 503
24.3.4 Operator ##: Zusammensetzen neuer Namen . . 503
24.3.5 Rekursive Makrodefinitionen 504
24.3.6 Makros mit variabler Anzahl von Argumenten (neu in C99) 505
24.3.7 Makrodefinitionen mit `#undef` wieder aufheben . 506
24.3.8 Unterschiede zwischen Funktionen und Makros . 507
24.4 Vordefinierte Makronamen 511
24.5 Die restlichen Präprozessor-Direktiven 512
24.5.1 `#line` – Festlegen einer neuen Zeilennumerierung . 512

24.5.2 #error – Ausgeben von Fehlermeldungen . . . 513
24.5.3 #pragma – Festlegen von compilerspezifischem Verhalten 513
24.5.4 # – Die Null-Direktive 513
24.6 Übung: Ausgeben der Konstanten aus limits.h und float.h 514

25 Zeiger und Arrays 517
25.1 Eindimensionale Arrays 517
25.1.1 Eindimensionale Arrays 517
25.1.2 Nur statische Arrays erlaubt (in C89) 522
25.1.3 Von Arrays belegter Speicherplatz 522
25.1.4 Fallgruben 523
25.1.5 Übungen 525
25.2 Mehrdimensionale Arrays 527
25.2.1 Zweidimensionale Arrays 527
25.2.2 Drei-, vier-, fünf- und sonstige mehrdimensionale Arrays 536
25.2.3 sizeof liefert die Größe eines Arrays . . . 536
25.2.4 Übung: Game of Life (Beispiel für zellulare Automaten) 537
25.3 Zusammenhänge zwischen Arrays und Zeigern . . . 539
25.3.1 Arrayname ist konstanter Zeiger auf erstes Element . 539
25.3.2 Zugriff auf Arrayelemente ist auch über Zeiger möglich 542
25.3.3 Unterschied zwischen Arraynamen und echtem Zeiger 547
25.3.4 Erlaubte Operationen mit Zeigern 551
25.3.5 Unerlaubte Operationen mit Zeigern 552
25.3.6 Übergabe eines Arrays an eine Funktion mittels Adresse 555
25.3.7 call by value für Arrays (Zeiger) 559
25.3.8 Nachlese zu Arrays und Zeiger 560
25.3.9 Algorithmus: Der Bubble-Sort 560
25.3.10 Verwendung der Bibliotheksfunktion qsort() . 562
25.3.11 Algorithmus: Binäre Suche 563
25.3.12 Verwendung der Bibliotheksfunktion bsearch() . 566
25.3.13 Übungen 568
25.4 Strings und char-Zeiger 569
25.4.1 Besonderheiten von C-Strings 569
25.4.2 Das Schlüsselwort restrict für Zeiger (neu in C99) 571
25.4.3 Eigene Realisierung der Funktion strcpy() mit Arrays 572
25.4.4 Eigene Realisierung der Funktion strcpy() mit Zeigern 573
25.4.5 Die Headerdatei <string.h> 576
25.4.6 Umwandeln von Strings in numerische Werte . . 594
25.4.7 Umwandeln von numerischen Werten in Strings . 601
25.4.8 Besonderheiten beim Einlesen von Strings mit scanf() 603
25.4.9 Ein- und Ausgabe von Strings mit gets() und puts() 604
25.4.10 Unterschied zwischen Zeiger- und Array-Deklaration 605
25.4.11 Direkter Zugriff auf Zeichen in einer String-Konstante 607
25.4.12 Übungen 608
25.5 Array-Initialisierungen 609
25.5.1 Initialisierung von Arrays 610

25.5.2 Dimensionierungsangaben bei der Initialisierung . 613
25.5.3 Zeiger auf unbenamte Arrays (neu in C99) . . 615
25.5.4 Implizite Initialisierung bei `static`-Variablen/Arrays 615
25.5.5 Initialisierung lokaler Variablen auch mit Nicht-Konstanten 616
25.5.6 Initialisierung von lokalen Arrays in C89/C99 . . 617
25.5.7 Initialisierung von lokalen Arrays mit variablen Werten (neu in C99) 618
25.5.8 Initialisierung von lokalen Arrays mit 0 oder NULL . 619
25.5.9 Initialisierte Arrays mit `const` vor Überschreiben schützen 620
25.5.10 Übungen 620
25.6 Lokale Arrays variabler Länge (neu in C99) 622
25.7 Zeigerarrays und Zeiger auf Zeiger 623
25.7.1 Einfache Zeigerarrays 623
25.7.2 Zeiger auf Arrays 624
25.7.3 Vertauschen von zwei Arrays über Zeiger . . . 625
25.7.4 Übergabe von Arrays an Funktionen 627
25.7.5 Zeiger-Zeiger 629
25.7.6 Unterschiede zwischen zweidimensionalen Arrays und Zeigerarrays 629
25.7.7 Zugriff auf beliebige Elemente in einem Zeigerarray 633
25.7.8 Zeigerarrays mit Funktionsadressen 639
25.7.9 Übungen 641

26 Argumente auf der Kommandozeile 645
26.1 Die Parameter `argc` und `argv` der Funktion `main()` . 645
26.2 Optionen auf der Kommandozeile 648
26.3 Optionen auswerten mit der Funktion `getopt()` . . 655
26.4 Übung: Konvertieren von Dezimalzahlen in Dual, Oktal und Hexa 657

27 Dynamische Speicher-Reservierung und -Freigabe 659
27.1 Nachteile von statischen Arrays 659
27.1.1 Gefahr der Speicherüberschreibung 660
27.1.2 Speicherplatzvergeudung 661
27.2 Speicher reservieren mit `malloc()` 662
27.2.1 Die Funktion `malloc()` 662
27.2.2 Dynamische Arrays für beliebige Datentypen . . 667
27.2.3 Konvertierung von `void`-Zeigern 670
27.3 Speicher reservieren und initialisieren mit `calloc()` . 671
27.4 Größenänderung eines allozierten Speicherbereichs mit `realloc()` 672
27.4.1 Die Funktion `realloc()` 672
27.4.2 Besonderheiten der Funktion `realloc()` . . . 676
27.4.3 Schnellere Programme mit größeren Speicherblöcken 676
27.5 Freigeben eines dynamisch reservierten Speicherbereichs mit `free()` 679
27.5.1 Die Funktion `free()` 679

27.5.2 Fallgrube: `free()` setzt übergebenen Zeiger nicht auf NULL 679
27.5.3 Tipp: Eigenes Makro zur Freigabe von dynamischen Speicher 681
27.5.4 Fallgrube: `free()` nur auf von `malloc()`, `calloc()` und `realloc()` gelieferte Zeiger 681
27.6 Fallgrube: Allozieren von Speicherplatz in einer Funktion . 683
27.7 Programmiertechnik: Dynamische Zeiger-Arrays . . 687
27.8 Fallgrube: `free()` bei Zeiger-Arrays 688
27.9 Übung: Numerierte oder Rückwärtige Ausgabe eines Textes 688

28 Strukturen 689
28.1 Deklaration und Definition von Strukturen 689
28.1.1 Deklaration von Strukturen 689
28.1.2 Wichtige Regeln und Hinweise für Strukturdeklarationen 690
28.1.3 Definition von Strukturvariablen 691
28.1.4 Zusammenfassung von Strukturdeklaration und -definition 693
28.1.5 Namenlose Strukturen 694
28.2 Operationen mit Strukturvariablen 695
28.2.1 Zugriff auf Strukturkomponenten mittels Punktoperator 695
28.2.2 Zuweisung zwischen Strukturkomponenten . . 696
28.2.3 Zuweisung ganzer Strukturvariablen 701
28.2.4 Vergleich von Strukturvariablen ist nicht möglich . 702
28.2.5 Casting für komplette Strukturvariable ist nicht möglich 702
28.2.6 Adreß- und `sizeof`-Operator für Strukturvariablen erlaubt 703
28.2.7 Übung: Bruchrechner 704
28.3 Initialisierung von Strukturvariablen 705
28.3.1 Initialisierung von Strukturvariablen in C89 und C99 705
28.3.2 Initialisierung von Strukturvariablen (nur in C99) . 706
28.4 Strukturarrays 709
28.4.1 Arrays von Strukturvariablen 709
28.4.2 Übung: Zählen der Schlüsselwörter in einem C-Programm 719
28.5 Strukturen als Funktionsparameter 719
28.5.1 Übergabe von Strukturen an Funktionen . . . 719
28.5.2 Übung: Tagesdifferenz zwischen zwei Daten . . 722
28.6 Zeiger und Strukturen 722
28.6.1 Allgemeines zu Zeiger und Strukturen . . . 722
28.6.2 Dynamische Strukturarrays 737
28.6.3 Rekursive Strukturen 745
28.6.4 Übungen 779
28.7 Strukturen mit variabel langen Arrays (neu in C99) . . 782
28.8 Spezielle Strukturen (Unions und Bitfelder) 783
28.8.1 Unions 783
28.8.2 Bitfelder 788

29 Eigene Datentypen 795
29.1 Definition eigener Datentypnamen mit `typedef` . . 795

29.1.1 Vergabe von neuen Namen an existierende Datentypen mit typedef . . . 795
29.1.2 Höhere Portabilität und bessere Lesbarkeit durch typedef 798
29.2 Definition eigener Datentypen mit enum 801
29.2.1 Definition eigener Datentypen mit enum . . . 801
29.2.2 Regeln für enum 804
29.3 Übung: Mischtabelle aus der Chemie 805

30 Dateien **807**
30.1 Höhere E/A-Funktionen 808
30.1.1 Vordefinierte Struktur FILE 808
30.1.2 Öffnen und Schließen von Dateien 809
30.1.3 Lesen und Schreiben in Dateien 813
30.1.4 Unterschied zwischen Text- und Binärmodus . . 837
30.1.5 Positionieren in Dateien 840
30.1.6 Öffnen einer Datei mit existierenden Stream . . 846
30.1.7 Löschen und Umbenennen von Dateien . . . 848
30.1.8 Pufferung 848
30.1.9 Temporäre Dateien 851
30.1.10 Ausgabe von System-Fehlermeldungen . . . 854
30.1.11 Übung: Ausgeben einer Datei mit Zeilennumerierung 859
30.2 Elementare E/A-Funktionen 859
30.2.1 Filedeskriptoren 860
30.2.2 Öffnen und Schließen von Dateien 860
30.2.3 Lesen und Schreiben in Dateien 863
30.2.4 Positionieren in Dateien 868
30.2.5 Effizienz von E/A-Operationen 870
30.2.6 Filedeskriptoren und der Datentyp FILE . . . 872
30.2.7 Übung: Anhängen einer Datei an eine andere . . 873

31 Dateien, Directories und ihre Attribute **875**
31.1 Dateiattribute 875
31.1.1 Struktur stat – Attribute zu einer Datei . . . 875
31.1.2 stat() und fstat() – Erfragen von Dateiattributen 876
31.2 Dateiarten 877
31.3 Zugriffsrechte einer Datei 878
31.3.1 chmod() – Ändern der Zugriffsrechte für eine Datei 878
31.3.2 access() – Prüfen der Zugriffsrechte für eine Datei 879
31.4 Größe einer Datei 880
31.5 Zeiten einer Datei 881
31.6 Directories (Verzeichnisse) 882
31.6.1 mkdir() – Anlegen eines neuen Directory . . 882
31.6.2 rmdir() – Löschen eines leeren Directory . . 882
31.6.3 chdir() – Wechseln in ein neues Directory . . 882
31.6.4 getcwd() – Erfragen des Working-Directory-Pfadnamens 882

31.6.5 struct dirent – Aufbau eines Eintrags in einer Directory-Datei 884
31.6.6 opendir(), readdir(), rewinddir() und closedir() – Lesen von Directories 884
31.6.7 Durchlaufen eines ganzen Directorybaums . . 886
31.7 Gerätedateien 890
31.8 Übung: Ermitteln der Bytes eines Directory 892

32 Die Umgebung eines ablaufenden Programms 893
32.1 Start eines Prozesses 893
32.1.1 Startup-Routine - Startadresse eines Programms . 894
32.1.2 main() – Benutzerdefinierter Startpunkt eines Programms 894
32.2 Beendigung eines Prozesses 894
32.2.1 Exit-Status eines Prozesses 894
32.2.2 Normales Beenden der Funktion main() mit return 897
32.2.3 exit() – Normales Beenden eines Programms mit cleanup 897
32.2.4 _exit() – Beenden eines Programms ohne cleanup 897
32.2.5 atexit() – Einrichten von Exithandlern . . . 898
32.2.6 Start und Beendigung eines Prozesses im Überblick . 899
32.3 Environment eines Prozesses 900
32.3.1 Environment-Liste 900
32.3.2 Zugriff auf die ganze Environment-Liste . . . 900
32.3.3 getenv() – Erfragen einzelner Environment-Variablen 902
32.3.4 putenv(), setenv() und unsetenv() – Ändern, Hinzufügen oder Löschen von Environment-Variablen . 902
32.3.5 Übung: Realisierung des which-Kommandos . . 903

33 Starten eines anderen Programms 905
33.1 Die Funktion system() 905
33.2 Die exec()-Funktionen 906
33.2.1 Unterschiede der exec()-Funktionen im Überblick . 907
33.2.2 Interpretation des Dateinamens bei execlp() und execvp() 908
33.2.3 Unterschiede in der Form der Argumentübergabe . 908
33.2.4 Unterschiede bei Benutzung der Environment . . 908
33.2.5 Vererbungen bei exec() 909
33.3 Übung: Interaktiver Directory-Wechsel 910

34 Signale 913
34.1 Das Signalkonzept und die Funktion signal() . . . 913
34.1.1 signal() – Einrichten von Signalhandlern . . 914
34.1.2 Signale und die exec-Funktionen 916
34.2 Signalnamen und Signalnummern 916
34.3 Mögliche Probleme mit der signal()-Funktion . . 917
34.3.1 Erfragen des aktuellen Signalstatus ohne Änderung nicht möglich 918

34.3.2 Zeitspanne zwischen Auftreten eines Signals und Aufruf der signal()-Funktion 918
34.3.3 Endlosschleifen beim Warten auf das Eintreten von Signalen 919
34.4 Anormale Beendigung mit Funktion abort() 920
34.5 Anhalten eines Prozesses mit Funktion sleep() . . 920
34.6 Übung: Reaktionstest über Signale 920

35 Nicht-Lokale Sprünge 923
35.1 setjmp() und longjmp() – Springen über Funktionsgrenzen 923
35.1.1 Der Datentyp jmp_buf 924
35.1.2 setjmp() – Einrichten eines Ansprungpunktes . 925
35.1.3 longjmp() – Sprung zu einem mit setjmp() markierten Punkt 925
35.2 Rückkehr über mehrere Ebenen bei bestimmten Ereignissen 925
35.3 auto-, register-, static- und volatile-Variable bei longjmp() 931

36 Weitere Headerdateien 933
36.1 Die Headerdateien von C89 und C99 im Überblick . . 933
36.2 Die Headerdatei <locale.h> 935
36.2.1 Allgemeines zur Headerdatei <locale.h> . . 935
36.2.2 Die Funktion setlocale() 936
36.2.3 Die Funktion localeconv() 937
36.3 Wide-Character-Funktionen 942
36.3.1 Wide-Character-Klassifizierungsfunktionen . . 943
36.3.2 Wide-Character-E/A-Funktionen 945
36.3.3 Wide-Character-Stringfunktionen 946
36.3.4 Wide-Character-Speicherfunktionen 947
36.3.5 Wide-Character-String-Konvertierungsfunktionen . 948
36.3.6 Multibyte/Wide-Character-Konvertierungsfunktionen 948
36.4 Die Headerdatei <complex.h> (neu in C99) . . . 951
36.5 Die Headerdatei <fenv.h> (neu in C99) 955
36.6 Die Headerdatei <stdint.h> (neu in C99) 958
36.7 Die Headerdatei <inttypes.h> (neu in C99) . . . 959
36.8 Die Headerdatei <tgmath.h> (neu in C99) 964
36.9 Die Headerdatei <iso646.h> (neu in C99) 965

37 Lösungen zu den Übungen 967
37.1 Lösungen zu Kapitel 1 967
37.2 Lösungen zu Kapitel 2 968
37.3 Lösungen zu Kapitel 3 972
37.4 Lösungen zu Kapitel 4 973
37.5 Lösungen zu Kapitel 5 973
37.6 Lösungen zu Kapitel 6 979
37.7 Lösungen zu Kapitel 7 981
37.8 Lösungen zu Kapitel 8 985
37.9 Lösungen zu Kapitel 9 986

37.10 Lösungen zu Kapitel 10 988
37.11 Lösungen zu Kapitel 11 988
37.12 Lösungen zu Kapitel 12 990
37.13 Lösungen zu Kapitel 13 991
37.14 Lösungen zu Kapitel 14 992
37.15 Lösungen zu Kapitel 15 992
37.16 Lösungen zu Kapitel 16 993
37.17 Lösungen zu Kapitel 17 995
37.18 Lösungen zu Kapitel 18 996
37.19 Lösungen zu Kapitel 19 999
37.20 Lösungen zu Kapitel 20 1002
37.21 Lösungen zu Kapitel 21 1002
37.22 Lösungen zu Kapitel 22 1010
37.23 Lösungen zu Kapitel 23 1019
37.24 Lösungen zu Kapitel 24 1023
37.25 Lösungen zu Kapitel 25 1025
37.26 Lösungen zu Kapitel 26 1038
37.27 Lösungen zu Kapitel 27 1040
37.28 Lösungen zu Kapitel 28 1041
37.29 Lösungen zu Kapitel 29 1047
37.30 Lösungen zu Kapitel 30 1048
37.31 Lösungen zu Kapitel 31 1050
37.32 Lösungen zu Kapitel 32 1052
37.33 Lösungen zu Kapitel 33 1053
37.34 Lösungen zu Kapitel 34 1054

38 Anhang **1057**
38.1 Wichtige Tabellen zum Nachschlagen 1057
38.1.1 Prioritätstabelle für die Operatoren 1057
38.1.2 Wertebereiche für die einzelnen Datentypen . . 1058
38.1.3 Die Funktion `printf()` 1059
38.1.4 Die Funktion `scanf()` 1061
38.2 Dezimal-, Hexa-, Oktal- und Dualtabelle 1063
38.3 ASCII-Tabelle 1065
38.4 Literaturverzeichnis 1066
38.5 Bestelldaten für Übungen und Lösungen zu diesem Buch . 1068

Danksagung

Zunächst möchten wir *Nico Millin* und *Dr. Markus Wirtz* von SuSE PRESS unseren Dank dafür aussprechen, dass sie sich bereit erklärten, dieses Buch zu veröffentlichen, und dann auch selbst Hand anlegten, um dieses Buch zu setzen. Trotz aller Widrigkeiten, die ein Setzen eines Buches nun mal mit sich bringt, war es immer eine angenehme Zeit, mit ihnen zusammenzuarbeiten.
Des weiteren möchten wir uns noch ganz besonders bei unserem Kollegen *Andreas Jaeger* von SuSE Labs bedanken, der dieses Buch zweimal von Anfang bis Ende durchgesehen hat und uns stets mit Rat und Tat zur Seite stand, als wir die Änderungen, die der im Jahre 1999 neu eingeführte C-Standard mit sich brachte, einarbeiteten.
Auch möchten wir uns bei *Dieter Bloms*, *Jörg Dippel*, *Klaas Freitag*, *Bruno Gerz*, *Bernhard Hoelcker*, *Björn Jacke*, *Dirk Pankonin*, *Wolfgang Rosenauer*, *Christian Steinrücken* und *Peter Varkoly* für ihre konstruktiven Beiträge und Anregungen, die sie während des Korrekturlesens vorschlugen und die wir sehr gerne in dieses Buch einarbeiteten, herzlich bedanken.

Weisendorf-Neuenbürg, im Februar 2004 — *Prof. Dr. Helmut Herold*
Nürnberg, im Februar 2004 — *Jörg Arndt*

Kapitel 0

Einleitung

Wer nur einen Hammer besitzt,
für den sieht die ganze Welt wie ein Nagel aus.
Mark Twain

0.1 Ein paar Worte zu C und seiner Geschichte

Zu Beginn wollen wir kurz auf die Historie von C und die verschiedenen Standardisierungen von C eingehen. Zudem soll auch gleich vorweg geklärt werden, dass die Sprache C++ nur eine objektorientierte Erweiterung der Sprache C ist. Dies bedeutet, dass für eine Programmierung in C++ die Kenntnis der Sprache C eine unabdingbare Voraussetzung ist.

0.1.1 Die Entstehung von C

Die Sprache C wurde im Jahre 1972 von *Dennis M.Ritchie* in den Bell-Laboratorien bei AT&T entwickelt und von *Brian W. Kernighan* in den Jahren 1973/74 weiter verbessert. Vorläufer von C waren die Sprachen *BCPL* (Basic Combined Programming Language) und *B*.
C war schon immer sehr eng mit dem Betriebssystem Unix verbunden, da die Sprache C auf diesem damals noch jungen Betriebssystem entwickelt wurde und Unix wiederum selbst mit allen seinen Dienstprogrammen nahezu vollständig in C geschrieben wurde. In den 80er Jahren hat sich aber C als eine universell einsetzbare Sprache entpuppt, was dazu führte, dass C heute auf nahezu jedem beliebigen Betriebssystem (Linux, Unix-Systeme, MS-DOS, Windows95/98/NT/..., VMS, OS/2, ...) angeboten wird.

0.1.2 C als Vorstufe zu C++

C++ wurde – beginnend im Jahr 1979 – von *Bjarne Stroustrup* entwickelt. C++, das eine objektorientierte Version zu C ist, hat sich inzwischen sehr verbreitet. Da aber

C++ nur eine Erweiterung zu C ist, muss auch jeder, der mit C++ programmieren will, die Sprache C richtig beherrschen, bevor er in die höheren Sphären der objektorientierten Programmierung aufsteigen kann.
Leser, die an den Erweiterungen der Sprache C++ zu C interessiert sind, seien auf die beiden folgenden Bücher verwiesen:

- *Das Qt-Buch. Portable GUI-Programmierung unter Linux/Unix/Windows*
  Dieses ebenfalls bei SuSE Press erschienene Buch gibt nicht nur eine Einführung in C++, sondern stellt auch die Qt-Klassenbibliothek vor. Mit der Qt-Klassenbibliothek lassen sich leicht Graphik-Oberflächen entwerfen, die plattformunabhängig sind und ohne jeglichen Portierungsaufwand unter den heute weit verbreiteten Betriebssystemen Linux/Unix und Windows verwendet werden können.
- *Go To Objektorientierung; Angewandte Objektorientierung mit C++ und UML*
  Dieses beim Addison-Wesley-Verlag erschienene Buch führt den Leser in das Denken und die Praxis des objektorientierten Programmierens mit C++ ein. In diesem Buch wird nicht nur die Sprache C++ vorgestellt, sondern auch der objektorientierte Entwurf mit UML (*Unified Modeling Language*) anhand einer Vielzahl von Beispielen für den Leser nachvollziehbar gezeigt.

0.1.3 Die Standardisierung von C

Die Sprache C wurde – wie zuvor erwähnt – im Jahre 1972 von *Dennis M. Ritchie* entwickelt und in den nächsten Jahren von *Brian W. Kernighan* weiter verbessert. Zu dieser Zeit gab es keine Standardbeschreibung dieser Sprache. Stattdessen galt die 1. Ausgabe des Buches *„The C Programming Language“* von Kernighan und Ritchie (Prentice Hall, 1978) als die Bibel für alle C-Fragen. Diese „Bibel“ ließ jedoch einige Fragen offen. So wurde bereits in den frühen 80er Jahren die Notwendigkeit für einen wirklichen C-Standard erkannt. Nachfolgend werden die einzelnen Standards in ihrer historischen Reihenfolge kurz vorgestellt.

C89 – der erste Standard für C

Im Jahre 1983 begann das ANSI-Komitee X3J11[1] mit dem Unterfangen, die Sprache C zu standardisieren. Im selben Jahr noch entschied das Komitee X3J11, dass nur *ein* C-Standard geschaffen werden soll, welcher von beiden Organisationen ANSI und ISO verabschiedet wird. C wurde zum ersten Mal Ende des Jahres 1989 mit der Annahme des ANSI-Standards standardisiert. Diese Version von C wird allgemein als *C89* bezeichnet. Dieser Standard wurde auch im Jahre 1990 von ISO übernommen. C89 wurde im Jahre 1995 leicht verbessert.

[1] *ANSI* (**A**merican **N**ational **S**tandard **I**nstitute) ist eine amerikanische Organisation, welche Mitglied der **I**nternational **S**tandards **O**rganisation (ISO) ist.

C99 – der neue Standard für C

Im Jahre 1999 wurde ein neuer Standard für C geschaffen. Diese Version nennt sich *ISO C99*, oder kurz auch nur *C99*. C99 enthält eine Reihe von Verbesserungen und einige neue Konstrukte, welche teilweise von C++ übernommen wurden.
Alle C-Programme, die die von C99 vorgegebenen Vorschriften einhalten, werden *C99 kompatibel* genannt. Solche Programme haben den Vorteil, dass sie leicht portierbar sind, also ohne größere Änderungen von einem anderen C99-Compiler auf einer anderen Maschine in eine ablauffähige Version übersetzt werden können.
Sie sollten also immer größten Wert darauf legen, dass Sie C-Programme erstellen, die den von C89 bzw. C99 vorgegebenen Regeln entsprechen. Halten Sie sich nicht daran, werden Sie eventuell später erhebliche Schwierigkeiten haben, das Programm auf einer anderen Maschine mit eventuell einem anderen Compiler zu übersetzen.
In diesem Buch wird der neueste Standard von C (*C99*) beschrieben.

0.2 Hinweise zu diesem Buch und zum begleitenden Übungs- bzw. Lösungsbuch

Zu diesem Buch existiert ein eigens darauf abgestimmtes Übungs- sowie auch ein dazugehöriges Lösungsbuch. Das Übungsbuch enthält an die 500 fächerübergreifenden Übungen aus allen möglichen anderen Fachgebieten (Informatik, Mathematik, Allgemeinwissen, Elektrotechnik, Physik, Wirtschaft, ...) mit unterschiedlichen Schwierigkeitsgraden.
Nachfolgend nun noch einige Hinweise zu diesem Buch und dem begleitenden Übungs- bzw. Lösungsbuch:

1. **Dieses Buch** beschreibt die **Theorie der Programmiersprache C** anhand vieler Beispiele. Dabei begnügt es sich jedoch nicht allein mit der Vorstellung der einzelnen C-Elemente, sondern vermittelt dem C-Neuling auch Einblicke in wichtige Grundlagen der Informatik. Zudem gibt es auch zu den einzelnen C-Konstruktionen Programmiertechniken aus der C-Praxis, typische Anwendungsgebiete, Tipps und Fallgruben, die in C leider nicht allzu selten sind. Diese Grundkenntnisse bilden das Fundament, das für eine erfolgreiche Programmierung in C unverzichtbar ist.
2. Im **begleitenden Übungsbuch** befinden sich dann an die **500 Übungen** zu den jeweiligen Kapiteln dieses Buches. Zu jeder Übung ist dabei immer eine kurze Information angegeben, an der sich sofort folgendes erkennen läßt:
   - Fachgebiet (C-Syntax, Allg., Wirtschaft, Informatik, Mathematik, usw.).
   - Schwierigkeitsgrad
   - Ungefährer Umfang der Aufgabe

   Am Anfang jedes Kapitels des Übungsbuches werden Rätsel und Denksportaufgaben gegeben. Der Zweck der Übungen und Denksportaufgaben ist nicht allein das Erlernen der Sprache C, sondern diese Übungen und Rätsel sollen Schlüsselqualifikationen wie fächerübergreifendes Wissen und Denken sowie die heute in jedem Beruf unverzichtbare Problemlösungsfähigkeit fördern.

3. Im **Lösungsbuch** befinden sich dann alle Lösungen zu den entsprechenden Denksportaufgaben und Übungen des Übungsbuches. Die Lösungen zu den Rätseln und Denksportaufgaben enthalten dabei auch allgemeine Tipps und Techniken, die für systematische Lösungen von Problemstellungen allgemeiner Art sehr hilfreich sind.
4. Außerdem wird eine **CD-ROM** angeboten, auf der sich alle C-Programmbeispiele aus diesem Buch sowie die möglichen Programmlösungen zu den Aufgabenstellungen aus dem begleitenden Übungsbuch befinden.

Wenn Sie sich für das begleitende Übungs- und Lösungsbuch interessieren, wenden sie sich bitte direkt an den Autor (**siehe dazu Seite 1068**)
Nach dem Durcharbeiten dieser Unterlagen verfügt der Leser über ein gesundes und breites C-Fundament, und ist in der Lage, erfolgreich in C zu programmieren. Die Intention dieser Lern- und Lehrunterlagen ist es nämlich:

- den C-Anfänger systematisch vom C-Basiswissen bis hin zu den fortgeschrittenen Techniken der Programmierung zu führen.
- dem bereits erfahrenen C-Programmierer – aufgrund der Vielzahl von Tipps, fundamentalen Algorithmen und nützlichen Programmiertechniken – eine Vertiefung bzw. Ergänzung seines C-Wissens zu ermöglichen.

0.3 Downloads zu diesem Buch

Bezug aller Beispiel- und Übungsprogramme dieses Buches

Alle Beispiel- und Übungsprogramme dieses Buches können von folgenden Webseiten

```
http://www.susepress.de/de/download

http://www2.efi.fh-nuernberg.de/~herold/
```

heruntergeladen werden.

Windows-Kompatibilität

Auch wenn dieses Buch ursprünglich für die C-Programmierung unter Linux konzipiert wurde, so sind doch nahezu alle in diesem Buch vorgestellten Programme auch unter Windows-Compilern lauffähig. Einschränkungen gibt es lediglich bei einigen Windows-Compilern, die noch nicht den neuesten C99-Standard implementiert haben, aber das betrifft nur einige wenige Programme in diesem Buch.
Auch zu der ursprünglich für Linux eigens im Rahmen dieses Buches entwickelten Graphikbibliothek LCGI existiert zwischenzeitlich eine Windows-Version, die von den oben erwähnten Webseiten heruntergeladen werden kann.
Ebenso existiert zu den Directory-Zugriffen unter Linux/Unix in Kapitel 31 eine Windows-Portierung, die im übrigen einfacher als die Windows-eigenen Funktionen ist. Diese Portierung (`w32dirent.zip`) kann auch von den obigen Webseiten heruntergeladen werden.

Kapitel 21

Graphikprogrammierung unter Linux

Eine redselige Dame konsultierte einst Doktor Heim: „Herr Professor“, klagte sie ganz bewegt, „ich glaube, ich habe mich überanstrengt!“. „Na“, lächelte Heim leicht verschmitzt, „dann zeigen Sie mal Ihre Zunge.“

Im Rahmen dieses Buches wurden eigene Graphikroutinen entworfen, mit denen eine einfache Graphikprogrammierung möglich ist. Die hier entworfenen Graphikroutinen ähneln denen von BGI (*Borland Graphics Interface*) des früher unter MS-DOS sehr beliebten Turbo-C und Turbo-Pascal. Diese Graphik-Implementierung hat den Namen LCGI (*Linux C Graphics Interface*). Natürlich sind Programme, welche die in diesem Kapitel vorgestellten Graphikroutinen verwenden, nicht mehr C89- bzw. C99-konform, sondern sind nur unter dem X-Windows-System von Linux/Unix ablauffähig, wenn die Graphikbibliothek *Qt* der Firma *Trolltech* installiert ist, was wohl meistens der Fall sein wird; andernfalls müssen Sie *Qt* erst installieren.

21.1 Bezug und Installation von LCGI

21.1.1 Bezug von LCGI

Die Graphikbibliothek LCGI befindet sich in dem gezippten tar-Archiv

```
lcgi-1.0.tgz
```

und kann von der Webseite

```
http://www.susepress.de/de/download
```

heruntergeladen werden.

21.1.2 Installation von LCGI

Um die Graphikbibliothek LCGI zu installieren, müssen Sie sich als `root` anmelden und anschließend in das Directory wechseln, in dem sich das heruntergeladene Archiv *lcgi-1.0.tgz* befindet. Nun müssen Sie die folgenden Schritte durchführen:

1. Entpacken des Archivs `lcgi-1.0.tgz` mit

   ```
   linux:~ #  tar xzvf lcgi-1.0.tgz ⏎
   ```

2. Wechseln in das Directory `lcgi-1.0`

   ```
   linux:~ #  cd lcgi-1.0 ⏎
   ```

3. Lesen Sie die Datei *README*, die die weiteren Installationsschritte beschreibt.

21.2 Benutzung von LCGI

21.2.1 Inkludieren von `<graphics.h>`

Da die hier vorgestellten Funktionen in der Headerdatei `graphics.h` deklariert sind, sollte immer angegeben werden, wenn man von diesen Funktionen in seinem Programm Gebrauch macht.

```
#include <graphics.h>
```

21.2.2 Angabe von `main()` mit Parametern

Wenn Sie in Ihrem Programm Routinen aus diesem Kapitel aufrufen, müssen Sie statt

```
int main(void)
```

immer folgendes angeben:

```
int main( int argc, char *argv[] )
```

21.2.3 Kompilieren und Linken von Graphikprogrammen

Wenn Sie ein Programm erstellt haben, das Routinen aus diesem Kapitel aufruft, müssen Sie dieses immer mit dem mitgelieferten Kommando `lcc` statt `gcc` bzw. `cc` kompilieren und linken.
Wenn Sie z. B. ein Programm `polygon.c` entworfen haben und dieses nun kompilieren wollen, müssen Sie z. B. folgendes aufrufen:

```
user@linux:~ >   lcc -o polygon polygon.c ⏎
```

`lcc` bietet die gleichen Optionen an wie das Kommando `gcc` bzw. `cc`.
Geben Sie danach zum Testen die folgende Kommandozeile ein:

```
user@linux:~ >  polygon ⏎
```

bzw. die folgende, wenn sie als `root` angemeldet sind:

```
linux:~ #  ./polygon ⏎
```

Wenn Sie mit der integrierten Environment `xwpe` arbeiten, müssen Sie unter dem Menüpunkt *Options* → *Compiler* → *C* die folgenden Einträge vornehmen:

Compiler:	`g++`
Compiler-Options:	`-g -I/usr/include/lcgi`
Loader-Options:	`-L/usr/lib/qt2/lib`[1] `-L/usr/lib -L/usr/X11R6/lib -lXext -lm -llcgi -lqt -lXext -lX11`
LAnguage:	`C`
File-Postfix:	`.c`

Die so eingestellten Optionen müssen Sie dann über den Menüeintrag *Options* → *Save Options* sichern.

21.3 Einige für Graphik benötigte C-Konstrukte

In diesem Kapitel werden einige C-Konstrukte benötigt, die erst in späteren Kapiteln behandelt werden. Die hier vorgeschobenen kurzen Erklärungen sollten jedoch ausreichen, diese Konstrukte zu verstehen und dann auch richtig einsetzen zu können.

21.3.1 Dynamisches Erstellen von Zeichenketten

Manchmal möchte man sich dynamisch Zeichenketten „zusammenbauen". Dazu steht die Funktion `sprintf()` zur Verfügung:

```
sprintf( string, format, ... );
```

Diese Funktion `sprintf()` kann genauso wie die Funktion `printf()` verwendet werden. Anders als `printf()` schreibt diese Funktion die formatierte Zeichenkette nicht auf den Bildschirm, sondern in die Variable `string`. Die Variable `string` selbst muss jedoch zuvor z. B. mit

```
char string[100];
```

definiert werden. Statt 100 kann auch eine größere oder eine kleinere Zahl angegeben werden. Die hier angegebene Zahl legt jedenfalls die maximale Anzahl von Zeichen fest, die ein so hergestellter String haben kann. Hat er mehr Zeichen führt dies zu einer Speicherüberschreibung, was katastrophale Folgen für das jeweilige Programm haben kann. Da C das Ende einer Zeichenkette mit einem 0-Byte kennzeichnet, muss man sogar Platz für ein Zeichen mehr reservieren. Wenn man also z. B. maximal 10 Zeichen (wie z. B. `bild10.gif`) nach `dateiname` schreiben möchte, muss man mindestens Speicherplatz für 11 Zeichen reservieren:

```
char dateiname[11];
```

Möchte man z. B. unter 10 Graphikdateien, die die Namen `bild1.gif` bis `bild10.gif` haben, eine zufällige auswählen, so wäre dies mit dem folgenden C-Code möglich:

[1] Sollten Sie für `QTDIR` einen anderen Pfad als `/usr/lib/qt2` eingestellt haben, müssen Sie diesen hier anstelle von `/usr/lib/qt2` angeben.

```
char dateiname[20];
int  z;
.....

z = rand()%10+1;
sprintf(dateiname, "bild%d.gif", z);
printf("%s\n", dateiname); /* gibt bild1.gif, bild2.gif oder ... oder bild10.gif
                              aus */
```

21.3.2 Kurze Beschreibung von Arrays

Die beiden später vorgestellten Graphikfunktionen `drawpoly()` (zeichnet den Umriss eines Polygons) und `fillpoly()` (zeichnet ein ausgefülltes Polygon) machen von so genannten *Arrays* Gebrauch. Ein Array ist eine Zusammenfassung von mehreren hintereinanderliegenden Speicherplätzen unter einem Namen.
Mit folgender Deklaration wird z. B. ein Array mit dem Namen `a` deklariert:

```
int a[10];
```

Dieses Array kann nun 10 `int`-Werte aufnehmen, wie z. B.

```
   a[0]=125; /* Im 1.Speicherplatz des Arrays 125 ablegen */
   a[1]=345; /* Im 2.Speicherplatz des Arrays 345 ablegen */
   a[2]=72;  /* Im 3.Speicherplatz des Arrays 72 ablegen  */
   ........
   a[9]=737; /* Im 10.Speicherplatz des Arrays 737 ablegen */
```

Anstelle von 10 `int`-Variablen wurde also ein Array `a` deklariert, das 10 `int`-Variablen unter einem Namen zusammenfasst:

```
      a
       +---------+
   [0] |   125   |
       +---------+
   [1] |   345   |
       +---------+
   [2] |    72   |
       +---------+
   [3] |   ...   |
   ... |   ...   |
   ... |   ...   |
   [8] |   ...   |
       +---------+
   [9] |   737   |
       +---------+
```

Um die einzelnen Speicherplätze zu unterscheiden, sind diese von `0` bis `9` (nicht bis `10`) durchnumeriert, wobei die entsprechende Nummer in `[...]` direkt nach dem Array-Namen (hier `a`) anzugeben ist. Arrays werden ausführlich in Kapitel 25 auf Seite 517 beschrieben.

21.4 Graphikmodus ein- und ausschalten

`initgraph(int breite, int hoehe)`

schaltet den Graphikmodus ein, indem ein Fenster eingeblendet wird, das `breite` Pixel breit und `hoehe` Pixel hoch ist.

`closegraph()`

beendet den Graphikmodus. Da in diesem Fall auch das Graphikfenster gelöscht wird, wird meist vor `closegraph()` noch ein `getch()` angegeben, so dass erst auf einen Tastendruck des Benutzers hin der Graphikmodus verlassen wird.

Typisch für die Graphikprogrammierung ist deshalb z. B. folgender Programmausschnitt:

```
#include <graphics.h>
.......
int  main( int argc, char *argv[] )
{
    initgraph( 640, 480 );
       ........
         Graphik-Programmteil
       ........
    getch();  /* auf einen Tastendruck warten */
    closegraph();
}
```

21.5 Eingaben im Graphikmodus

Während man im Graphikmodus arbeitet, kann man nicht mehr die Standardroutinen für die Ein- und Ausgabe (`printf()`, `scanf()`, `getchar()` und `putchar()`) verwenden, sondern muss die eigens dafür angebotenen Routinen verwenden, welche nachfolgend vorgestellt sind:

`getcharacter(char *text, ...)`

fordert den Benutzer durch Ausgabe des Textes `text` zur Eingabe eines Zeichens auf und liefert das vom Benutzer eingegebene Zeichen zurück.

`gettext(char *text, ...)`

fordert den Benutzer durch Ausgabe des Textes `text` zur Eingabe eines Textes auf und liefert den vom Benutzer eingegebenen Text als Rückgabewert.

`getint(char *text, ...)`

fordert den Benutzer durch Ausgabe des Textes `text` zur Eingabe einer ganzen Zahl auf und liefert die vom Benutzer eingegebene Zahl als `int`-Rückgabewert.

`getdouble(char *text, ...)`

fordert den Benutzer durch Ausgabe des Textes `text` zur Eingabe einer Gleitpunktzahl auf und liefert die vom Benutzer eingegebene Zahl als `double`-Rückgabewert.

`kbhit()`

prüft, ob eine Taste gedrückt wurde. Falls eine Taste gedrückt wurde, liefert `kbhit()` einen `int`-Wert ungleich 0 (TRUE), falls nicht, liefert diese Funktion 0 (FALSE) zurück. Das dabei eingegebene Zeichen kann mit der nachfolgend vorgestellten Routine `getch()` nachträglich erfragt werden.

`getch()`

liest ein Zeichen ein.

Anders als bei `getchar()` findet hier jedoch keine Zwischenpufferung statt, sondern das Zeichen wird direkt von der Tastatur gelesen. Das eingegebene Zeichen wird dabei nicht am Bildschirm angezeigt. In jedem Fall liefert diese Routine den Tastencode des eingegebenen Zeichens zurück. Mit `getch()` lassen sich nicht nur ASCII-Zeichen, sondern auch beliebige andere Steuerzeichen, wie z. B. Funktionstasten, ↑, Bild↓ usw. einlesen. `getch()` liefert einen eigenen Code für das eingegebene Zeichen. Für jede einzelne Taste steht im Graphikmodus eine eigene Konstante zur Verfügung:

```
Verschiedene Tasten
     Key_Escape, Key_Tab,    Key_Backtab, Key_Backspace, Key_Return, Key_Enter,
     Key_Insert, Key_Delete, Key_Pause,   Key_Print,     Key_Sysreq,
Tasten zur Cursor-Steuerung
     Key_Home,   Key_End,      Key_Left, Key_Up, Key_Right, Key_Down,
     Key_Pageup, Key_Pagedown,
Umschalt-Tasten
     Key_Shift,    Key_Control, Key_Meta,      Key_Alt,
     Key_Capslock, Key_Numlock, Key_Scrolllock,
Funktions-Tasten
     Key_F1,  Key_F2,  Key_F3,  Key_F4,  Key_F5,  Key_F6,  Key_F7,  Key_F8,
     Key_F9,  Key_F10, Key_F11, Key_F12, Key_F13, Key_F14, Key_F15, Key_F16,
     Key_F17, Key_F18, Key_F19, Key_F20, Key_F21, Key_F22, Key_F23, Key_F24,
     Key_F25, Key_F26, Key_F27, Key_F28, Key_F29, Key_F30, Key_F31, Key_F32,
     Key_F33, Key_F34, Key_F35,
Sonder-Tasten
     Key_Super_L, Key_Super_R, Key_Menu, Key_Hyper_L, Key_Hyper_R,
7-Bit druckbare ASCII-Tasten
     Key_Space,     Key_Exclam,     Key_Quotedbl,  Key_Numbersign,
     Key_Dollar,    Key_Percent,    Key_Ampersand, Key_Apostrophe,
     Key_Parenleft, Key_Parenright, Key_Asterisk,  Key_Plus,
     Key_Comma,     Key_Minus,      Key_Period,    Key_Slash,
     Key_0, Key_1, Key_2, Key_3, Key_4, Key_5, Key_6, Key_7, Key_8, Key_9,
     Key_Colon, Key_Semicolon,
     Key_Less,  Key_Equal,     Key_Greater, Key_Question, Key_At,
     Key_A, Key_B, Key_C, Key_D, Key_E, Key_F, Key_G, Key_H, Key_I, Key_J,
```

```
Key_K, Key_L, Key_M, Key_N, Key_O, Key_P, Key_Q, Key_R, Key_S, Key_T,
Key_U, Key_V, Key_W, Key_X, Key_Y, Key_Z,
Key_Bracketleft, Key_Backslash, Key_Bracketright, Key_Asciicircum,
Key_Underscore,  Key_Quoteleft, Key_Braceleft,    Key_Bar,
Key_Braceright,  Key_Asciitilde
```

`outtextxy(int left, int top, int right, int bottom, char *text, ...)`

gibt den Text `text` in dem Rechteck aus, dessen linke obere Ecke durch `(left, top)` und dessen rechte untere Ecke durch `(right,bottom)` festgelegt wird. `outtextxy()` benutzt die Schriftart und Formatierung, die mit `settextstyle()` und `settextjustify()`[2] festgelegt werden kann.

Bei den Graphikfunktionen `getcharacter()`, `gettext()`, `getint()`, `getdouble()` und `outtextxy()` bedeuten die drei Punkte, dass `text` die Formatierungszeichen von `printf()` enthalten kann, und dass dann eventuell angegebene weitere Argumente entsprechend formatiert im Text eingebettet werden, wie z. B.:

```
int  ganz, unten, oben;
....
unten = 5;
oben = 107;

ganz = getint( "Gib eine Zahl zwischen %d und %d ein", unten, oben );
```

Dieser Programmausschnitt blendet ein Fenster ein, in dem der Benutzer mit dem Satz „`Gib eine Zahl zwischen 5 und 107 ein`“ zur Eingabe einer ganzen Zahl aufgefordert wird.
Eine weitere in der Graphikbibliothek angebotene Routine ist:

`delay(int millisekunden)`

hält die Programmausführung für `millisekunden` an.

Beispiel:
Das folgende Programm `einaus.c` demonstriert die Verwendung der Routinen zum Einlesen von Zeichen, Zahlen und Text.

```
#include <graphics.h>

int main( int argc, char *argv[] ) {
   char     zeich;
   int      iZahl;
   double   dZahl;
   const char *string;

   initgraph( 400, 250 );
   cleardevice( WHITE ); /* löscht den Inhalt des Graphikfensters
                            und setzt seinen Hintergrund auf weiss */
   zeich = getcharacter( "Gib ein Zeichen ein!" );
```

[2] siehe Kapitel 21.9 auf Seite 324

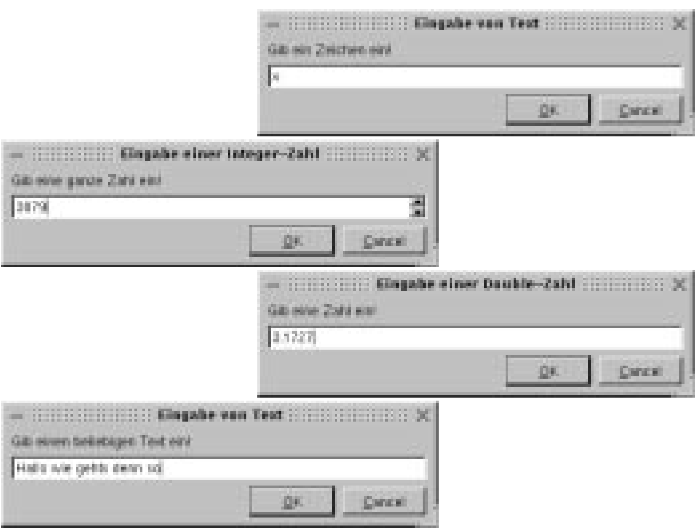


Abbildung 21.1: Fenster zur Eingabe von einzelnen Zeichen, Zahlen und Text

```
    iZahl = getint( "Gib eine ganze Zahl ein!" );
    dZahl = getdouble( "Gib eine Zahl ein!" );
    string = gettext( "Gib einen beliebigen Text ein!" );
    outtextxy( 100,  50, 400, 250, "Das Zeichen ist: %c", zeich );
    outtextxy( 100, 100, 400, 250, "Die ganze Zahl ist: %d", iZahl  );
    outtextxy( 100, 150, 400, 250, "Die Gleitpunktzahl ist: %g", dZahl  );
    outtextxy( 100, 200, 400, 250, "Der Text ist: %s", string  );
    getch();
    closegraph();
    return(0);
}
```

Abbildung 21.2: Anzeige der Eingaben im Hauptfenster

Das Programm `einaus.c` blendet die in Abbildung 21.1 gezeigten Fenster für die jeweiligen Eingaben ein. Nachdem der Benutzer alle geforderten Eingaben getätigt hat, werden ihm im Hauptfenster nochmals seine Eingaben angezeigt; siehe auch Abbildung 21.2.

Beispiel:

Mit dem folgenden C-Programm `reaktion.c` können Sie Ihre Reaktionszeit testen:

```
#include <graphics.h>
#include <stdlib.h>
#include <time.h>

int main( int argc, char *argv[] ) {
   double   zeit, min=1000000000;
   clock_t  start, ende;

   srand( time(NULL ) );
   initgraph( 640, 480 );

   do {
      cleardevice( WHITE ); /* loescht Graphikfenster und fuellt es mit weiss */
      outtextxy( 10, 10, 400, 400,
                "Reaktionstest\n"
                "============\n\n\n\n"
                "Gleich wird dieses Fenster gelb\n\n"
                "Dann musst du so schnell wie möglich eine Taste drücken....");
      delay( 2000+ rand()%4000 );
      if (kbhit()) {
         cleardevice( LIGHTRED ); /* loescht Graphikfenster u. fuellt’s hellrot*/
         outtextxy(100, 200, 400, 400,
                   "Du hast versucht zu schummeln. Das ist nicht erlaubt!\n");
         while (kbhit())
            getch();
      } else {
         cleardevice( YELLOW ); /* loescht Graphikfenster und fuellt es gelb */
         start=clock(); /* Stopp-Uhr beginnt zu ticken */
         while (!kbhit())
            ;
         ende=clock(); /* Stopp-Uhr wird angehalten */
         getch(); /* Ueberlesen des eingegebenen Zeichens */
         zeit = (ende-start)/(double)CLOCKS_PER_SEC;
         if (zeit<min)
            min=zeit;
         outtextxy( 100, 300, 500, 400,
                    "Du hast %g Sek. gebraucht. (Bisheriger Rekord: %g Sek.)\n",
                     zeit, min);
      }
      outtextxy( 100, 350, 400, 400,
                "....Willst du es noch einmal probieren (j/n) ? ");
   } while (getch()==Key_J);
```

```
   closegraph();
   return(0);
}
```

21.6 Bildschirm-, Farben- und Pixel-Operationen

Tabelle 21.1 zeigt die im Graphikmodus möglichen Farben.
Der Bildschirm entspricht bei dieser Graphik einem x-y-Koordinatensystem, dessen Nullpunkt die linke obere Ecke ist (x=0,y=0). Der Graphikcursor kann unter Angabe eines (x,y)-Werts positioniert werden:

Folgende Graphikroutinen können dabei verwendet werden:

`cleardevice(int farbe)`

löscht den ganzen Inhalt des Graphikfensters und füllt es mit Farbe `farbe`.

`getmaxx()` bzw. `getmaxy()`

liefern die größte x- bzw. y-Koordinate (als `int`-Wert) des Graphikfensters.

`getx()` bzw. `gety()`

liefern die aktuelle x- bzw. y-Koordinate (als `int`-Wert) des Graphikcursors.

Tabelle 21.1: Die im Graphikmodus möglichen Farbekonstanten bzw. -nummern

Name	Wert	auf deutsch
`BLACK`	0	schwarz
`BLUE`	1	blau
`GREEN`	2	grün
`CYAN`	3	türkis
`RED`	4	rot
`MAGENTA`	5	violett
`BROWN`	6	braun
`LIGHTGRAY`	7	hellgrau
`DARKGRAY`	8	dunkelgrau
`LIGHTBLUE`	9	hellblau
`LIGHTGREEN`	10	hellgrün
`LIGHTCYAN`	11	helles Türkis
`LIGHTRED`	12	hellrot
`LIGHTMAGENTA`	13	helles Violett
`YELLOW`	14	gelb
`WHITE`	15	weiss

`getmaxcolor()`

liefert die höchste Farbnummer (als `int`-Wert), die im Graphikmodus verwendet werden kann.

`putpixel(int x, int y, int farbe)`

zeichnet einen Punkt (Pixel) mit der Farbe `farbe` an der Position (`x,y`).

`getpixel(int x, int y)`

liefert die Farbe (als `int`-Wert) des Pixels an der Position (`x,y`).

Beispiel:
Dieses Beispiel stammt aus dem 11. Informatik-Wettbewerb. Das hier verwendete Modell gibt stark vereinfacht eine Vorstellung davon, wie Öl in das Erdreich einsickert. Dazu stellt man sich einen vertikalen Schnitt durch den Erdboden vor und verwendet dafür ein rechteckiges Feld mit ganzzahligen Koordinaten. Die y-Achse des Koordinatensystems sei nach unten orientiert, die x-Achse nach rechts.
Über die anfängliche Verteilung des Öls nimmt man folgendes an: Das Öl befindet sich in den mittleren beiden Vierteln der oberen Erdschicht, d. h. an Positionen mit den Koordinaten (x,y), für die folgendes gilt:
$\frac{1}{4} \cdot Feldbreite <= x <= \frac{3}{4} \cdot Feldbreite$ und $y = 1$
Zusätzlich sei `x` ungerade, also nur jedes zweite Pixel.
Über das Eindringen des Öls in den Erdboden machen wir folgende Annahmen: Befindet sich Flüssigkeit an Position (x,y), so dringt sie mit der Wahrscheinlichkeit p (0 < p <= 1) in die nächsttiefere Schicht zur Position (x-1,y+1) und unabhängig davon auch mit der Wahrscheinlichkeit p zur Position (x+1,y+1) vor. Dabei kann p als Maß für die Bodenbeschaffenheit gedeutet werden. Befindet sich an Position (x,y) keine Flüssigkeit, gelangt von dieser Stelle her auch keine Flüssigkeit mehr in die nächsttiefere Erdschicht. Folgendes C-Programm `oel.c` löst diese Aufgabenstellung:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <graphics.h>

int  main( int argc, char *argv[] ) {
   float  p;  /* Einzulesende Wahrscheinlichkeit */
   int grenze, x, y, x_max, y_max, farbe,
       noch_oel=1; /* Boole'sche Var.: 1, solange Oel da, sonst dann 0 */
    srand(time(NULL)); /* Zufallszahlen-Generator initialisieren */
      /*--- Graphik einschalten */
    initgraph( 640, 480 );
      /*--- Wahrscheinlichkeit fuer Oel-Eindringen einlesen */
    p = getdouble("Wahrscheinlichkeit fuer Eindringen des Oels ? " );
    grenze = (int)(RAND_MAX*p);
      /*--- Graphikbildschirm loeschen und maximal moegl. Koordinaten ermitteln */
    cleardevice( LIGHTGRAY );
    x_max = getmaxx(); y_max = getmaxy();
```

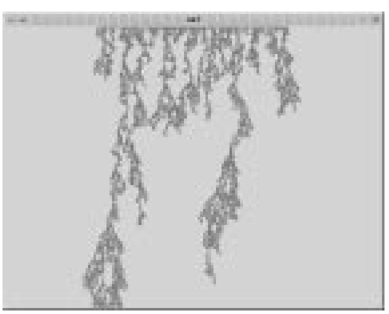
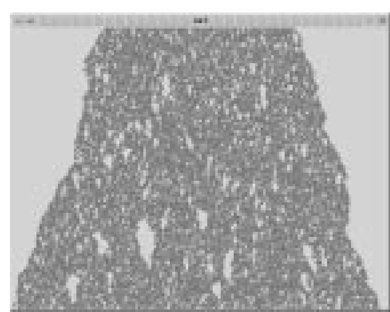



Abbildung 21.3: Wahrscheinlichkeit von 0.62 (links) und 0.7 (rechts)

```
      /*--- Eindringen des Oels am Bildschirm simulieren */
    for (x=x_max/4 ; x<=x_max*3/4 ; x++)
       if ((x+1)%2==0)
          putpixel(x,1,BLUE);
    for (y=1 ; y<y_max-1 && noch_oel ; y++) {
       noch_oel=0;
       for (x=0 ; x<x_max-1 ; x++) {
          farbe = getpixel(x,y);
          if (farbe==BLUE || farbe==RED) {
             noch_oel=1;
             if (x-1>=0 && rand()<=grenze)
                putpixel(x-1,y+1,BLUE);
             if (x+1<=x_max && rand()<=grenze)
                putpixel(x+1,y+1,RED);
          }
       }
       if ( kbhit() )
          break;
    }
    getch();
    closegraph();
    return(0);
}
```

Abbildung 21.3[3] zeigt das Einsickern von Öl für die Wahrscheinlichkeiten von `0.62` und `0.7`.

Beispiel:

Das folgende Programm `pixel.c` wählt zufällig eine Hintergrundfarbe und malt dann 1 000 kleine Vierecke an zufällige Bildschirmpositionen mit zufällig gewählten Farben. Ein Viereck sind dabei 9 Pixel:

```
xxx
xox
xxx
```

[3]Beim Zeigen von Graphik-Bildschirmausgaben sind die Farben, wie sie wirklich am Bildschirm erscheinen, hier nicht erkennbar

wobei nur der Mittelpunkt (o) zufällig gewählt ist. Danach wird der Inhalt dieses Fensters gelöscht, mit einer zufälligen Hintergrundfarbe gefüllt und wieder mit neuen zufälligen kleinen Rechtecken bemalt. Mit einem Tastendruck kann dieses Programm beendet werden.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <graphics.h>

#define PIXEL_ZAHL    1000

int  main( int argc, char *argv[] )
{
   int i,
       maxx, maxy,
       x, y;
   int farbe,
       max_farbe;

    srand(time(NULL));

    initgraph( 640, 480 );
       /* Maximal moegliche Koordinaten */
    maxx=getmaxx();
    maxy=getmaxy();
       /* Maximale Farbennummer */
    max_farbe = getmaxcolor();
       /* Zufaellige Hintergrund-Farben und Setzen von farbigen Pixeln */
    while ( !kbhit()) {
       cleardevice( rand()%(max_farbe+1) );
       for (i=1 ; i<=PIXEL_ZAHL ; i++) {
          x = rand()%maxx;
          y = rand()%maxy;
          farbe = rand()%(max_farbe+1);
             /* Aus Pixel ein kleines Viereck malen */
          putpixel(x, y, farbe);
          putpixel(x-1,y-1,farbe);
          putpixel(x,y-1,farbe);
          putpixel(x+1,y-1,farbe);
          putpixel(x+1,y,farbe);
          putpixel(x+1,y+1,farbe);
          putpixel(x,y+1,farbe);
          putpixel(x-1,y+1,farbe);
          putpixel(x-1,y,farbe);
       }
    }
    closegraph();
    return 0;
}
```

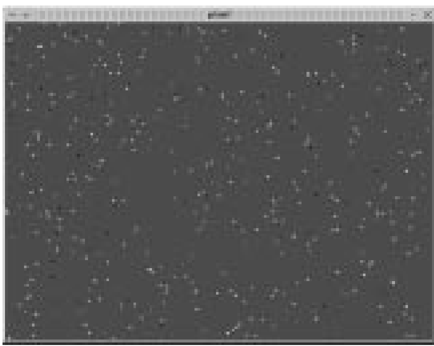
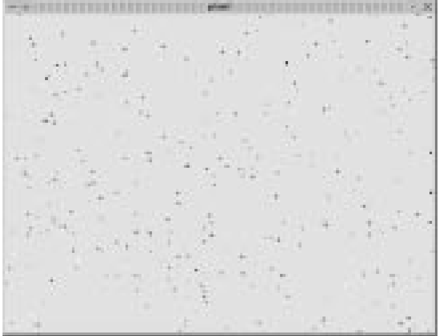
 

Abbildung 21.4: Ständige Ausgabe von kleinen Vierecken auf wechselnden Hintergrund

Abbildung 21.4 zeigt die Inhalte des von Programm `pixel.c` eingeblendeten Fensters zu gewissen Zeitpunkten.

21.7 Positionieren, Linien zeichnen und Farbe einstellen

`setcolor(int farbe)`

legt die Zeichenfarbe für Textausgaben und das Malen von Linien, Kreisen, Vierecken usw. auf die Farbe `farbe` fest.

`getcolor()`

liefert die aktuell eingestellte Zeichenfarbe für Textausgaben und das Malen von Linien, Kreisen, Vierecken usw. (als `int`-Wert).

`moverel(int dx, int dy)`

bewegt den nicht sichtbaren Graphikcursor um eine Distanz (`dx,dy`) relativ zu seiner momentanen Position, ohne dabei zu zeichnen.

`moveto(int x, int y)`

positioniert den Graphikcursor auf den Punkt (`x,y`).

`line(int x1, int y1, int x2, int y2)`

zeichnet eine Linie von (`x1,y1`) nach (`x2,y2`). Dabei werden die aktuelle Farbe, Linienart und Linienbreite verwendet.

`linerel(int dx, int dy)`

zeichnet eine Linie relativ (`dx,dy`) zur momentanen Position des Graphikcursors. Es werden die aktuelle Farbe, Linienart und Linienbreite verwendet.

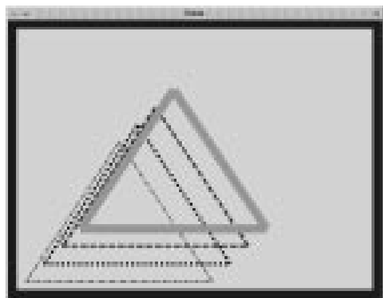


Abbildung 21.5: Anzeige des Programms `linie.c` nach vier Tastendrücken

`lineto(int x, int y)`

zeichnet eine Linie von der momentanen Position des Graphikcursors zu dem angegebenen absoluten Punkt (`x,y`). Es werden die aktuelle Farbe, Linienart und Linienbreite verwendet.

`setlinestyle(int linestyle, int dicke)`

setzt die Linienart und -dicke für folgende Zeichenaktionen. Die hier gesetzte Linienart wird von linienzeichnenden Graphikfunktionen, wie z. B. `line()`, `linerel()`, `lineto()`, `rectangle()`, `drawpoly()`, `arc()`, `circle()`, `ellipse()` oder `pieslice()` verwendet.

Für `linestyle` ist einer der folgenden Namen anzugeben:

Name	Wert	Art der Linie
NO_LINE	0	keine
SOLID_LINE	1	durchgezogen
DASHED_LINE	2	gestrichelt
DOTTED_LINE	3	gepunktet
DASH_DOT_LINE	4	Strich Punkt Strich Punkt...
DASH_DOT_DOT_LINE	5	Strich Punkt Punkt Strich Punkt Punkt Strich...
MAXLINESTYLE	5	

Beispiel:

Das folgende Programm `linie.c` zeichnet Dreiecke in die linke untere Ecke des Bildschirms. Die einzelnen Dreiecke sind dabei immer um etwas nach rechts oben versetzt. Abbildung 21.5 zeigt den Inhalt des von Programm `linie.c` eingeblendeten Fensters, nachdem viermal eine beliebige Taste gedrückt wurde.
Das Programm `linie.c`:

```
#include <graphics.h>

#define OFFSET   30

int  main( int argc, char *argv[] ) {
```

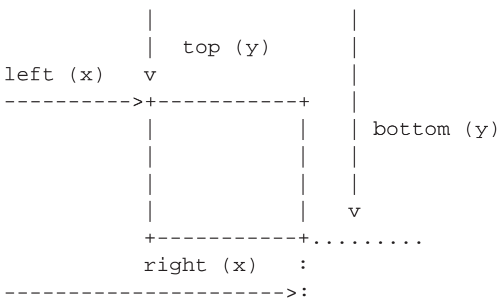
```
int maxx, maxy;
 initgraph( 640, 480 );
    /* Maximal moegliche Koordinaten */
 maxx = getmaxx();
 maxy = getmaxy();
    /*--- Bildschirm loeschen ---*/
 cleardevice( LIGHTGRAY );
    /*--- Blauen dicken Rahmen um Bildschirm zeichnen ---*/
 setcolor( BLUE );
 setlinestyle( SOLID_LINE, 30 );
 line( 0, 0, maxx, 0 );
 line( maxx, 0, maxx, maxy );
 line( maxx, maxy, 0, maxy );
 line( 0, maxy, 0, 0 );
 getch();
    /*--- Rotes Dreieck an linken unteren Bildschirm ---*/
 setcolor( RED );
 setlinestyle( DASH_DOT_LINE, 3 );
 moveto( OFFSET, maxy-OFFSET );
 lineto( maxx/2+OFFSET, maxy-OFFSET );
 linerel( -maxx/4,-maxy/2 );
 linerel( -maxx/4,maxy/2 );
 getch();
    /*--- Schwarzes Dreieck mit dicker gepunkteter Linie etwas nach ---*/
    /*--- rechts oben versetzt                                      ---*/
 setcolor( BLACK );
 setlinestyle( DOTTED_LINE, 5 );
 moverel( +OFFSET, -OFFSET );
 linerel( maxx/2, 0 );
 linerel( -maxx/4, -maxy/2 );
 linerel( -maxx/4, maxy/2 );
 getch();
    /*--- Blaues Dreieck mit gestrichelter Linie nach rechts oben versetzt */
 setcolor( BLUE );
 setlinestyle( DASHED_LINE, 4 );
 moverel( +OFFSET, -OFFSET );
 linerel( maxx/2, 0 );
 linerel( -maxx/4, -maxy/2 );
 linerel( -maxx/4, maxy/2 );
 getch();
    /*--- Gruenes Dreieck mit durchgezogener Linie nach rechts oben -*/
 setcolor( GREEN );
 setlinestyle( SOLID_LINE, 15 );
 moverel( +OFFSET, -OFFSET );
 linerel( maxx/2, 0 );
 linerel( -maxx/4, -maxy/2 );
 linerel( -maxx/4, maxy/2 );
 getch(); getch();
 closegraph();
```

```
    return(0);
}
```

21.8 Figuren zeichnen und ausfüllen

`rectangle(int left, int top, int right, int bottom)`

zeichnet ein Rechteck, wobei die aktuellen Einstellungen für Linienart, -stärke und -farbe benutzt werden; siehe auch `setcolor()` und `setlinestyle()`. Die linke obere Ecke ist dabei durch die Koordinaten (`left,top`) und die rechte untere Ecke durch (`right,bottom`) in Pixeln festgelegt:

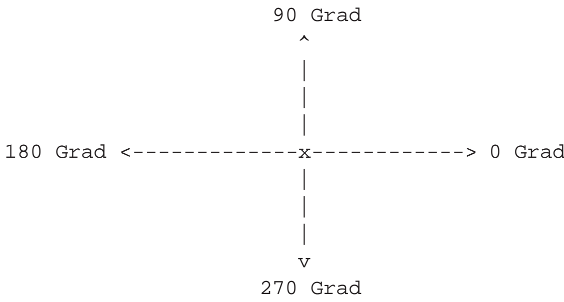


`circle(int x, int y, int radius)`

zeichnet in der aktuellen Zeichenfarbe (siehe `setcolor()`) einen Kreis um den Mittelpunkt (`x,y`) mit dem Radius `radius`.

`arc(int x, int y, int start, int end, int radius)`

zeichnet in der aktuellen Zeichenfarbe (siehe `setcolor()`) einen Kreisbogen. (`x,y`) ist der Mittelpunkt. Die Winkel `start` und `end` legen Start- und Endpunkt des mit Radius `radius` gezogenen Kreisbogens fest. Die Angabe beider Winkel erfolgt in Grad, wobei entgegen dem Uhrzeigersinn gezählt wird:



`ellipse(int x, int y, int start, int end, int xradius, int yradius)`

zeichnet einen elliptischen Kreisausschnitt um den Mittelpunkt (x,y) mit den Radien xradius (horizontal) und yradius (vertikal). start und end legen dabei den Start- und Endpunkt des elliptischen Bogens fest. Hat start den Wert 0 und end den Wert 360, wird eine vollständige Ellipse gezeichnet. Die Angabe beider Winkel erfolgt in Grad, wobei entgegen dem Uhrzeigersinn gezählt wird; siehe auch arc().

`setfillstyle(int pattern, int farbe)`

setzt das Muster (pattern) und die dabei zu verwendende farbe für Flächenfüllungen. Für pattern ist einer der folgenden Namen anzugeben:

Name	Wert	Füllmuster
EMPTY_FILL	0	Hintergrundfarbe
SOLID_FILL	1	angegebene farbe
DENSE1_FILL	2	sehr stark ausgefüllt
DENSE2_FILL	3	‖
DENSE3_FILL	4	‖
DENSE4_FILL	5	‖
DENSE5_FILL	6	‖
DENSE6_FILL	7	⇓
DENSE7_FILL	8	sehr schwach ausgefüllt
HORLINE_FILL	9	horizontale Linien
VERLINE_FILL	10	vertikale Linien
CROSS_FILL	11	horizontale und vertikale Linien
LDIAG_FILL	12	//////////
RDIAG_FILL	13	\\\\\\\\\\
CROSSDIAG_FILL	14	beides (LDIAG_FILL und RDIAG_FILL)
MAXFILLSTYLE	14	

`fillellipse(int x, int y, int xradius, int yradius)`

zeichnet eine ausgefüllte Ellipse mit (x,y) als Mittelpunkt; xradius wird dabei als horizontaler und yradius als vertikaler Radius verwendet. Die Ellipse selbst wird mit dem aktuellen Füllmuster und der aktuellen Füllfarbe gefüllt; siehe auch setfillstyle().

`bar(int left, int top, int right, int bottom)`

zeichnet einen zweidimensionalen Balken. Wie bei rectangle() wird die linke obere Ecke durch die Koordinaten (left,top) und die rechte untere Ecke durch (right,bottom) in Pixeln festgelegt. Es wird dabei die aktuelle Füllfarbe und das aktuelle Füllmuster (siehe setfillstyle()) verwendet. Einen Umriss zeichnet bar() nicht; dazu müßte bar3d() mit depth=0 verwendet werden.

`bar3d(int left, int top, int right, int bottom, int depth)`

zeichnet einen dreidimensionalen Balken. Ein dreidimensionaler Balken besteht aus einem gefüllten Rechteck, bei dem – wie bei rectangle() – die linke obere Ecke durch die Koordinaten (left,top) und die rechte untere Ecke

durch (`right,bottom`) in Pixeln festgelegt wird. `bar3d()` zeichnet zuerst die Umrisslinien in der aktuellen Zeichenfarbe (siehe `setcolor()`) und der aktuellen Linienart (siehe `setlinestyle()`). Danach füllt es die umschlossene Fläche mit dem aktuellen Füllmuster; siehe auch `setfillstyle()`. Über `depth` (zu deutsch: Tiefe) läßt sich die räumliche Tiefe des dreidimensionalen Balkens (in Pixel) festlegen. Eine Faustregel besagt dabei, dass die Tiefe ungefähr 25% der Breite betragen sollte. Bei `depth=0` wird nur ein zweidimensionaler Balken mit einer Umrisslinie gezeichnet.

`drawpoly(int numpoints, int *polypoints)`

zeichnet den Umriss eines Polygons mit `numpoints` Eckpunkten in der aktuellen Linienart und -farbe. `numpoints` gibt die Anzahl der Eckpunkte an. `polypoints` muss der Name des Arrays sein, das fortlaufende Koordinatenpaare enthalten muss. Zum Zeichnen einer geschlossenen Figur mit *n* Eckpunkten muss `polypoints` *n+1* Koordinatenpaare enthalten, wobei das letzte Paar dieselben Werte wie das erste Paar hat; siehe auch nachfolgende Beispiele.

`fillpoly(int numpoints, int *polypoints)`

zeichnet ein ausgefülltes Polygon mit `numpoints` Eckpunkten in der aktuellen Linienart und -farbe. Danach wird dieses Polygon mit der aktuellen Füllfarbe und dem aktuellen Füllmuster (siehe `setfillstyle()`) gefüllt. Wie bei `drawpoly()` gilt: `numpoints` gibt die Anzahl der Eckpunkte an. `polypoints` muss der Name des Arrays sein, das fortlaufende Koordinatenpaare enthalten muss. Zum Zeichnen einer geschlossenen Figur mit *n* Eckpunkten muss `polypoints` *n+1* Koordinatenpaare enthalten, wobei das letzte Paar dieselben Werte wie das erste Paar hat; siehe auch nachfolgende Beispiele.

Beispiele:
Das folgende Programm `figur.c` malt eine Person mit Tisch auf den Bildschirm. Die Farben und Muster in den jeweiligen Symbolen sind dabei zufällig ausgewählt. Bei einem Tastendruck wird immer ein neues Bild gemalt. Das Programm kann mit Drücken der (ESC)-Taste verlassen werden. Mögliche durch `figur.c` gezeigte Bilder könnten z. B. aussehen, wie sie in Abbildung 21.6 gezeigt sind.
Das Programm `figur.c`:

```
#include <graphics.h>
#include <time.h>
#include <stdlib.h>

#define ZUFALLSFARBE \
       farbe = rand()%(max_farbe+1);   /* Zufaellige Zeichenfarbe */ \
       setcolor(farbe);                /* setzen                  */

#define ZUFALLS_FUELLMUSTER \
       muster = rand()%MAXLINESTYLE;   /* Zufaelliges Fuellmuster */ \
       hfarbe = rand()%(max_farbe+1);  /* Zufaellige Fuellfarbe   */ \
```

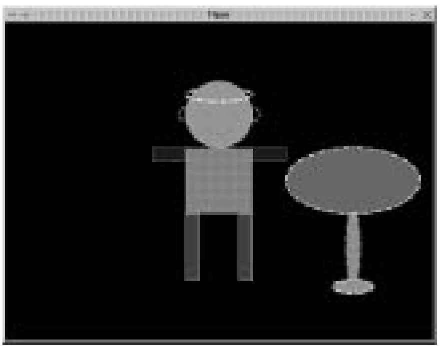
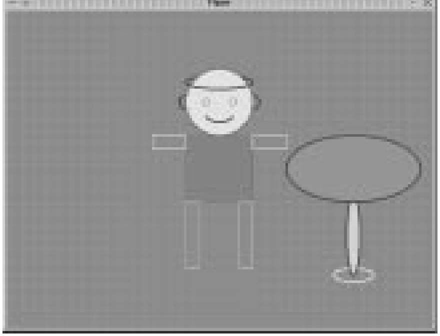



Abbildung 21.6: Unterschiedliche Anzeigen des Programms `figur.c` nach einem Tastendruck

```
      setfillstyle(muster,hfarbe);    /* setzen */

int  main( int argc, char *argv[] ) {
   int maxx, maxy,
       muster, lmuster,
       farbe, hfarbe,
       mittex, mittey,
       max_farbe, poly[10];

    srand(time(NULL));  /* Zufallsgenerator initialisieren */

    initgraph( 640, 480 );

    maxx = getmaxx();  /* Maximal moegliche Koordinaten */
    maxy = getmaxy();
    mittex = maxx/2;   /* Bildschirm-Mittelpunkt */
    mittey = maxy/2;
    max_farbe = getmaxcolor();      /* Maximale Farbennummer */

    while (1) {  /* Figur malen */
          /*------ Bildschirm-Rahmen und -Hintergrund -------------------*/
       cleardevice( BLACK );
       setlinestyle( SOLID_LINE, 1 );
       ZUFALLSFARBE; ZUFALLS_FUELLMUSTER;
       poly[0] = poly[1] = poly[3] = poly[6] = poly[8] = poly[9] = 0;
       poly[2] = poly[4] = maxx;
       poly[5] = poly[7] = maxy;
       fillpoly( 5, poly );
          /*------ Bauch ------------------------------------------------*/
       ZUFALLSFARBE; ZUFALLS_FUELLMUSTER;
       bar3d( mittex-50, mittey-50, mittex+50, mittey+50, 0 ); /* Bauch malen */
          /*------ Kopf -------------------------------------------------*/
```

```
        ZUFALLSFARBE;
        ZUFALLS_FUELLMUSTER;
        fillellipse( mittex, mittey-100, 50, 50 );   /* Kopf malen */
           /*------ Beine -------------------------------------------------*/
        ZUFALLSFARBE; ZUFALLS_FUELLMUSTER;
        bar3d( mittex-50, mittey+50, mittex-30, mittey+150, 0 ); /* Linkes Bein */
        bar3d( mittex+30, mittey+50, mittex+50, mittey+150, 0 ); /* Rechtes Bein */
           /*------ Arme --------------------------------------------------*/
        ZUFALLSFARBE; ZUFALLS_FUELLMUSTER;
        bar3d( mittex-100, mittey-50, mittex-50,  mittey-30, 0 ); /* Linker Arm */
        bar3d( mittex+50,  mittey-50, mittex+100, mittey-30, 0 ); /* Rechter Arm */
           /*------ Augen -------------------------------------------------*/
        ZUFALLSFARBE; ZUFALLS_FUELLMUSTER;
        fillellipse( mittex-20, mittey-100, 5, 5 ); /* Linkes Auge */
        fillellipse( mittex+20, mittey-100, 5, 5); /* Rechtes Auge */
           /*------ Ohren ------------------------------------------------*/
        lmuster = rand()%MAXLINESTYLE;   /* Zufaelliges Linienmuster */
        setlinestyle( lmuster, 3 );      /* und dicke Linie setzen */
        arc( mittex-50, mittey-100,  90, 270, 10 ); /* Linkes Ohr */
        arc( mittex+50, mittey-100, 270,  90, 10 ); /* Rechtes Ohr */
          /*------- Mund ------------------------------------------------*/
        arc( mittex, mittey-100, 225, 315, 30 );
           /*------ Heiligenschein ---------------------------------------*/
        ZUFALLSFARBE;
        ellipse( mittex, mittey-130, 135,45, 50,10 );
           /*------ Tisch ------------------------------------------------*/
        ZUFALLSFARBE; ZUFALLS_FUELLMUSTER;
        fillellipse( mittex+200,mittey+160, 30,10 );
        ZUFALLSFARBE; ZUFALLS_FUELLMUSTER;
        fillellipse( mittex+200,mittey+100, 10,60 );
        ZUFALLSFARBE;
        setfillstyle( SOLID_FILL, rand()%getmaxcolor() );
        fillellipse(mittex+200,mittey, 100,50 );

        if ( getch() == Key_Escape ) /* Abbruch bei ESC */
           break;
     }
     closegraph();
     return(0);
}
```

Das folgende Programm `fmuster.c` gibt nacheinander alle Füllmuster mit `bar()` und `bar3d()` am Bildschirm aus, wie es in der Abbildung 21.7 gezeigt ist.

```
#include <graphics.h>

int  main( int argc, char *argv[]) {
   int links, oben, i;
    initgraph( 640, 500 );
    setcolor( RED );
       /* Alle Fuellmuster fuer bar durchlaufen */
```

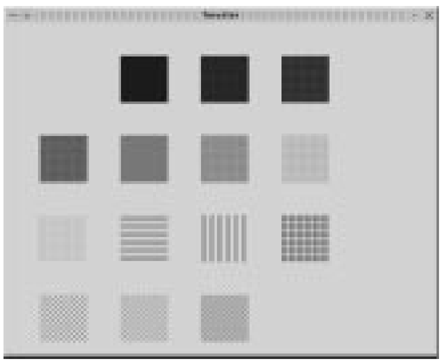
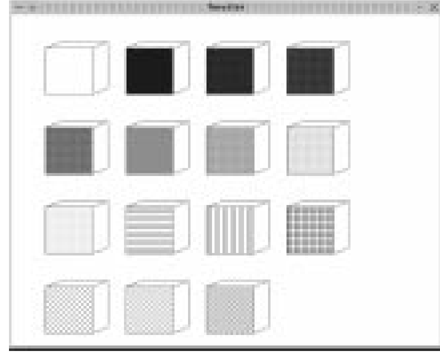



Abbildung 21.7: Unterschiedliche Anzeigen des Programms `fmuster.c` nach einem Tastendruck

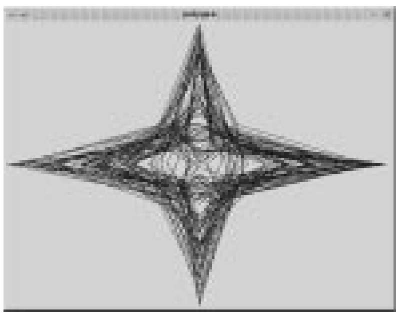
```
    for (i=SOLID_FILL ; i<=MAXFILLSTYLE ; i++) {
       links=i%4;
       oben=i/4;
       setfillstyle( i, BLUE );
       bar(50+links*120, 50+oben*120, 120+links*120, 120+oben*120);
    }
    getch();
    cleardevice( WHITE );
       /* Alle Fuellmuster fuer bar3d durchlaufen */
    for (i=EMPTY_FILL ; i<=MAXFILLSTYLE ; i++) {
       links=i%4;
       oben=i/4;
       setfillstyle( i, BLUE );
       bar3d(50+links*120, 50+oben*120, 120+links*120, 120+oben*120, 25 );
    }
    getch();
    closegraph();
    return 0;
}
```

Das folgende Programm `polygon.c` gibt „Stern-Polygone“ nacheinander am Bildschirm aus. Die einzelnen Polygone, deren Eckpunkte in einem gewissen Bereich zufällig gewählt werden, haben immer den Fenster-Mittelpunkt als Mittelpunkt und werden übereinander gezeichnet. Der Benutzer kann dabei am Anfang des Programms angeben, ob die Polygone ausgefüllt sein sollen oder nicht.

```
#include <ctype.h>
#include <stdlib.h>
#include <time.h>
#include <graphics.h>

int main( int argc, char* argv[] ) {
   int ausgefuellt;  /* 1, wenn ausgefuellte Polygone gewuenscht, sonst 0 */
   int mx, my;
```

```
   int poly[18]; /*----- immer eins mehr hier angeben als Elemente
                 ----- wirklich benoetigt werden. */

    printf("Ausgefuellte Polygone (j/n): ");
    ausgefuellt = toupper(getchar())=='J' ? 1 : 0;
    srand(time(NULL)); /* Zufallszahlen-Generator initialisieren */

    initgraph( 640, 480 );
    mx = getmaxx()/2; my = getmaxy()/2;
    setcolor( BLUE );
    while (1) {
       poly[0] = mx+rand()%mx;    poly[1] = my;              /* 1.Eckpunkt */
       poly[2] = mx+20+rand()%50; poly[3] = my-20-rand()%50; /* 2.Eckpunkt */
       poly[4] = mx;              poly[5] = my-rand()%my;    /* 3.Eckpunkt */
       poly[6] = mx-20-rand()%50; poly[7] = my-20-rand()%50; /* 4.Eckpunkt */
       poly[8] = mx-rand()%mx;    poly[9] = my;              /* 5.Eckpunkt */
       poly[10] = mx-20-rand()%50;poly[11] = my+20+rand()%50;/* 6.Eckpunkt */
       poly[12] = mx;             poly[13] = my+rand()%my;   /* 7.Eckpunkt */
       poly[14] = mx+20+rand()%50;poly[15] = my+20+rand()%50;/* 8.Eckpunkt */
          /* Da drawpoly das Polygon nicht automatisch schließt, muss als */
          /* Endpunkt wieder der Anfangspunkt angegeben werden.           */
       poly[16] = poly[0]; poly[17] = poly[1];
          /* Polygon zeichnen */
       if (ausgefuellt) {
          setfillstyle(SOLID_FILL, rand()%getmaxcolor());
          fillpoly(9, poly);
       } else
          drawpoly(9, poly);
       if (getch()==Key_Escape) /* Abbruch mit ESC */
          break;
    }
    closegraph();
    return 0;
}
```

Mögliche durch `polygon.c` ausgegebene Bilder nach x-maligen Tastendruck zeigt Abbildung 21.8.



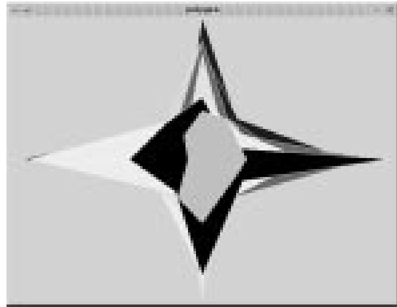


Abbildung 21.8: Anzeige des Programms `polygon.c` nach x-maligen Tastendruck (links: nicht ausgefüllt, rechts: ausgefüllt)

Kapitel 25

Zeiger und Arrays

Was man auch redet, schreibt und funkt:
Unheilbar bleibt der wunde Punkt.
Punktion, Eugen Roth

In der Programmierpraxis sind häufig Probleme anzutreffen, bei denen man nicht mit einfachen Variablen auskommt, sondern einen ganzen Block von Variablen des gleichen Typs benötigt. Ein Beispiel dafür wären die Monatumsätze eines Unternehmens für ein Jahr. In diesem Fall ist es sehr nützlich, 12 `double`-Variablen in einem Block zu definieren, an dem nur ein Name vergeben wird. Zur Unterscheidung der Monate wird dann z. B. ein Index (ganze Zahl) verwendet.

25.1 Eindimensionale Arrays

Ein *Array* ist die namentliche Zusammenfassung einer Anzahl von gleichartigen Objekten eines Datentyps, wie z. B. `int`- oder `char`-Variablen. Anstelle des Begriffes Array wird manchmal auch der Begriff *Vektor* verwendet. Hier werden wir immer den Begriff Array benutzen.

25.1.1 Eindimensionale Arrays

Ein eindimensionales Array mit 26 Elementen vom Typ `char` könnte wie folgt vereinbart werden:

```
char buchst[26];
```

Mit dieser Angabe wird ein Array `buchst` mit 26 Elementen vom Typ `char` definiert. Die Anzahl der Elemente wird bei der Definition in eckigen Klammern angegeben. Unter

```
char buchst[26];
```

können wir uns die Reservierung eines Speicherblocks mit Namen `buchst` und mit 26 aufeinanderfolgenden Elementen (hier also 26 `char`-Variablen) vorstellen, wie dies auch in Abbildung 25.1 gezeigt ist. Mit der Definition

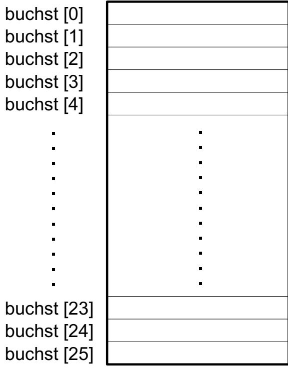


Abbildung 25.1: `char`-Array `buchst` mit 26 Speicherplätzen

```
char buchst[26];
```

werden also, wenn man so will, auf einmal 26 `char`-Variablen (`buchst[0]`, `buchst[1]`, `buchst[2]`, `...`, `buchst[25]`) festgelegt.
Werden n Elemente (hier: $n = 26$) für ein Array definiert, so erfolgt die Adressierung über so genannte Indizes von Element 0 bis Element $n - 1$.
Beachten Sie, dass jedes Array mit der Elementnummer 0 und nicht mit 1 beginnt!
Mit den Anweisungen

```
buchst[3] ='a';
buchst[0] ='!';
buchst[24] ='H';
```

ergäbe sich dann ein Speicherbild, wie es in Abbildung 25.2 gezeigt ist.
Beispiele:

`int tage[32];`
definiert ein Array `tage` mit 32 `int`-Variablen `tage[0], ..., tage[31]`.

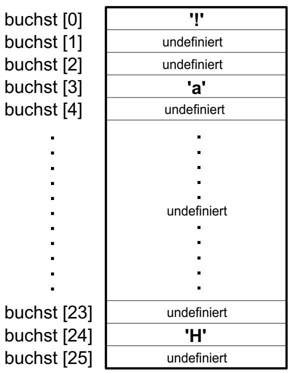


Abbildung 25.2: `char`-Array `buchst` mit 26 Speicherplätzen nach einigen Zuweisungen

`int *zeig[20];`
: definiert ein Array `zeig` mit 20 Zeigern `zeig[0]`, ...,`zeig[19]` auf `int`-Variablen.

Beispiel:

Wir wollen nun ein C-Programm `buchstnr.c` erstellen, das zunächst in einem Array `buchst` alle Kleinbuchstaben speichert. Danach kann der Benutzer eine Nummer eingeben, zu der ihm der entsprechende Kleinbuchstabe am Bildschirm ausgegeben wird. Unser Programm soll zur Ermittlung des Buchstabens auf das Array `buchst` zugreifen.

```
#include  <stdio.h>

int  main(void) {
   char   buchst[26];
   int    i,  zahl;
    for (i=0 ; i<26 ; i++)  /* Array mit Kleinbuchstaben belegen */
        buchst[i] = 'a'+i;
    while (1) {
       printf("Geben Sie eine Zahl zw. 1 und 26 ein (Ende=Zahl 100): ");
       scanf("%d", &zahl);
       if (zahl==100)
          break;
       else if (zahl<1 || zahl>26)
          printf("     .......Falsche Eingabe (Eingabe wiederholen!)....\n");
       else
          printf(" ---> %d. Kleinbuchstabe = %c\n\n", zahl, buchst[zahl-1]);
    }
    printf("------Programmende------\n");
    return(0);
}
```

In der `for`-Schleife

```
   for (i=0 ; i<26 ; i++)  /* Array mit Kleinbuchstaben belegen */
       buchst[i] = 'a'+i;
```

werden im Array `buchst` die Kleinbuchstaben abgespeichert.
Da der erste Buchstabe `a` in `buchst[0]`, der zweite Buchstabe `b` in `buchst[1]`, ... und der 26. Buchstabe `z` in `buchst[25]` liegt, müssen wir beim Zugriff auf das Feld `buchst` als Index die eingegebene ***zahl minus 1*** verwenden, um den richtigen Buchstaben zu erhalten. Deshalb wurde auch bei

```
   printf(" ---> %d. Kleinbuchstabe = %c\n\n", zahl, buchst[zahl-1]);
```

nicht `buchst[zahl]`, sondern `buchst[zahl-1]` angegeben.
Nachfolgend nun noch ein weiteres Beispiel zu eindimensionalen Arrays.

Beispiel:

Der Algorithmus, um Zahlen aus dem Zehnersystem in ein anderes System umzuwandeln, ist folgender:

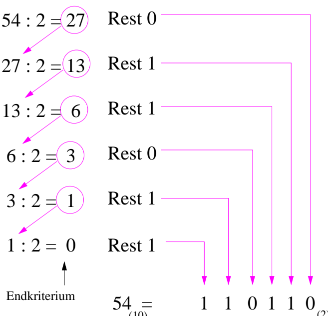


Abbildung 25.3: Umwandlung der Zahl 54 in das Dualsystem

1. Man muss ständig durch die Basis des Zielsystems dividieren und sich den Rest merken.
2. Bei erneuter Division wird das Ergebnis der vorausgehenden Division zum Dividenden.
3. Die Umwandlung ist beendet, wenn bei der Division der Quotient 0 lautet.
4. Die Zahl des Zielsystems ergibt sich dann, wenn man sich die gemerkten Reste in umgekehrter Reihenfolge (von unten nach oben) hinschreibt.

Dieser Algorithmus soll nochmals anhand eines Beispiels (Umwandlung der Zahl 54 ins Dualsystem) – wie es in Abbildung 25.3 gezeigt ist – verdeutlicht werden.
Wir wollen nun ein C-Programm `dezumwan.c` erstellen, das zunächst das Zielsystem anfordert und dann nach der umzuwandelnden Zahl aus dem Zehnersystem fragt. Nach diesen Benutzereingaben soll unser Programm die entsprechende Zahl im Zielsystem ermitteln. Für die Basis `B` des Zielsystems soll gelten: `B<=10`.
Mögliches Aussehen des C-Programms `dezumwan.c`:

```
#include  <stdio.h>

int  main(void) {
    int  zahl, basis, zaehler=0, i;
    int  ziel[100];

    while (1) {
       printf("Gib Basis des Zielsystem ein (2<=Basis<=10): ");
       scanf("%d", &basis);
       if (basis>=2 && basis<=10)
          break;
    }
    printf("Gib die zu wandelnde Zahl aus dem Zehnersystem ein: ");
    scanf("%d", &zahl);
    printf("     ---> %d(10) = ", zahl);
    while (zahl>0)  {
       ziel[zaehler] = zahl % basis;
       zahl /= basis;
       ++zaehler;
```

```
    }
    for (i=zaehler-1 ; i>=0 ; i--)
       printf("%d", ziel[i]);
    printf("(%d)\n",basis);
    return(0);
}
```

Mögliche Abläufe des Programms `dezumwan.c`:

```
Gib Basis des Zielsystem ein (2<=Basis<=10): 2 ↵
Gib die zu wandelnde Zahl aus dem Zehnersystem ein: 25 ↵
     ---> 25(10) = 11001(2)
```

```
Gib Basis des Zielsystem ein (2<=Basis<=10): 5 ↵
Gib die zu wandelnde Zahl aus dem Zehnersystem ein: 2536 ↵
     ---> 2536(10) = 40121(5)
```

Das angegebene C-Programm ist nur lauffähig mit eingegebenen positiven Startzahlen (>0). Da in diesem Beispiel nicht von vornherein klar ist, wieviele Stellen die umgewandelte Zahl benötigt, wäre die Programmierung dieser Aufgabe ohne Verwendung des Arrays `ziel`, also nur mit einfachen Variablen, sehr umständlich. Die Ausgabe der umgewandelten Zahl erfolgt in umgekehrter Reihenfolge, d. h., die durch den Umwandlungsalgorithmus zuerst gefundenen Ziffern sind zuletzt auszugeben. Dies macht die Speicherung in einem Array notwendig. Zur Ausgabe der gespeicherten Ziffern wird, mit dem höchsten Index beginnend, das Array „rückwärts“ durchlaufen.
Wurde z. B. für die Basis der Wert `5` und für die zu wandelnde Zahl `2536` angegeben, so hat das Array `ziel`, nachdem der Umwandlungsalgorithmus ausgeführt wurde, die in Abbildung 25.4 gezeigte Belegung.
Um die konvertierte Zahl richtig am Bildschirm auszugeben, müssen wir also zuerst `ziel[4]`, dann `ziel[3]` usw., bis `ziel[0]` ausgeben. Diese Ausgabe wird durch folgende `for`-Schleife vorgenommen:

```
   for (i=zaehler-1 ; i>=0 ; i--)
      printf("%d", ziel[i]);
```

Der Startwert für den rückwärtslaufenden Index `i` ist `zaehler-1`, da `zaehler` (bedingt durch den Umwandlungsalgorithmus) um 1 zu hoch gezählt wurde. So hat z. B. in unserem Beispiel `zaehler` den Wert 5.

ziel[0]	1
ziel[1]	2
ziel[2]	1
ziel[3]	0
ziel[4]	4

Abbildung 25.4: Belegung des Arrays `ziel` bei Umwandlung der Zahl 2536 in das Zahlensystem zur Basis `5`

25.1.2 Nur statische Arrays erlaubt (in C89)

In C89 sind nur statische Arrays erlaubt. Dies bedeutet, dass bei der Definition eines Arrays in C89 die obere Grenze immer eine Konstante sein muss:

```
datentyp   arrayname[ konstante ];
```

Es gibt natürlich viele Anwendungsfälle, in denen man sich dynamische Arrays wünschen würde, wie z. B.:

```
int zahlen[n];
```

wobei die Variable `n` erst zur Programmlaufzeit auf einen bestimmten Wert gesetzt wird. Solche Konstruktionen sind jedoch mit gewissen Einschränkungen nur in C99 erlaubt; siehe dazu auch Kapitel 25.6 auf Seite 622.
Wir werden jedoch in Kapitel 27.2.2 auf Seite 667 sehen, wie man auch in C89 dynamische Arrays nachbilden kann.

25.1.3 Von Arrays belegter Speicherplatz

Die Größe des Speicherplatzes, den ein Array belegt, läßt sich leicht berechnen:

```
Anzahl der Bytes eines Elements * Array-Länge
```

So wird z. B. durch die folgende Array-Definition

```
int monat[12];
```

ein Speicherplatz von

```
48 Bytes (12 * 4 Bytes für int)
```

belegt. Wenn der Datentyp `double` durch 8 Bytes realisiert ist, so belegt die folgende Definition:

```
double werte[1000];
```

einen Speicherplatz von

```
8000 Bytes (1000 * 8 Bytes für double)
```

Die Größe des durch ein Array belegten Speicherplatzes läßt sich auch unter Verwendung des `sizeof`-Operators ermitteln. Das nachfolgende Programm `arrygros.c` verdeutlicht dies, indem es den `sizeof`-Operator sowohl auf den Arraynamen als auch auf den Datentyp selbst (wie `sizeof(char[100])`) anwendet. Beide Operationen liefern natürlich dasselbe Ergebnis:

```
#include  <stdio.h>

#define GROS   100

int  main(void) {
  char          c[GROS];
  short         s[GROS];
  int           i[GROS];
  long          l[GROS];
  long long    ll[GROS];
  float         f[GROS];
  double        d[GROS];
```

```
   long double  ld[GROS];

   printf(" c[%d] = %d Bytes (%d)\n", GROS, sizeof(c), sizeof(char[GROS]));
   printf(" s[%d] = %d Bytes (%d)\n", GROS, sizeof(s), sizeof(short[GROS]));
   printf(" i[%d] = %d Bytes (%d)\n", GROS, sizeof(i), sizeof(int[GROS]));
   printf(" l[%d] = %d Bytes (%d)\n", GROS, sizeof(l), sizeof(long[GROS]));
   printf("ll[%d] = %d Bytes (%d)\n", GROS, sizeof(ll), sizeof(long long[GROS]));
   printf(" f[%d] = %d Bytes (%d)\n", GROS, sizeof(f), sizeof(float[GROS]));
   printf(" d[%d] = %d Bytes (%d)\n", GROS, sizeof(d), sizeof(double[GROS]));
   printf("ld[%d] = %d Bytes (%d)\n", GROS, sizeof(ld), sizeof(long double[GROS]));
   return 0;
}
```

Mögliche Ausgabe durch das Programm `arrygros.c`:

```
 c[100] = 100 Bytes (100)
 s[100] = 200 Bytes (200)
 i[100] = 400 Bytes (400)
 l[100] = 400 Bytes (400)
ll[100] = 800 Bytes (800)
 f[100] = 400 Bytes (400)
 d[100] = 800 Bytes (800)
ld[100] = 1200 Bytes (1200)
```

25.1.4 Fallgruben

Array-Indizes erstrecken sich von 0 bis *n*-1 (nicht von 1 bis *n*)

Ein sehr häufiger Fehler in C ist, dass man ein Array der Größe `n` definiert, und dann mit einer `for`-Schleife der folgenden Art dieses Array beschreibt:

```
   for (i=1; i<=n ; i++)
      array[i] = ...;
```

Der Fehler hierbei ist, dass man annimmt, dass das Array sich über die Elemente `1..n` erstreckt. In Wirklichkeit erstreckt es sich aber über die Elemente `0..n-1`, was dazu führt, dass man beim letzten Durchlauf der `for`-Schleife (`i` ist dann `n`) mit

```
array[n] = ...;
```

bereits in fremden Speicherplatz schreibt wie dies das nachfolgende Programm `arryfehl.c` verdeutlicht:

```
#include  <stdio.h>

int  main(void) {
   int  i;
   int  vor_array=0;
   int  quad[5];
   int  nach_array=0;

   for (i=1; i<=5; i++)
      quad[i] = i*i;
   for (i=1; i<=5; i++)
```

```
      printf("%2d *%2d = %3d\n", i, i, quad[i]);
   printf(" ----> vor_array=%d\n", vor_array);
   printf(" ----> nach_array=%d\n", nach_array);
   return(0);
}
```

Mögliche Ausgabe durch dieses Programm `arryfehl.c`:

```
 1 * 1 =   1
 2 * 2 =   4
 3 * 3 =   9
 4 * 4 =  16
 5 * 5 =  25
 ----> vor_array=25
 ----> nach_array=0
```

An der Ausgabe ist zu erkennen, dass hier durch das Schreiben über die Arraygrenzen hinaus der Inhalt der Variablen `vor_array`[1] überschrieben wird, denn dieser Variablen wurde ursprünglich der Wert 0 zugewiesen, und nun hat sie plötzlich den Wert 25, der aus der Zuweisung (in der `for`-Schleife)

```
      quad[5]=5*5;
```

resultiert. Solches Überschreiben von fremden Speicherplatz kann vor allen Dingen in größeren Programmen zu schwer auffindbaren Fehlern führen. Um eine solche Überschreibung in Anwendungen, bei denen eine Indizierung von `1` bis `n` naheliegender ist, zu vermeiden, läßt man häufig das 0. Element ungenutzt. Man wendet dann oft folgende Technik an:

```
#define MAX_ELEMENTE    konstante

datentyp  array_name[MAX_ELEMENTE+1];
```

Unser Programm `arryfehl.c` könnten wir dann wie folgt umschreiben (`arryok.c`)

```
#include  <stdio.h>

#define MAX_ELEMENTE   5

int  main(void) {
   int  i, vor_array=0;
   int  quad[MAX_ELEMENTE+1];
   int  nach_array=0;
   for (i=1; i<=MAX_ELEMENTE; i++)
      quad[i] = i*i;
   for (i=1; i<=MAX_ELEMENTE; i++)
      printf("%2d *%2d = %3d\n", i, i, quad[i]);
   printf(" ----> vor_array=%d\n", vor_array);
   printf(" ----> nach_array=%d\n", nach_array);
   return(0);
}
```

[1] Variablen wurden umgekehrt zu ihrer Deklaration im Stack abgelegt

Nebeneffekte bei Array-Indizierung

Ein anderer Fehler kann bei der Indizierung von Arrays entstehen, wenn Postfix- oder Präfix-Operatoren verwendet werden. Das folgende Programm `arryfeh2.c` verdeutlicht die dabei möglichen Seiteneffekte:

```
#include  <stdio.h>

int  main(void)
{
   int  i, a[10];

   for (i=0; i<10; i++)
      a[i] = 0;

   i = 2;
   a[i] = i--;

   printf("a[1] = %d, a[2] = %d\n", a[1], a[2]);
   return(0);
}
```

In diesem Programm `arryfeh2.c` ist es keinesfalls festgelegt, in welches Arrayelement der Wert von `i` abgelegt wird. Je nach Compiler kann es hier zu unterschiedlichen Ergebnissen kommen.

1. **Möglichkeit:**
   Der Index wird ausgewertet, bevor die Variable `i` dekrementiert wird, und das Arrayelement `a[2]` erhält den Wert `2`:

   ```
   a[1] = 0, a[2] = 2
   ```

2. **Möglichkeit:**
   Der Index wird erst ausgewertet, nachdem die Variable `i` dekrementiert wurde, und das Arrayelement `a[1]` erhält den Wert `2`:

   ```
   a[1] = 2, a[2] = 0
   ```

Grundsätzlich sollte also auf Anweisungen dieser Art verzichtet werden.

25.1.5 Übungen

Amnestie bei Dodon, dem Märchenkönig

Dodon, der Märchenkönig, nahm bei einem Feldzug `x` Feinde gefangen, die er in `x` Einzelzellen steckte. An seinem Geburtstag sollten einige freigelassen werden, und zwar nach einem ganz eigenartigen Verfahren (vom Hofmathematiker ausgedacht). Dieses Verfahren arbeitet mit mehreren Durchgängen, wobei in jedem Durchgang für jede betroffene Zellentür folgender Zustandswechsel durchgeführt wird:

- Ist entsprechende Zellentür zu diesem Zeitpunkt offen, wird sie geschlossen.
- Ist entsprechende Zellentür zu diesem Zeitpunkt geschlossen, wird sie geöffnet.

Zunächst sind alle Zellentüren geschlossen:

`1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18....`

Bei den nachfolgenden Durchgängen werden offene Zellen fett dargestellt.

1. Im ersten Durchgang ist dann jede Tür von einem Zustandswechsel betroffen, was heißt, da zu diesem Zeitpunkt alle Türen geschlossen sind, dass in diesem Durchgang alle Türen geöffnet (fett hier dargestellt) werden:

   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18...

2. Im zweiten Durchgang ist dann nur jede zweite Tür von einem Zustandswechsel betroffen:

   1 2 **3** 4 **5** 6 **7** 8 **9** 10 **11** 12 **13** 14 **15** 16 **17** 18...

3. Im dritten Durchgang ist dann nur jede dritte Tür von einem Zustandswechsel betroffen:

   1 2 3 4 **5** **6** **7** 8 9 10 **11** **12** **13** 14 15 16 **17** **18...**

Und so geht es im vierten,

1 2 3 **4** **5** **6** **7** **8** 9 10 **11** 12 **13** 14 15 **16** **17** **18...**

fünften,

1 2 3 **4** 5 **6** **7** **8** 9 **10** **11** 12 **13** 14 **15** **16** **17** **18...**

... bis zum `x`.ten Durchgang weiter.
Die Frage ist nun, welche Zellentüren standen offen, als der Geburtstag des Königs anbrach? Erstellen Sie ein C-Programm `dodon.c`, das die Zahl der Zellentüren (`x`) einliest und dann diese Frage beantwortet.

Wachsen von Strichen

Erstellen Sie ein C-Programm `wachsen.c`, das ein zufälliges Wachsen von Strichen vom unteren zum oberen Rand eines eingeblendeten Graphikfenster simuliert. Nach einer gewissen Zeit soll die Farbe sich dabei immer ändern. Wird der obere Fensterrand erreicht, so wird die entsprechende Pixelspalte gelöscht und das Wachsen für diese Spalte beginnt erneut vom unteren Bildschirmrand; siehe auch Abbildung 25.5 und 25.6. Um diese Übung zu lösen, muss man das Kapitel 21 auf Seite 301 zur Graphikprogrammierung gelesen haben.

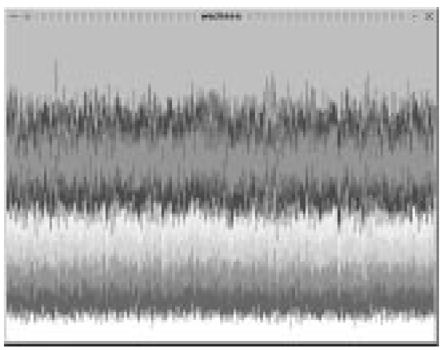


Abbildung 25.5: Wachsen von Strichen (am Anfang und etwas später)

Abbildung 25.6: Wachsen von Strichen (nachdem erste Striche oberen Rand erreichten)

25.2 Mehrdimensionale Arrays

Bisher haben wir nur mit eindimensionalen Arrays gearbeitet. In der Praxis benötigt man aber häufig auch mehrdimensionale Arrays.
Deswegen bietet C neben eindimensionalen Arrays auch noch mehrdimensionale Arrays an, wie z. B. zwei- oder dreidimensionale Arrays.

25.2.1 Zweidimensionale Arrays

In der Mathematik werden zweidimensionale Arrays auch *Matrizen* genannt.
Um uns zweidimensionale Arrays besser vorstellen zu können, wollen wir wieder ein Beispiel heranziehen: Eine Firma A bestehe aus drei Abteilungen, wobei sich für die Jahre 1994, 1995, 1996 und 1997 die in Abbildung 25.7 gezeigte Investitionsgelderverteilung ergibt.
Soll diese Tabelle innerhalb eines C-Programms in einem zusammenhängenden Speicherbereich abgelegt werden, müßte z. B. folgende Array-Definition angegeben werden:

```
int investition[3][4];
```

Diese Definition würde einen zusammenhängenden Speicherbereich reservieren, den wir uns – wie es in Abbildung 25.8 gezeigt ist – vorstellen können.
Wollten wir nun die Investition der Abteilung 1 im Jahr 1997 abspeichern, dann würde dies folgender Zuweisung entsprechen:

```
   investition[0][3] = 11783;
```

was ein Speicherbild ergäbe, wie es in Abbildung 25.9 gezeigt ist.
An der Zuweisung

Jahr / Abteilungen	1994	1995	1996	1997
Abteilung 1	10 532,-	8 955,-	9 374,-	11 783,-
Abteilung 2	9 743,-	12 377,-	11 539,-	13 893,-
Abteilung 3	3 747,-	5 988,-	10 782,-	12 977,-

Abbildung 25.7: Investionstabelle aufgeteilt nach Jahr und Abteilung

Abbildung 25.8: Vorstellung zur Speicherbelegung eines zweidimensionalen Arrays

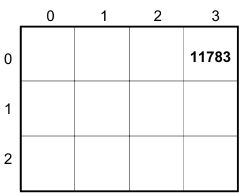


Abbildung 25.9: Belegung des zweidimensionalen Arrays nach `investition[0][3]=11783`

```
investition[0][3] = 11783;
      /* nicht: investition[0,3] = 11783; wie in anderen Sprachen */
```

können wir erkennen, dass in C ein zweidimensionales Array in Wirklichkeit ein eindimensionales Array ist – wie dies in Abbildung 25.10 gezeigt ist – bei dem jedes Element wieder ein Array darstellt, wie es in Abbildung 25.11 gezeigt ist.
Die Array-Elemente werden im Speicher natürlich nicht in einer zweidimensionalen, sondern in einer eindimensionalen Anordnung (nacheinander) gespeichert, wie dies Abbildung 25.12 zeigt.

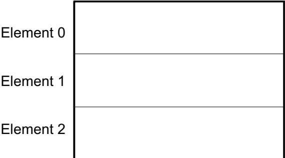


Abbildung 25.10: Zweidimensionales Array ist in Wirklichkeit ein eindimensionales Array

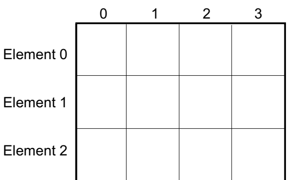


Abbildung 25.11: Zweidimensionales Array ist ein eindimensionales Array, dessen Elemente wieder eindimensionale Arrays sind

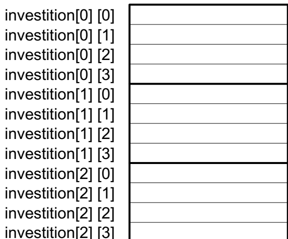


Abbildung 25.12: Array-Elemente werden im Speicher sequentiell nacheinander abgelegt

Über die Indizes wird dabei eine zweidimensionale Darstellung nachgebildet.

Beispiel:

Wir wollen nun ein C-Programm `invest.c` erstellen, das diese Tabelle speichert und dann am Bildschirm ausgibt:

```
#include <stdio.h>
/*-------------- striche -------------------------------------------------*/
void striche(char zeich) {
   int i;
    for (i=1 ; i<=52 ; i++)
       printf("%c", zeich);
    printf("\n");
}
/*-------------- main ---------------------------------------------------*/
int  main(void) {
    int  investition[3][4];
    int  i, j;

    investition[0][0] = 10532;
    investition[0][1] = 8955;
    investition[0][2] = 9374;
    investition[0][3] = 11783;

    investition[1][0] = 9743;
    investition[1][1] = 12377;
    investition[1][2] = 11539;
    investition[1][3] = 13893;

    investition[2][0] = 3747;
    investition[2][1] = 5988;
    investition[2][2] = 10782;
    investition[2][3] = 12977;

    printf("\n\n%15s|", " ");
    for (i=1994 ; i<=1997 ; i++)     /* Ausgabe der Jahre   */
        printf("%7d |", i);
```

```
    printf("\n");
    striche('=');
    for (i=0 ; i<=2 ; i++)  {
       printf("%13s%d |","Abteilung",i+1);
       for (j=0 ; j<=3 ; j++)
          printf("%7d |",investition[i][j]);
       printf("\n");
       if (i==2)
          striche('=');
       else
          striche('-');
    }
    return(0);
}
```

Im Programmteil

```
    investition[0][0] = 10532;
    investition[0][1] = 8955;
    investition[0][2] = 9374;
    investition[0][3] = 11783;

    investition[1][0] = 9743;
    investition[1][1] = 12377;
    investition[1][2] = 11539;
    investition[1][3] = 13893;

    investition[2][0] = 3747;
    investition[2][1] = 5988;
    investition[2][2] = 10782;
    investition[2][3] = 12977;
```

wird das zweidimensionale Array `investition` mit Werten belegt. Die Anweisungen:

```
    printf("\n\n%15s|", " ");
    for (i=1994 ; i<=1997 ; i++)     /* Ausgabe der Jahre   */
        printf("%7d |", i);
    printf("\n");
```

bewirken dann die Ausgabe des Tabellenkopfes:

```
               |   1994 |   1995 |   1996 |   1997 |
```

Mit dem Aufruf der Funktion

```
     striche("=");
```

wird folgende Zeile am Bildschirm ausgegeben:

```
=====================================================
```

Mit dem Programmteil

```
    for (i=0 ; i<=2 ; i++)  {
       printf("%13s%d |","Abteilung",i+1);
       for (j=0 ; j<=3 ; j++)
          printf("%7d |",investition[i][j]);
```

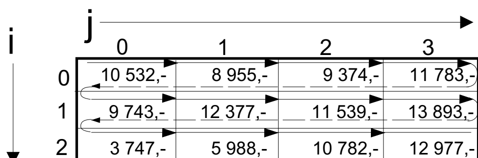


Abbildung 25.13: Reihenfolge, in der auf Elemente des Arrays `investition` zugegriffen wird

```
      printf("\n");
      if (i==2)
         striche('=');
      else
         striche('-');
   }
```

wird dann die eigentliche Tabelle am Bildschirm ausgegeben. Mit der äußeren Schleife werden die Zeilen von `investition` durchlaufen (Laufindex `i`); die innere Schleife (Laufindex `j`) sorgt dann dafür, dass zu jeder Zeile alle Spalten (eigentliche Werte) durchlaufen werden.
Somit wird auf die einzelnen Elemente von `investition` in der Reihenfolge zugegriffen, wie es in Abbildung 25.13 gezeigt ist.
Das Programm `invest.c` würde somit folgendes am Bildschirm ausgeben:

```
               |  1994 |   1995 |   1996 |   1997 |
====================================================
    Abteilung1 |  10532 |   8955 |   9374 |  11783 |
----------------------------------------------------
    Abteilung2 |   9743 |  12377 |  11539 |  13893 |
----------------------------------------------------
    Abteilung3 |   3747 |   5988 |  10782 |  12977 |
====================================================
```

Beispiel:

Es soll ein C-Programm `matadd.c` erstellt werden, das zwei zweidimensionale Arrays (Matrizen) durch Benutzereingaben mit Werten belegt und dann die Summe dieser beiden Matrizen wieder ausgibt. Eine Summe aus zwei Matrizen kann nur

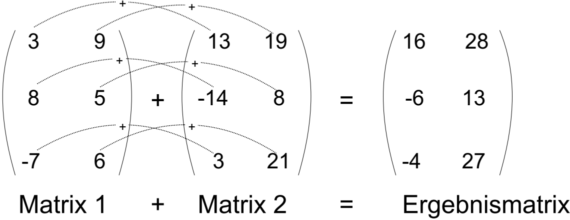


Abbildung 25.14: Addition von zwei Matrizen

dann gebildet werden, wenn beide Matrizen gleiche Zeilen- und Spaltenzahl besitzen. Bei der Summenbildung werden die einzelnen Elemente, die in beiden Matrizen an der gleichen Position stehen, addiert; siehe auch Abbildung 25.14.
Möglicher Ablauf des Programms `matadd.c`:

```
Addition zweier Matrizen
========================
Eingabe der Zeilen und Spalten der beiden Matrizen
   Zeilen: 3 ⟵
   Spalten: 2 ⟵

----Eingabe der Werte fuer Matrix 1----
Element [0][0]: 3 ⟵
Element [0][1]: 9 ⟵

Element [1][0]: 8 ⟵
Element [1][1]: 5 ⟵

Element [2][0]: -7 ⟵
Element [2][1]: 6 ⟵

----Eingabe der Werte fuer Matrix 2----
Element [0][0]: 13 ⟵
Element [0][1]: 19 ⟵

Element [1][0]: -14 ⟵
Element [1][1]: 8 ⟵

Element [2][0]: 3 ⟵
Element [2][1]: 21 ⟵

    3    9        13   19          16   28
    8    5  +    -14    8   =      -6   13
   -7    6         3   21          -4   27
```

Das Programm `matadd.c`:

```
#include  <stdio.h>
/*----------- eingabe -----------------------------------------------------*/
void eingabe(int matrix[10][10], int zeil, int spal) {
   int  i,  j;
    for (i=0 ; i<zeil ; i++)  {
       for (j=0 ; j<spal ; j++) {
          printf("Element [%d][%d]: ", i, j);
          scanf("%d",&matrix[i][j]);
       }
       printf("\n");
    }
}
/*----------- ausgabe -----------------------------------------------------*/
void ausgabe(int m1[10][10], int m2[10][10], int erg[10][10],
             int zeil, int spal) {
```

```
   int  i, j;
    for (i=0 ; i<zeil ; i++)  {
       for (j=0 ; j<spal ; j++)
          printf("%5d", m1[i][j]);
       printf("%s", (i==zeil/2) ? "  +  " : "     ");
       for (j=0 ; j<spal ; j++)
          printf("%5d", m2[i][j]);
       printf("%s", (i==zeil/2) ? "   =   " : "       ");
       for (j=0 ; j<spal ; j++)
          printf("%5d", erg[i][j]);
       printf("\n");
    }
}
/*----------- main --------------------------------------------------------*/
int  main(void) {
    int   mat1[10][10], mat2[10][10], sum_mat[10][10];
    int   i, j, zeilen, spalten;

    printf("\n\nAddition zweier Matrizen\n"
           "=======================\n\n\n");
      /*-------------- Eingabe der Zeilen- und Spaltenzahl */
    printf("Eingabe der Zeilen und Spalten der beiden Matrizen\n");
    printf("   Zeilen: ");
    scanf("%d", &zeilen);
    printf("   Spalten: ");
    scanf("%d", &spalten);
      /*-------------- Eingabe der Werte fuer   mat1  und   mat2 */
    printf("\n----Eingabe der Werte fuer Matrix 1----\n");
    eingabe(mat1, zeilen, spalten);
    printf("\n----Eingabe der Werte fuer Matrix 2----\n");
    eingabe(mat2, zeilen, spalten);
      /*--------------- Addition von  mat1  und  mat2 */
    for (i=0 ; i<zeilen ; i++)
       for (j=0 ; j<spalten ; j++)
          sum_mat[i][j] = mat1[i][j]+mat2[i][j];
      /*--------------- Ausgabe der Ergebnis-Matrix */
    ausgabe(mat1, mat2, sum_mat, zeilen, spalten);
    return(0);
}
```

Matrizen in `matadd.c` dürfen maximal 10 Zeilen und 10 Spalten besitzen.

Beispiel:

Um eine Nachricht zu verschlüsseln, kann man sie in einem zweidimensionalen Array zeilenweise eintragen und spaltenweise wieder auslesen; z. B. kann man die Nachricht

```
Eine Katze jagt die Maus
```

zeilenweise eintragen, wie es in Abbildung 25.15 gezeigt ist.
und spaltenweise auslesen (verschlüsselte Nachricht):

E	i	n	e	_	K
a	t	z	e	_	j
a	g	t	_	d	i
e	_	M	a	u	s

Abbildung 25.15: Zeilenweises Eintragen eines Textes in zweidimensionales Array

```
Eaaeitg nztMee a duKjis
```

Es soll nun ein geeignetes Programm `hebrae.c` für diese so genannte ***hebräische Verschlüsselungs-Methode*** entwickelt werden.
Zuerst muss der Benutzer seinen zu verschlüsselnden Satz eingeben. Nach dieser Eingabe wird der Benutzer nach der Zeilen- und Spaltenzahl für die Eintragung des zu verschlüsselnden Satzes gefragt.
Wenn beispielsweise die Nachricht

```
Ich lerne C
```

zu verschlüsseln ist und folgende Zeilen- und Spaltenzahl

```
Zeilen: 3
Spalten: 5
```

vom Benutzer angegeben wurde, so ergibt sich aus Abbildung 25.16 die verschlüsselte Nachricht:

```
IeCcr hn e l
```

I	c	h	_	l
e	r	n	e	_
C	_	_	_	_

Abbildung 25.16: Zeilenweises Eintragen eines Textes in zweidimensionales Array

Das Programm `hebrae.c`:

```
#include <stdio.h>

#define  MAX_ZEILEN  20
#define  MAX_SPALTEN 20
char   rechteck [MAX_ZEILEN] [MAX_SPALTEN];
char   text[MAX_ZEILEN*MAX_SPALTEN];
/*---------- lies_text --------------------------------------------------*/
int lies_text(void) {
   int  i=0;
    while ( (text[i++]=getchar())!=’\n’ && i<MAX_ZEILEN*MAX_SPALTEN)
       ;
    return(i-1);
}
/*---------- main -------------------------------------------------------*/
int  main(void) {
    int  laenge, i, j, k, zeilen, spalten;
```

```
    printf("\nGeben Sie den zu verschluesselnden Satz ein !\n");
    laenge = lies_text();  /* In laenge wird Laenge des
                              eingegebenen Textes gespeichert */
        /*---------------------------------------------------------------------*/
        /* Eingabe der Zeilen- und Spaltenzahl fuer Verschluessel.-Rechteck  */
        /*---------------------------------------------------------------------*/
    do  {
       printf("\nWieviel Zeilen soll Verschluessel.-Rechteck besitzen? ");
       scanf("%d", &zeilen);
       printf("Wieviel Spalten soll Verschluessel.-Rechteck besitzen? ");
       scanf("%d", &spalten);
       if (zeilen*spalten < laenge)  {
           printf("\n .....Zu wenig Platz im Verschluessel.-Rechteck !!\n");
           printf(" .....Wiederholen Sie Ihre Eingabe !\n");
       }
    } while (zeilen*spalten < laenge);
        /*---------------------------------------------------------*/
        /* Auffuellen des Arrays text mit Leerzeichen             */
        /*---------------------------------------------------------*/
    for (i=laenge ; i<MAX_ZEILEN*MAX_SPALTEN ; i++)
       text[i] = ' ';
        /*---------------------------------------------------------*/
        /* Eintragung des Textes in das Verschluessel.-Rechteck   */
        /*---------------------------------------------------------*/
    k = 0;
    for (i=0 ; i<zeilen ; i++)
        for (j=0 ; j<spalten ; j++)
           rechteck[i][j] = text[k++];
        /*---------------------------------------------------------*/
        /* Auslesen des Textes aus Verschluessel.-Rechteck        */
        /*---------------------------------------------------------*/
    printf("\nDie verschluesselte Nachricht ist:\n");
    for (j=0 ; j<spalten ; j++)
        for (i=0 ; i<zeilen ; i++)
           printf("%c", rechteck[i][j]);
    printf("\n");
    return(0);
}
```

Mögliche Abläufe des Programms `hebrae.c`:

```
Geben Sie den zu verschluesselnden Satz ein !
Eine Katze jagt die Maus ⏎
Wieviel Zeilen soll Verschluessel.-Rechteck besitzen? 4 ⏎
Wieviel Spalten soll Verschluessel.-Rechteck besitzen? 6 ⏎

Die verschluesselte Nachricht ist:
Eaaeitg nztMee a  duKjis
```

```
Geben Sie den zu verschluesselnden Satz ein !
Eaaeitg nztMee a  duKjis ↵
Wieviel Zeilen soll Verschluessel.-Rechteck besitzen? 6 ↵
Wieviel Spalten soll Verschluessel.-Rechteck besitzen? 4 ↵

Die verschluesselte Nachricht ist:
Eine Katze jagt die Maus
```

```
Geben Sie den zu verschluesselnden Satz ein !
Ich lerne C ↵
Wieviel Zeilen soll Verschluessel.-Rechteck besitzen? 3 ↵
Wieviel Spalten soll Verschluessel.-Rechteck besitzen? 5 ↵

Die verschluesselte Nachricht ist:
IeCcr hn  e l
```

25.2.2 Drei-, vier-, fünf- und sonstige mehrdimensionale Arrays

Es können natürlich nicht nur zweidimensionale Arrays, sondern auch drei-, vier- und sonstige mehrdimensionale Arrays definiert werden, wie z. B. das folgende fünfdimensionale Array

```
double x[10][200][50][30][100];
```

Bei solchen mehrdimensionalen Array-Definitionen muss man nur berücksichtigen, dass man oft mit harmlos aussehenden Definitionen wie der obigen sehr viel Speicher anfordert, über den die betreffende Maschine eventuell nicht verfügt. So würde z. B. die obige Definition bereits zu einer Speicheranforderung von

```
10*200*50*30*100 = 300 Millionen * 8 Byte (für double)
      --> mehr als 2 GigaByte
```

führen.

25.2.3 `sizeof` liefert die Größe eines Arrays

Mit dem `sizeof`-Operator läßt sich die Anzahl der Bytes ermitteln, die von einem Array belegt werden.
Dividiert man diese Größe dann noch durch die Anzahl der Bytes, die der Datentyp eines Elements belegt (wie z. B. `sizeof(int)` oder `sizeof(double)`), so erhält man die Anzahl der Elemente im entsprechenden Array.
Beispiel:
Das folgende Programm `arrysize.c` demonstriert diese Anwendung des `sizeof`-Operators.

```
#include  <stdio.h>

int     kalender[12][31];
double  matrix[25][40];
char    adressen[200][100];
```

```
int  main(void) {
    printf("kalender:\n");
    printf("   %d Elemente\n", sizeof(kalender)/sizeof(int));
    printf("   %d Bytes\n", sizeof(kalender));
    printf("   bei %d Bytes fuer int\n", sizeof(int));
    printf("matrix:\n");
    printf("   %d Elemente\n", sizeof(matrix)/sizeof(double));
    printf("   %d Bytes\n", sizeof(matrix));
    printf("   bei %d Bytes fuer double\n", sizeof(double));
    printf("adressen:\n");
    printf("   %d Elemente\n", sizeof(adressen)/sizeof(char));
    printf("   %d Bytes\n", sizeof(adressen));
    printf("   bei %d Bytes fuer char\n", sizeof(char));
    return(0);
}
```

Möglicher Ablauf dieses Programms `arrysize.c`:

```
kalender:
   372 Elemente
   1488 Bytes
   bei 4 Bytes fuer int
matrix:
   1000 Elemente
   8000 Bytes
   bei 8 Bytes fuer double
adressen:
   20000 Elemente
   20000 Bytes
   bei 1 Bytes fuer char
```

25.2.4 Übung: Game of Life (Beispiel für zellulare Automaten)

Die wesentliche Eigenschaft lebender Organismen ist ihre Fähigkeit zur Selbstreproduktion. Jeder Organismus kann Nachkommen erzeugen, die – bis auf Feinheiten – eine Kopie des erzeugenden Organismus sind. *John von Neumann* stellte folgende Frage: ***Sind auch Maschinen (z. B. Roboter) zur Selbstreproduktion fähig? Welche Art logischer Organisation ist dafür notwendig und hinreichend?*** *S. M. Ulam* schlug die Verwendung so genannter *zellularer Automaten* vor. Einen zellularen Automaten kann man sich anschaulich als eine in Quadrate (Zellen) aufgeteilte Ebene vorstellen. Auf jedem Quadrat befindet sich ein endlicher Automat, dessen Verhalten von seinem eigenen Zustand und von den Zuständen gewisser Nachbarn (Zellen) abhängt. Alle Automaten sind gleich und arbeiten im gleichen Takt.
Ein berühmtes Beispiel für einen zellularen Automaten ist das *Game of Life (Lebensspiel)* des englischen Mathematikers *John H. Conway*. Jede Zelle hat zwei Zustände (lebend, tot) und die Umgebung der Zelle besteht aus den angrenzenden acht Nachbarquadraten. Die Zeit verstreicht in diskreten Schritten. Von einem Schlag der kosmischen Uhr bis zum nächsten verharrt die Zelle im zuvor eingenommenen

Zustand, beim Gong aber wird nach den folgenden Regeln erneut über Leben und Tod entschieden:

- **Geburt**
  Eine tote Zelle feiert Auferstehung, wenn drei ihrer acht Nachbarn leben.
- **Tod durch Überbevölkerung**
  Eine Zelle stirbt, wenn vier oder mehr Nachbarn leben.
- **Tod durch Vereinsamung**
  Eine Zelle stirbt, wenn sie keinen oder nur einen lebenden Nachbarn hat.

Eine lebende Zelle bleibt also genau dann am Leben, wenn sie zwei oder drei lebende Nachbarn besitzt. Der Reiz dieses Spiels liegt in seiner Unvorhersehbarkeit. Nach den oben angegebenen Regeln kann eine Population aus lebenden Zellen grenzenlos wachsen, sich zu einem periodisch wiederkehrenden oder stabilen Muster entwickeln oder aussterben.

Erstellen Sie ein Programm `life.c`, bei dem der Benutzer zunächst die Länge und Breite des Spielfelds eingibt. Danach soll der Benutzer wählen können, ob er die Anfangspopulation selbst (über Mausklicks) eingeben will oder ob diese zufällig sein soll. Im zweiten Fall soll der Benutzer noch eingeben können, wie viele Zellen zu Beginn leben sollen. Um diese Übung zu lösen, muss man das Kapitel 21 auf Seite 301 zur Graphikprogrammierung gelesen haben. Die Abbildungen 25.17 und 25.18 zeigen mögliche Anzeigen des Programms `life.c`.

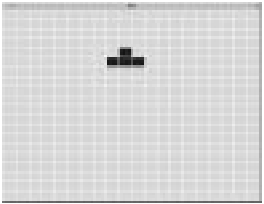
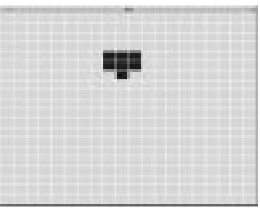
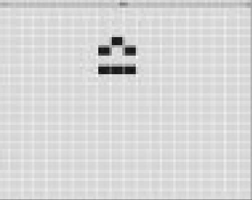




Abbildung 25.17: Population zu Beginn, nach dem ersten und dem zweiten Schritt

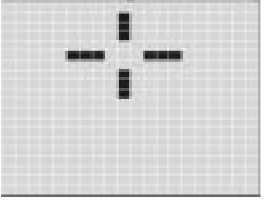
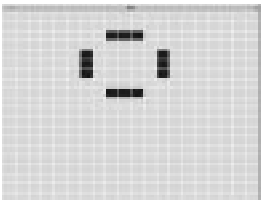



Abbildung 25.18: Populationen nach weiteren Schritten, die sich nun ständg wiederholen

Weitere interessante Figuren sind:

```
                                      x    x
                x                      xxxx
     xx        xxx       x            x    x
    xx          x         x           x xx x
     x          x       xxx           x    x
                                       xxxx
Pentomino     Kreuz   Gleiter      Chesire-Katze
```

Eine andere Population mit überraschendem Verhalten ist:

```
xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx
```

25.3 Zusammenhänge zwischen Arrays und Zeigern

Arrays stehen in engem Zusammenhang mit Zeigern. Zeigervariablen enthalten Adressen und werden bei der Deklaration durch Voranstellen eines Stern (*) vor dem Variablennamen als solche gekennzeichnet, wie z. B.:

```
int *zgr1, *zgr; /* Zwei Zeigervariablen, die auf int-Speicherplätze
                    zeigen können                                    */
```

In diesem Kapitel werden wir erfahren, dass Arrays in der Programmiersprache C sehr viel mit Zeigern gemeinsam haben.

25.3.1 Arrayname ist konstanter Zeiger auf erstes Element

In C gilt immer, dass der Name eines Arrays automatisch ein ***konstanter Zeiger*** auf das erste Element des Arrays ist. Wenn wir z. B. folgende Array-Definition angeben:

```
int vektor[5];
```

so definieren wir ein Array `vektor` mit den `5` Elementen `vektor[0]`, `vektor[1]`, ..., `vektor[4]`. `vektor[i]` liegt dann `i` Elemente vom Array-Anfang entfernt. Gibt man nur den Arraynamen `vektor` alleine (ohne Index) an, so erhält man die Anfangsadresse von `vektor`. Will man also einen Zeiger, der z. B. mit

```
int *zeiger;
```

definiert ist, auf den Anfang des Arrays `vektor` positionieren, so kann eine der beiden folgenden Anweisungen, die beide das gleiche leisten, angegeben werden:

```
zeiger = &vektor[0]; /* & = Adreßoperator; liefert Adresse des entspr. Objekts */
zeiger = vektor;
```

Soll nun das erste Element von `vektor` einer `int`-Variablen `ganz` zugewiesen werden, so könnten wir, da sowohl `zeiger` als auch `vektor` auf das erste Element zeigen, wieder zwischen den folgenden beiden Anweisungen wählen:

```
ganz = *zeiger;
ganz = *vektor;
```

Beide Anweisungen entsprechen der folgenden Anweisung

```
ganz = vektor[0];
```

Mit der in Abbildung 25.19 gezeigten Symbolik für Zeiger, Variablen und Werte lassen sich die eben beschriebenen Operationen in einer Bildsequenz anschaulich machen:

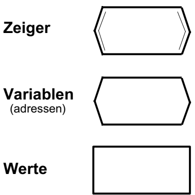


Abbildung 25.19: Symbolik für Zeiger, Variablen und Werte

int vektor[5];

Bei der Darstellung dieser Definition wird davon Gebrauch gemacht, dass ein konstanter Zeiger mit Namen `vektor` automatisch angelegt wird, der auf das `0`. Element des Arrays `vektor` zeigt, wie dies in Abbildung 25.20 gezeigt ist.

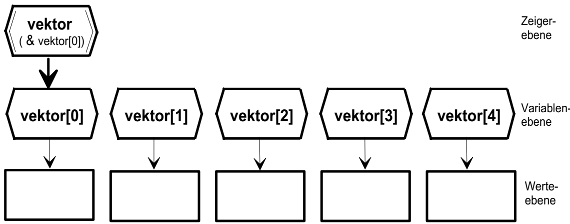


Abbildung 25.20: Konstanter Zeiger `vektor` nach der Deklaration `int vektor[5]`

int *zeiger;

Deklaration eines noch undefinierten Zeigers; siehe auch Abbildung 25.21.

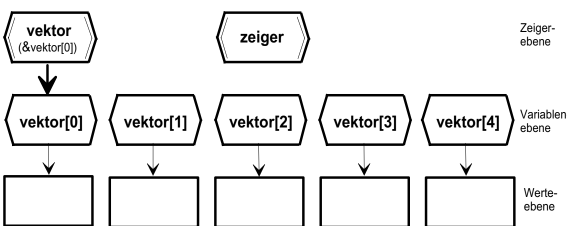


Abbildung 25.21: Nach der Deklaration eines undefinierten Zeigers mit `int *zeiger`

int ganz;

Deklaration einer weiteren Variablen; siehe auch Abbildung 25.22.

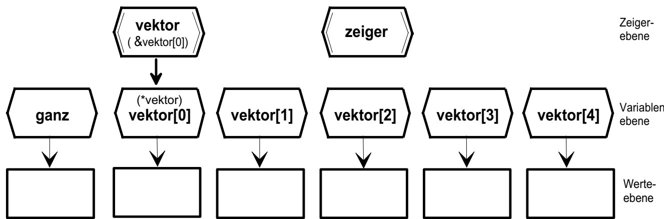


Abbildung 25.22: Nach der Deklaration einer weiteren Variablen mit `int ganz`

zeiger = vektor; (entspricht **zeiger = & vektor [0];)**

Den Zeiger `zeiger` auf den Arrayanfang zu positionieren; siehe auch Abbildung 25.23.

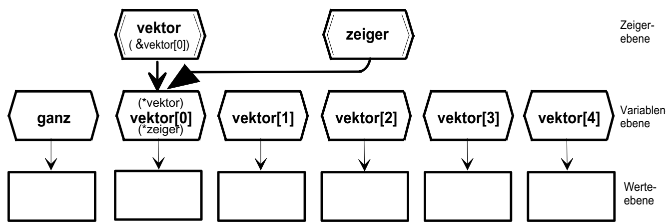


Abbildung 25.23: Positionieren des Zeigers auf den Arrayanfang (mit `zeiger = vektor`)

for (i=0; i<=4; ++i)

vektor[i]=i+1;

Den Elementen des Arrays Werte zuweisen; siehe auch Abbildung 25.24.

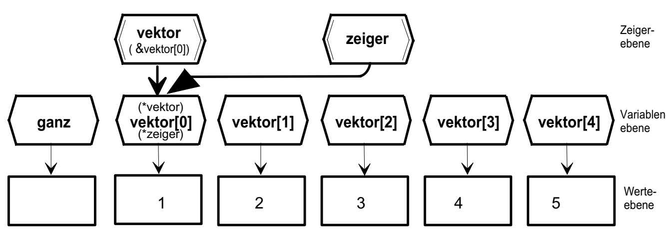


Abbildung 25.24: Nach der Zuweisung von Werten an die Elemente des Arrays

ganz=*zeiger; (entspricht **ganz=*vektor;** oder **ganz=vektor[0];**)

Das Element `vektor[0]` der Variablen `ganz` zuweisen; siehe auch Abbildung 25.25.

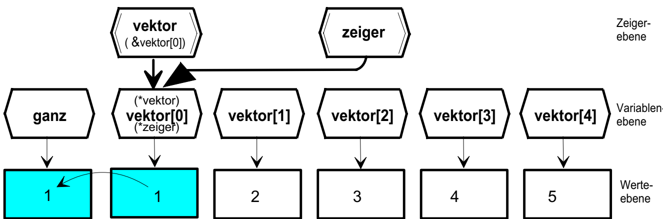


Abbildung 25.25: Nach der Zuweisung `ganz = *zeiger`

Der Name eines Arrays ohne Angabe von Indizes entspricht also immer einem Zeiger auf den Anfang des Arrays, ohne dass dieser als Zeiger deklariert wurde; d. h., wir können z. B. über einen Zeiger verfügen, ohne dass für diesen Zeiger Speicherplatz reserviert wird. Dies wiederum bedeutet, dass ein solcher implizit fest vorgegebener Zeiger nicht verändert werden kann und immer auf dieselbe Adresse (Anfangsadresse eines Arrays) zeigt.

25.3.2 Zugriff auf Arrayelemente ist auch über Zeiger möglich

Nachfolgend wird noch ein weiteres Beispiel in einer Bildsequenz nachvollzogen. Zeigt `zeiger` auf ein bestimmtes Element eines Arrays `elem`, dann zeigt `zeiger+1` auf das nachfolgende Element und `zeiger-1` auf das vorhergehende:

```
int elem[3], *zeiger; /* siehe Abb. 25.26 */
```

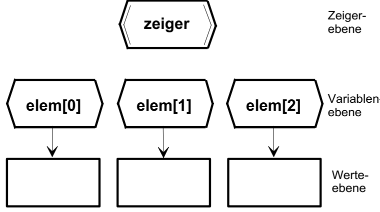


Abbildung 25.26: Nach der Deklaration `int elem[3], *zeiger;`

Bei Abbildung 25.26 und den folgenden Abbildungen wird aus Übersichtsgründen der implizit vorhandene und konstante Zeiger `elem` nicht dargestellt.

```
zeiger=&elem[1]; /* siehe Abb. 25.27 */
```

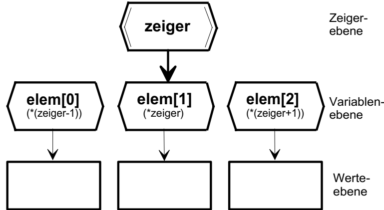


Abbildung 25.27: Nach der Zuweisung `zeiger=&elem[1]`

```
*(zeiger-1) = 10; (entspricht elem[0]=10) /* s. Abb. 25.28 */
*zeiger = 20; (entspricht elem[1]=20)
*(zeiger+1) = 30; (entspricht elem[2]=30)
```

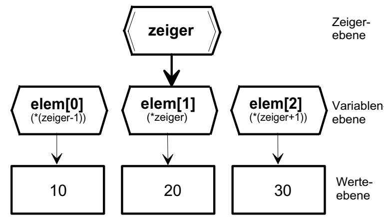
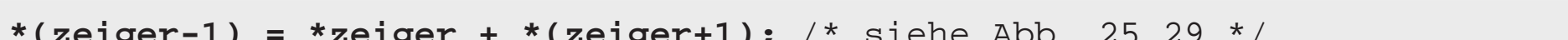


Abbildung 25.28: Nach der Zuweisung von Werten über `zeiger`



```
*(zeiger-1) = *zeiger + *(zeiger+1); /* siehe Abb. 25.29 */
```

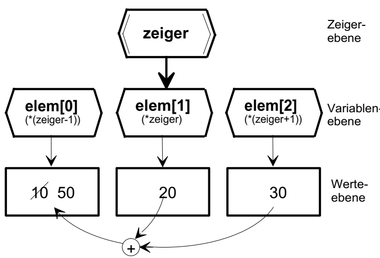


Abbildung 25.29: Nach der Zuweisung `*(zeiger-1) = *zeiger + *(zeiger+1);`

Diese Anweisung entspricht also:

```
elem[0]= elem[1]+ elem[2];
```

Dieses Beispiel wollen wir nun einmal mit fiktiven Adressen auf Maschinenebene durchspielen, wobei hier für den Datentyp `int` zwei Bytes angenommen werden:

```
int elem[3], *zeiger; /* siehe Abb. 25.30 */
```

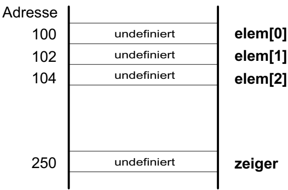


Abbildung 25.30: Nach der Deklaration `int elem[3], *zeiger;`

```
zeiger=&elem[1]; /* siehe Abb. 25.31 */
```

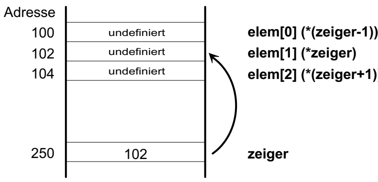


Abbildung 25.31: Nach der Zuweisung `zeiger=&elem[1]`

```
*(zeiger-1) = 10; /* siehe Abb. 25.32 */
*zeiger = 20;
*(zeiger+1) = 30;
```

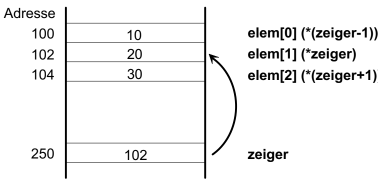


Abbildung 25.32: Nach der Zuweisung von Werten über `zeiger`

```
*(zeiger-1) = *zeiger + *(zeiger+1); /* siehe Abb. 25.33 */
```

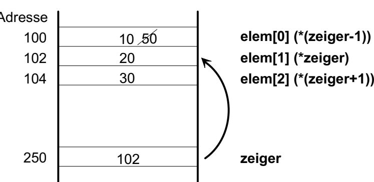


Abbildung 25.33: Nach der Zuweisung `*(zeiger-1) = *zeiger + *(zeiger+1);`

Allgemein können wir also feststellen, dass `zeiger+i` auf das `i`-te Element nach `zeiger` und `zeiger-i` auf das `i`-te Element vor `zeiger` zeigt.
Wenn also `zeiger` auf `vektor[0]` zeigt, dann entspricht `*(zeiger+1)` dem Element `vektor[1]` und `*(zeiger+i)` dem Element `vektor[i]`; siehe auch Abbildung 25.34.

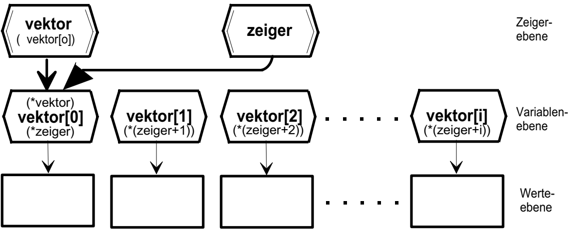


Abbildung 25.34: Zugriff auf Arrayelemente sowohl über `*(zeiger+i)` als auch über `vektor[i]` möglich

`zeiger+i` ist also die Adresse des Elements `vektor[i]` (`&vektor[i]`) und `*(zeiger+i)` ist das Array-Element `vektor[i]` selbst.

Beispiel:

Wir wollen das Konvertierungs-Programm `dezumwan.c` von Seite 520 so umschreiben, dass die Arrayelemente nicht mehr mit Angabe eines Index wie in `ziel[i]`, sondern über den Arraynamen `ziel` mit Distanzangabe `+i` angesprochen werden.
Das folgende Programm `dezumwa2.c` leistet dann dasselbe wie das Programm `dezumwan.c` von Seite 520.

```
#include  <stdio.h>

int  main(void)
{
    int  zahl, basis, zaehler=0, i;
    int  ziel[100];

    while (1) {
       printf("Gib Basis des Zielsystem ein (2<=Basis<=10): ");
       scanf("%d", &basis);
       if (basis>=2 && basis<=10)
          break;
    }
    printf("Gib die zu wandelnde Zahl aus dem Zehnersystem ein: ");
    scanf("%d", &zahl);

    printf("     ---> %d(10) = ", zahl);
    while (zahl>0)  {
       *(ziel+zaehler) = zahl % basis; /* statt: ziel[zaehler] = zahl % basis; */
       zahl /= basis;
```

Index

< Operator, 44
<<Operator, 61
<= Operator, 44
== Operator, 44
> Operator, 44
>= Operator, 44
>>Operator, 61
| Operator, 47, 52
|| Operator, 48
\ \, 101, 1059
\ ', 101, 1059
\ ", 101, 1059
\ a, 101, 1059
\ b, 101, 1059
\ e, 101, 1059
\ f, 101, 1059
\ ooo, 101, 1059
\ r, 101, 1059
\ t, 101, 1059
\ v, 101, 1059
\ x, 101, 1059
++ Operator, 68, 74
, Operator, 199, 207
-- Operator, 68, 74
<assert.h>, 495
<complex.h>, 951
<ctype.h>, 93
<dirent.h>, 884
<errno.h>, 854
<fcntl.h>, 861
<fenv.h>, 955
<float.h>, 149
<graphics.h>, 302
<inttypes.h>, 959
<iso646.h>, 965
<limits.h>, 147, 860
<locale.h>, 935
<math.h>, 121
<setjmp.h>, 923
<signal.h>, 915
<stdarg.h>, 390
<stdbool.h>, 46
<stddef.h>, 106, 117
<stdint.h>, 106, 117, 958
<stdio.h>, 808
<stdlib.h>, 594
<string.h>, 576
<sys/stat.h>, 877
<tgmath.h>, 964
<time.h>, 286
<wchar.h>, 942
<wctype.h>, 942
?:, 183
Übung
  adturing.c, 485
  aepfel.c, 264
  anhaeng.c, 873
  aribit.c, 64
  armstron.c, 272
  basket.c, 345
  benzinv.c, 81
  bis100.c, 278
  block3.c, 432
  bruchrec.c, 704
  bucstrei.c, 608
  c_worte.c, 719
  caprecap.c, 112
  chinzahl.c, 621
  constzei.c, 460
  datediff.c, 722
  dauerkal.c, 621
  demere.c, 374
  dhondt.c, 642
  dodon.c, 526
  domino.c, 243
  dualaddi.c, 485
  dualwand.c, 413
  engel.c, 14
  entferbi.c, 609
  erdumf.c, 81
  float.c, 514
  geldstap.c, 82
  geldstck.c, 128
  ggtrekur.c, 412
  gleiteig.c, 155
  harmon.c, 243
  idealgew.c, 187
  inkdek.c, 71
  josephus.c, 779
  kandproz.c, 145
  kettenpi.c, 44
  koch.c, 346, 414
  komment.c, 14
  kosmisch.c, 643
  kreis.c, 198
  kugel.c, 80
  laugsaeu.c, 805
  life.c, 538
  limits.c, 514
  logbit.c, 64
  machebi.c, 609
  meincd.c, 910
  menue.c, 14
  mywhich.c, 903
  nulleins.c, 569
  numausg.c, 859
  numrueck.c, 688
  nytimes.c, 43
  okdezhex.c, 128
  pflanze1.c, 413
  piregen.c, 344
  polydraw.c, 396
  primfakt.c, 264
  primzahl.c, 278
  quadzeic.c, 347
  reaktest.c, 920
  rechnung.c, 181
  schalt.c, 181
  sincos.c, 129
  sparrate.c, 374
  sparswei.c, 141
  speikla1.c, 456
  spiegel.c, 568
  treesize.c, 892
  vielmax.c, 396
  vumrech.c, 100, 101
  wachsen.c, 526
  wertber.c, 148
  wortlen.c, 608
  wortstat.c, 780
  zahlrat.c, 271
  zahlsys.c, 657
  zeitrech.c, 388
  ziffadd1.c, 294
  ziffadd2.c, 294
  zusop.c, 67
  zusop2.c, 76
(Null-Direktive), 513
Operator, 503
Operator, 503
#define, 77, 79, 500
#elif, 488
#else, 488
#endif, 488
#error, 513
#if, 488
#ifdef, 488

Index

#ifndef, 488
#include, 83, 497
#line, 512
#pragma, 513
#undef, 506
& Adreßoperator, 384
& Operator, 52
&& Operator, 47, 48
_Bool Datentyp, 20, 46
_Complex Datentyp, 20, 951
_IOFBF Konstante, 849
_IOLBF Konstante, 849
_IONBF Konstante, 849
_Imaginary Datentyp, 20, 951
__DATE__, 511
__FILE__, 511
__LINE__, 511
__STDC_HOSTED__, 511
__STDC_IEC_559_COMPLEX__, 511
__STDC_IEC_559__, 511
__STDC_ISO_10646__, 512
__STDC_VERSION__, 511
__STDC__, 511
__TIME__, 511
_exit(), 897
! Operator, 44
^ Operator, 52
Bit-Operatoren, 52
Bitfelder, 788
Makro
  __DATE__, 511
  __FILE__, 511
  __LINE__, 511
  __STDC_HOSTED__, 511
  __STDC_IEC_559_COMPLEX__, 511
  __STDC_IEC_559__, 511
  __STDC_ISO_10646__, 512
  __STDC_VERSION__, 511
  __STDC__, 511
  __TIME__, 511
  Vordefiniert, 511

abort(), 920
access time, 881
access(), 879
acos(), 121
acosh(), 122
add2.c, 119
add2alt.c, 365
add2neu.c, 364
addalt.c, 362
addiere.c, 91
addmult.c, 69
addneu.c, 363
Adreßoperator &, 384
adress1.c, 799
adturing.c, 485, 1020
aepfel.c, 264, 993
Aktuelle Koordinate, 310
Aktueller Parameter, 351, 376
Aktuelles Argument, 351, 376
ALGOL, 464
AND-Operator, 48
anhaeng.c, 873, 1049
ANSI, 4
Anweisung, 40
arc(), 317
arg_env.c, 909
argreih.c, 386
argtest.c, 645
argtest2.c, 646
argtest3.c, 647
Argument, 351, 376
Argumente
  Kommandozeile, 645
  Optionen, 648, 655
argvor.c, 385
aribit.c, 64, 975
arithmetische Operatoren, 39
armstron.c, 272, 995
arr_swap.c, 625
Array, 304, 517
  Funktionsadressen, 639
  Initialisierung, 609
  mehrdimensional, 527
  Struktur, 709
  und Zeiger, 539
  zweidimensional, 527
Array-Zeiger, 623
arryfeh2.c, 525
arrygros.c, 522
arryinitc99.c, 612
arrynoname.c, 615
arryok.c, 524
arrysize.c, 536
arrystatic.c, 627
artizahl.c, 717
asctime(), 287
asin(), 121
asinh(), 122
Assembler, 462
assert(), 495
Assoziativität, 72
atan(), 121
atan2(), 121
atanh(), 122
atexit(), 898
atexit.c, 898
atof(), 600
atoi(), 599
atol(), 600
atoll(), 600
attribut.c, 879
Attribute von Dateien, 876
Ausdruck, 40
Ausgabe
  ein Zeichen, 85
  gepuffert, 86
  printf(), 101, 1059
  putchar(), 85
Auswertung, 39
Auswertungszeitpunkt, 74
auto, 436
autoadr.c, 444
autoggt.c, 436
autokauf.c, 162

B Sprache, 3
Balken
  dreidimansional, 318
  zweidimensional, 318
bar(), 318
bar3d(), 318
BASIC, 464
basket.c, 345, 1004
Baum
  Binär, 767
baum.c, 886
Baumstrukturen, 404
BCPL Sprache, 3
Bedingte Bewertung, 183
Bedingte Kompilierung, 488
Beenden eines Programms, 897
benzinv.c, 81, 980
Bibliothek, 83, 351
bildschi.c, 477
bildschi.h, 475
Binärbaum, 767
Binäre Suche, 563, 726
Binärmodus, 837
binatext.c, 838
bis100.c, 278, 996
bitand.c, 54
bitfeld.c, 790
bitfeld2.c, 792
bitgleit.c, 58
bitor.c, 56
bitumdre.c, 493
bitxor.c, 57
Blöcke, 427
block.c, 430
block3.c, 432, 1019
Blockstruktur, 427
bonbon.c, 247
bonbon2.c, 247
bool Datentyp, 46
booltyp.c, 49
Bottom-up, 470
break, 273
Breite eines Textes, 324
briefmar.c, 230
bruchrec.c, 704, 1041
btowc(), 950
Bubble-Sort, 560
bubble.c, 560
buchstat.c, 858
buchstnr.c, 519
buchzaeh.c, 574
bucstrei.c, 608, 1031
BUFSIZ Konstante, 850
byte stream, 807
bytezahl.c, 816

C++, 3
C89, 4
C99, 5
c_worte.c, 719, 1043
cabs(), 952
cacos(), 952
cacosh(), 952
Call
  by reference, 381
  by value, 381
calloc(), 671
caprecap.c, 112, 982
carg(), 952
casin(), 952
casinh(), 952
Cast, 142

cast.c, 144
casting, 142
cat Kommando, 867
catan(), 952
catanh(), 952
cbrt(), 122
ccos(), 952
ccosh(), 952
ceil(), 121
cexp(), 952
Chaos, 341
char Datentyp, 19
char-Konstanten, 26
char-Zeiger, 569
chdir(), 882
chinzahl.c, 621, 1034
chmod(), 878
Chromsky-Grammatik, 406
cimag(), 952
circle(), 317
cleardevice(), 310
clearenv(), 903
clearerr(), 814
clearviewport(), 334
clock(), 287
clock1.c, 283
clog(), 952
close(), 863
closedir(), 884
closegraph(), 305
COBOL, 464
Compiler, 471
complex.c, 952
conj(), 952
const, 79, 457, 620
const.c, 458
constzei.c, 460, 1020
continue, 279
copysign(), 122
copyzeit.c, 833
cos(), 121
cosh(), 121
cpow(), 952
cproj(), 952
creal(), 952
creat(), 862
creation time, 881
csin(), 952
csinh(), 952
csqrt(), 952
ctan(), 952
ctanh(), 952
ctime(), 287

Danksagung, 1
dataus.c, 819
datbytes.c, 842
datediff.c, 722, 1044
Datei, 807
  Änderungszeit, 881
  Öffnen, 809, 860, 862
  Anlegen, 862
  Attribute, 876
  Binärmodus, 837
  creation time, 881
  EOF-Flag, 813, 814
  Fehler-Flag, 813, 814
  Größe, 880
  Kreieren, 862
  Löschen, 848
  Lesen (blockweise), 827
  Lesen (byteweise), 863
  Lesen (ein Zeichen), 814, 815
  Lesen (eine Zeile), 818
  Lesen (formatiert), 820
  modification time, 881
  Positionieren, 841, 845, 846, 868
  Pufferung, 849
  Schliessen, 809, 811, 863
  Schreiben (blockweise), 827
  Schreiben (byteweise), 866
  Schreiben (ein Zeichen), 814, 815
  Schreiben (eine Zeile), 818
  Schreiben (formatiert), 820, 834
  Stream verknüpfen, 846
  Temporäre, 851
  Textmodus, 837
  Umbenennen, 848
  Zeit, 881
  Zeit der i-node-Änderung, 881
  Zugriffszeit, 881
  Zurückschieben (ein Zeichen), 826
dateiart.c, 877
dateiauf.c, 811
Datentyp, 21, 1058
  _Bool, 20, 46
  _Complex, 20, 951
  _Imaginary, 20, 951
  bool, 46
  char, 19
  double, 20
  eigener, 795
  FILE, 808
  float, 20
  fpos_t, 840
  Gleitpunkt-, 149
  Grund-, 19
  int, 19
  long, 20
  long long, 20
  short, 19
  signed, 19
  Tabelle, 21, 1058
  unsigned, 19, 20
  va_list, 390
  void, 357, 376
  Wertebereich, 21, 1058
Datentypen
  und Operatoren, 73
Datentypumwandlung, 133, 142, 366
  casting, 142
  explizit, 142
  implizit, 133, 366
  implizite, 131
Datum des Tages, 291
datuminfo.c, 292
Datumsangaben, 286
datzeile.c, 843
datzeit.c, 881
dauerkal.c, 621, 1034
defined, 488
Definition
  Funktionen (Alt-C), 356
  Funktionen (C89/C99), 353
  Konstanten, 77, 79
  Variablen, 32
Dekrement-Operator --, 68
delay(), 307
demere.c, 374, 1010
devnr.c, 891
dezimale Konstanten, 27
dezumwa2.c, 545
dezumwan.c, 520
dhondt.c, 642, 1036
difftime(), 287
DIR Struktur, 885
dirdemo.c, 883
Directory, 875, 882
  Anlegen, 882
  Löschen, 848, 882
  Lesen, 884
  Wechseln, 882
  Working, 882
dirent Struktur, 884
do...while, 267
dodon.c, 526, 1025
domino.c, 243, 992
Doppelt verkettete Liste, 761
double Datentyp, 20, 149
drawpoly(), 319
dreichi1.c, 634
dreichi2.c, 638
dreieck.c, 404
dreizeic.c, 86
du Kommando, 880
dualaddi.c, 485, 1022
dualbcd.c, 616
Duale Addition, 18
dualwand.c, 413, 1017
dualzahl.c, 217
Dynamische
  Speicherallozierung, 659
  Strukturarrays, 737
dynarr.c, 622

E/A-Funktionen, 807, 859
E/A-Umlenkung, 823
e_reihe.c, 455
eifrau1.c, 277
eifrau2.c, 277
Eigener Datentyp, 795
eimalei.c, 212
einaus.c, 307
einauscp.c, 823
Einer-Komplement, 52
Eingabe
  ein Zeichen, 85
  gepuffert, 86
  getchar(), 85
  scanf(), 113, 1061
einkompl.c, 53
einles.c, 433
Einseitige
  if-Anweisung, 166
einzeich.c, 85
Elementare E/A-Funktionen, 859
Ellipse
  ausgefüllt zeichnen, 318, 330

zeichnen, 318
ellipse(), 318
Ellipsen-Prototypen, 389
endlos.c, 814
Endlose for-Schleife, 219
engel.c, 14, 967
englzah1.c, 629
englzah2.c, 631
entferbi.c, 609, 1033
Entwurf
Bottom-up, 470
Modularer, 470
Top-Down, 470
enum, 801
enumdemo.c, 805
environ Variable, 900
Environment, 900
envlist1.c, 900
envlist2.c, 901
EOF-Flag, 813, 814
erdumf.c, 81, 979
errno Variable, 854
errodemo.c, 855
erst.c, 7
execdemo.c, 909
execl(), 907
execle(), 907
execlp(), 907
execv(), 907
execve(), 907
execvp(), 907
Exit
Handler, 898
Status, 894
exit(), 897
exp(), 121
exp2(), 122
Explizite Datentypumwandlung, 142
expm1(), 122
exstat.c, 896
extern, 432

Füllmuster, 318
fabs(), 121
faktor.h, 981
fakul.c, 399
fakul2.c, 401
FALSE, 46
false Konstante, 46
Farbpalette, 332
farbstuf.c, 332
fclose(), 811
fdim(), 122
fdopen(), 873
feclearexcept(), 955
fegetenv(), 956
fegetexceptflag(), 955
fegetround(), 955
fehlausg.c, 394
Fehler-Flag, 813, 814
fehler.c, 479, 856
fehler.h, 476
Fehlermeldung, 854
fehlhand.c, 856
feholdexcept(), 956
fenv.c, 956
feof(), 813
feraiseexcept(), 955
ferror(), 813
fesetenv(), 956
fesetexceptflag(), 955
fesetround(), 956
fetestexcept(), 955
feupdateenv(), 956
fflush(), 851
fgetc(), 815
fgetpos(), 845
fgets(), 818
fgetwc(), 946
fgetws(), 946
fibonaci.c, 405
figur.c, 319
File
siehe Datei, 809
FILE Datentyp, 808
Filedeskriptor, 860
fileno(), 872
fillellipse(), 318
fillpoly(), 319
firmaelt.c, 696, 701
firmver2.c, 737
firmverw.c, 710
float Datentyp, 20, 149
float.c, 514, 1023
floataus.c, 41
floatint.c, 141
floor(), 121
FLT_ROUNDS, 151
fma(), 122
fmax(), 122
fmin(), 122
fmod(), 121
Font, 324
fopen(), 809
for, 201
endlos, 219
fordrueb.c, 242
Formaler Parameter, 351, 376
FORTRAN, 464
fprintf(), 820
fputc(), 815
fputs(), 818
fputwc(), 946
fputws(), 946
Fraktale, 341, 406
fread(), 827
free(), 679
free1.c, 679
free2.c, 680
free3.c, 681
freeimage(), 327
Freigabe
Speicher, 659
fremdspe.c, 576
freopen(), 846
freopen.c, 846
frexp(), 121
fscanf(), 820
fseek(), 841
fsetpos(), 845
fstat(), 876
ftell(), 841
funcbez.c, 398
functions, 349
funkproj.c, 338
Funktion, 349
_exit(), 897
abort(), 920
access(), 879
acos(), 121
acosh(), 122
arc(), 317
asctime(), 287
asin(), 121
asinh(), 122
atan(), 121
atan2(), 121
atanh(), 122
atexit(), 898
atof(), 600
atoi(), 599
atol(), 600
atoll(), 600
bar(), 318
bar3d(), 318
bsearch(), 566
btowc(), 950
cabs(), 952
cacos(), 952
cacosh(), 952
calloc(), 671
carg(), 952
casin(), 952
casinh(), 952
catan(), 952
catanh(), 952
cbrt(), 122
ccos(), 952
ccosh(), 952
ceil(), 121
cexp(), 952
chdir(), 882
chmod(), 878
cimag(), 952
circle(), 317
cleardevice(), 310
clearenv(), 903
clearerr(), 814
clearviewport(), 334
clock(), 287
clog(), 952
close(), 863
closegraph(), 305
conj(), 952
copysign(), 122
cos(), 121
cosh(), 121
cpow(), 952
cproj(), 952
creal(), 952
creat(), 862
csin(), 952
csinh(), 952
csqrt(), 952
ctan(), 952
ctanh(), 952
ctime(), 287
Definition (Alt-C), 356
Definition (C89/C99), 353
Deklaration, 360

delay(), 307
difftime(), 287
drawpoly(), 319
ellipse(), 318
Ellipsen-Prototypen, 389
execl(), 907
execle(), 907
execlp(), 907
execv(), 907
execve(), 907
execvp(), 907
exit(), 897
exp(), 121
exp2(), 122
expm1(), 122
fabs(), 121
fclose(), 811
fdim(), 122
fdopen(), 873
feclearexcept(), 955
fegetenv(), 956
fegetexceptflag(), 955
fegetround(), 955
feholdexcept(), 956
feof(), 813
feraiseexcept(), 955
ferror(), 813
fesetenv(), 956
fesetexceptflag(), 955
fesetround(), 956
fetestexcept(), 955
feupdateenv(), 956
fflush(), 851
fgetc(), 815
fgetpos(), 845
fgets(), 818
fgetwc(), 946
fgetws(), 946
fileno(), 872
fillellipse(), 318
fillpoly(), 319
floor(), 121
fma(), 122
fmax(), 122
fmin(), 122
fmod(), 121
fopen(), 809
fprintf(), 820
fputs(), 818
fputwc(), 946
fputws(), 946
fread(), 827
free(), 679
freeimage(), 327
freopen(), 846
frexp(), 121
fscanf(), 820
fseek(), 841
fsetpos(), 845
fstat(), 876
ftell(), 841
fwprintf(), 946
fwrite(), 827
fwscanf(), 946
getc(), 815
getch(), 306
getchar(), 814
getcharacter(), 305
getcolor(), 314
getcwd(), 882
getdouble(), 306
getenv(), 902
getimage(), 327
getint(), 305
getmaxcolor(), 311
getmaxx(), 310
getmaxy(), 310
getopt(), 655
getpixel(), 311
gets, 818
gets(), 604
getwc(), 946
getwchar(), 946
getx(), 310
gety(), 310
gmtime(), 287
hypot(), 122
ilogb(), 122
imaxabs(), 960
imaxdiv(), 960
initgraph(), 305
isalnum(), 93
isalpha(), 93
isblank(), 93
iscntrl(), 93
isdigit(), 93
isgraph(), 93
islower(), 93
isprint(), 93
ispunct(), 93
isspace(), 93
isupper(), 93
iswalnum(), 943
iswalpha(), 943
iswblank(), 943
iswcntrl(), 943
iswdigit(), 943
iswgraph(), 943
iswlower(), 943
iswprint(), 943
iswpunct(), 943
iswspace(), 943
iswupper(), 943
iswxdigit(), 943
isxdigit(), 93
kbhit(), 306
ldexp(), 121
lgamma(), 122
line(), 314
linerel(), 314
lineto(), 315
llrint(), 122
llround(), 123
loadimage(), 327
localeconv(), 937
localtime(), 288
log(), 121
log10(), 121
log1p(), 123
log2(), 123
logb(), 123
longjmp(), 923
lrint(), 123
lround(), 123
lseek(), 868
main(), 894
malloc(), 662
mblen(), 949
mbrlen(), 950
mbrtowc(), 950
mbsinit(), 950
mbsrtowcs(), 951
mbstowcs(), 949
mbtowc(), 949
memchr(), 594
memcmp(), 593
memcpy(), 592
memmove(), 592
memset(), 593
mkdir(), 882
mktime(), 288
modf(), 121
mouse_button(), 334
mouse_getpos(), 334
mouse_hide(), 334
mouse_inwindow(), 335
mouse_left(), 334
mouse_mid(), 334
mouse_right(), 334
mouse_setcursor(), 335
mouse_setpos(), 334
mouse_setwindow(), 335
mouse_show(), 334
mouse_visible(), 334
moverel(), 314
moveto(), 314
nearbyint(), 123
nextafter(), 123
nexttoward(), 123
open(), 860
opendir(), 884
outtextxy(), 307
perror(), 854
pieslice(), 330
pow(), 121
printf(), 820
Prototypen, 361
putchar(), 814
putenv(), 902
putimage(), 327
putpixel(), 311
puts(), 604, 818
putwc(), 946
putwchar(), 946
qsort(), 562, 741
read(), 863
readdir(), 884
realloc(), 672
rectangle(), 317
Rekursiv, 399
remainder(), 123
remove(), 848
remquo(), 123
rename(), 848
rewind(), 846
rewinddir(), 884
rint(), 123
rmdir(), 882
round(), 123
scalbln(), 123
scalbn(), 123

scanf(), 603, 820
sector(), 330
setbuf(), 849
setcolor(), 314
setenv, 902
setfillstyle(), 318
setjmp(), 923
setlinestyle(), 315
setlocale(), 936
setpalette(), 332
setrgbpalette(), 332
settextjustify(), 324
settextstyle(), 324
setvbuf(), 849
setviewport(), 333
setwritemode(), 332
signal(), 914
sin(), 121
sinh(), 121
sleep(), 920
snprintf(), 836
sprintf(), 303, 601, 836
sqrt(), 121
sscanf(), 600, 835
stat(), 876
strcat(), 579
strchr(), 583
strcmp(), 581
strcoll(), 591
strcpy(), 573, 578
strcspn(), 588
strerror(), 590, 855
strftime(), 288
strlen(), 582
strncat(), 580
strncpy(), 578
strpbrk(), 586
strrchr(), 584
strspn(), 587
strstr(), 585
strtod(), 598
strtof(), 599
strtoimax(), 960
strtol(), 594
strtold(), 599
strtoll(), 597
strtoul(), 598
strtoull(), 598
strtoumax(), 960
Struktur, 719
strxfrm(), 591
swprintf(), 946
swscanf(), 946
system(), 905
tan(), 121
tanh(), 121
textheight(), 324
textwidth(), 324
tgamma(), 123
time(), 289
tmpfile(), 852
tmpnam(), 851
tolower(), 93
toupper(), 93
towlower(), 943
towupper(), 943
trunc(), 124
ungetc(), 826
ungetwc(), 946
unlink(), 848
unsetenv(), 902
vfscanf(), 837
vfwprintf(), 946
vfwscanf(), 946
vprintf(), 834
vscanf(), 837
vsnprintf(), 837
vsprintf(), 836
vsscanf(), 837
vswprintf(), 946
vswscanf(), 946
vwprintf(), 946
vwscanf(), 946
wcrtomb(), 951
wcscat(), 947
wcschr(), 947
wcscmp(), 947
wcscoll(), 947
wcscpy(), 947
wcscspn(), 947
wcsftime(), 948
wcslen(), 947
wcsncat(), 947
wcsncmp(), 947
wcsncpy(), 947
wcspbrk(), 947
wcsrchr(), 947
wcsrtombs(), 951
wcsspn(), 947
wcsstr(), 947
wcstod(), 948
wcstof(), 948
wcstoimax(), 960
wcstok(), 947
wcstol(), 948
wcstold(), 948
wcstoll(), 948
wcstombs(), 950
wcstoul(), 948
wcstoull(), 948
wcstoumax(), 960
wcsxfrm(), 947
wctob(), 951
wctomb(), 949
Wide-Character, 942
wmemchr(), 947
wmemcmp(), 947
wmemcpy(), 947
wmemmove(), 947
wmemset(), 947
wprintf(), 946
write(), 866
wscanf(), 946
Adresse, 639
assert(), 495
inline, 396
va_arg() Makro, 390
va_copy() Makro, 390
va_end() Makro, 390
va_start() Makro, 390
Zeiger auf, 415
funkzgr1.c, 418
funkzgr2.c, 419
fwprintf(), 946
fwrite(), 827
fwscanf(), 946

Ganzzahlige Konstanten, 27, 46
Ganzzahltypen
Grenzwerte, 147
geheim.c, 449
geldstap.c, 82, 981
geldstck.c, 128, 983
georeih3.c, 209
georeih4.c, 236
georeih5.c, 237
georeihe.c, 205
gepufferte Ausgabe, 86
gepufferte Eingabe, 86
gerade.c, 509
Geschachtelte Schleifen, 211
getc(), 815
getch(), 306
getchar(), 85, 814
getcharacter(), 305
getcolor(), 314
getcwd(), 882
getdouble(), 306
getenv(), 902
getimage(), 327
getint(), 305
getmaxcolor(), 311
getmaxx(), 310
getmaxy(), 310
getopt(), 655
getpixel(), 311
gets(), 604, 818
getwc(), 946
getwchar(), 946
getx(), 310
gety(), 310
ggtkgv.c, 269
ggtrekur.c, 412, 1016
Gleichheitsoperatoren, 44
gleiteig.c, 155, 987
gleitpkt.c, 786
Gleitpunkt-Ausdruck, 41
Gleitpunkt-Variable, 41
Gleitpunktkonstanten, 27
Gleitpunkttypen
Grenzwerte, 149
Gleitpunktzahlen, 149
gleizae2.c, 241
gleizae3.c, 241
gleizae4.c, 242
gleizaeh.c, 240
gmtime(), 287
goldbac1.c, 284
goldbac2.c, 285
goldbac3.c, 286
Goldbach-Vermutung, 284, 286
goto, 297
Größe von Text, 324
Graphikmodus
ausschalten, 305
einschalten, 305
Grenzwerte
Ganzzahltypen, 147
Gleitpunkttypen, 149
groesse.c, 324, 880
Grunddatentypen, 19

guelt1.c, 440
guelt2.c, 442
guelt3.c, 442

Höhe eines Textes, 324
Höhere Programmiersprache, 464
hamming1.c, 238
hamming2.c, 239
harmon.c, 243, 992
haus.c, 202
Headerdatei, 497
  <assert.h>, 495
  <complex.h>, 951
  <ctype.h>, 93
  <dirent.h>, 884
  <errno.h>, 854
  <fcntl.h>, 861
  <fenv.h>, 955
  <float.h>, 149
  <graphics.h>, 302
  <inttypes.h>, 959
  <iso646.h>, 965
  <limits.h>, 147, 860
  <locale.h>, 935
  <math.h>, 121
  <setjmp.h>, 923
  <signal.h>, 915
  <stdarg.h>, 390
  `<stdbool.h>`, 46
  <stddef.h>, 106, 117
  <stdint.h>, 106, 117, 958
  <stdio.h>, 808
  <stdlib.h>, 594
  <string.h>, 576
  <sys/stat.h>, 877
  <tgmath.h>, 964
  <time.h>, 286
  <wchar.h>, 942
  <wctype.h>, 942
  selbst.h, 97, 98
Headerdateien, 83, 351, 933
Hebräische Methode, 534
hebrae.c, 534
heron.c, 249
heron2.c, 250
heron3.c, 250
Hexadezimal
  Konstanten, 27
  System, 25
  Zahlen, 25
  Ziffer, 192, 195
hexafloat1.c, 153
hexafloat2.c, 155
hexd.c, 829
hexextra.c, 826
hexokdua.c, 214
hexziff1.c, 192
hexziff2.c, 195
hintquad.c, 235
hochdrei.c, 507
hypot(), 122

idealgew.c, 187, 990
if, 159, 166
ilogb(), 122
imaxabs(), 960
imaxdiv(), 960
Implizite Datentypumwandlung, 133, 366
implizite Datentypumwandlung, 131
incpout.c, 870
indatei.c, 816
Information Hidding, 467
iniconst.c, 620
initgraph(), 305
Initialisieren Variable, 75
Initialisierung, 38, 609
  Struktur, 705
inkdek.c, 71, 977
inkrdekr.c, 69
Inkrement-Operator ++, 68
Inline-Funktion, 396
inline.c, 397
int Datentyp, 19
`int`-Ausdruck, 40
`int`-Variable, 40
intaus.c, 40
intcatch.c, 915
inttypes.c, 961
inttypes2.c, 961
invest.c, 529
isalnum(), 93
isalpha(), 93
isblank(), 93
iscntrl(), 93
isdigit(), 93
isgraph(), 93
islower(), 93
isnrlow.c, 587
ISO, 4
ISO C99, 5
isprint(), 93
ispunct(), 93
isspace(), 93
isupper(), 93
iswalnum(), 943
iswalpha(), 943
iswblank(), 943
iswcntrl(), 943
iswdigit(), 943
iswgraph(), 943
iswlower(), 943
iswprint(), 943
iswpunct(), 943
iswspace(), 943
iswupper(), 943
iswxdigit(), 943
isxdigit(), 93

jaeger1.c, 261
jaeger2.c, 262
jmp_buf Datentyp, 924
josephus.c, 779, 1045
just.c, 325

kandproz.c, 145, 986
karten.c, 329
kbhit(), 306
keinadr.c, 118
kettenpi.c, 44, 974
kill Kommando, 917
kleigro1.c, 93
kleigro2.c, 94
kleigro3.c, 97
kleigros.c, 252
koch.c, 346, 414, 1007, 1018
Kochsche Schneeflocke, 341
Komma-Operator, 199, 207
Kommando
  cat, 867
  kill, 917
Kommandozeile, 645, 648, 655
  Argumente, 645
  Optionen, 648, 655
komment.c, 14, 968
Kommentar, 12
  Geschachtelt, 13
Komplement
  Einer, 52
  Zweier, 16, 17, 52
kompzahl.c, 720
Konkatenation von Strings, 109
Konstante
  _IOFBF, 849
  _IOLBF, 849
  _IONBF, 849
  false, 46
  L_tmpnam, 852
  O_APPEND, 861, 866
  O_CREAT, 861
  O_EXCL, 861
  O_RDONLY, 861
  O_RDWR, 861
  O_SYNC, 861, 867
  O_TRUNC, 861
  O_WRONLY, 861
  OPEN_MAX, 860
  S_IRGRP, 862, 878
  S_IROTH, 862, 878
  S_IRUSR, 862, 878
  S_IRWXG, 862, 878
  S_IRWXO, 862, 878
  S_IRWXU, 862, 878
  S_ISGID, 862, 878
  S_ISUID, 862, 878
  S_ISVTX, 862, 878
  S_IWGRP, 862, 878
  S_IWOTH, 862, 878
  S_IWUSR, 862, 878
  S_IXGRP, 862, 878
  S_IXOTH, 862, 878
  S_IXUSR, 862, 878
  SEEK_CUR, 841, 869
  SEEK_END, 841, 869
  SEEK_SET, 841, 869
  SIG_ERR, 914
  stderr, 813, 823
  stdin, 813, 823
  stdout, 813, 823
  TMP_MAX, 852
  true, 46
Konstanten, 26, 27, 77, 79
  char, 26
  Dezimal, 27
  Ganzzahlig, 27, 46
  Gleitpunkt, 27
  Hexadezimal, 27
  Oktal, 27
konto.c, 88
konto2.c, 89

Konvertierung
  Dezimal-in Dualzahl, 18
  Dual-in Hexadzimalzahl, 26
  Dual-in Oktalzahl, 25
Koordinate
  aktuell, 310
  Maximales x, 310
  Maximales y, 310
  Transformation, 337
kosmisch.c, 643, 1037
Kreis
  ausgefüllt zeichnen, 330
  zeichnen, 317
kreis.c, 78, 198, 991
Kreisbogen
  zeichnen, 317
kubikzah.c, 557
Kuchenstück, 330
kugel.c, 80, 979
kugzyvol.c, 415

Lösung
  adturing.c, 1020
  aepfel.c, 993
  anhaeng.c, 1049
  aribit.c, 975
  armstron.c, 995
  basket.c, 1004
  benzinv.c, 980
  bis100.c, 996
  block3.c, 1019
  bruchrec.c, 1041
  bucstrei.c, 1031
  c_worte.c, 1043
  caprecap.c, 982
  chinzahl.c, 1034
  constzei.c, 1020
  datediff.c, 1044
  dauerkal.c, 1034
  demere.c, 1010
  dhondt.c, 1036
  dodon.c, 1025
  domino.c, 992
  dualaddi.c, 1022
  dualwand.c, 1017
  engel.c, 967
  entferbi.c, 1033
  erdumf.c, 979
  faktor.h, 981
  float.c, 1023
  geldstap.c, 981
  geldstck.c, 983
  ggtrekur.c, 1016
  gleiteig.c, 987
  harmon.c, 992
  idealgew.c, 990
  inkdek.c, 977
  josephus.c, 1045
  kandproz.c, 986
  kettenpi.c, 974
  koch.c, 1007, 1018
  komment.c, 968
  kosmisch.c, 1037
  kreis.c, 991
  kugel.c, 979
  laugsaeu.c, 1047
  life.c, 1026
  limits.c, 1023
  logbit.c, 977
  machebi.c, 1032
  meincd.c, 1053
  menue.c, 967
  mywhich.c, 1052
  nulleins.c, 1030
  numausg.c, 1048
  numrueck.c, 1040
  nytimes.c, 974
  okdezhex.c, 983
  pflanze1.c, 1017
  piregen.c, 1002
  polydraw.c, 1015
  primfakt.c, 994
  primzahl.c, 999
  quadzeic.c, 1009
  reaktest.c, 1054
  rechnung.c, 989
  schalt.c, 988
  sincos.c, 984
  sparrate.c, 1012
  sparswei.c, 985
  speikla1.c, 1020
  spiegel.c, 1029
  treesize.c, 1050
  vielmax.c, 1015
  vumrech.c, 981
  wachsen.c, 1026
  wertber.c, 986
  wortlen.c, 1032
  wortstat.c, 1046
  zahlrat.c, 995
  zahlsys.c, 1038
  zeitrech.c, 1013
  ziffadd2.c, 999
  zusop.c, 977
  zusop2.c, 978
L_tmpnam Konstante, 852
laugsaeu.c, 805, 1047
LCGI, 301
ldat.c, 885
ldexp(), 121
Lebensdauer, 431
leerpara.c, 376
Lesbarkeit, 174
Lesen von Directory, 884
lgamma(), 122
life.c, 538, 1026
limits.c, 514, 1023
Lindenmayer-Systeme, 406
line(), 314
linerel(), 314
lineto(), 315
Linie
  absolut zeichnen, 314
  Art, 315
  Dicke, 315
  relativ zeichnen, 314
  zeichnen, 315
linie.c, 315
Linked List, 745, 753, 761
Linker, 471
Linux C Graphics Interface, 301
Liste
  Ausgeben, 753
  Doppelt verkettete, 761
  Einfügen, 753
  Löschen, 753
  Linked, 745, 753, 761
  Operationen, 753
  Sortierte, 753
  Verkettete, 745
llrint(), 122
llround(), 123
loadimage(), 327
localeconv(), 937
localtime(), 288
log(), 121
log10(), 121
log1p(), 123
log2(), 123
logb(), 123
logbit.c, 64, 977
Logische Operatoren, 46
Logischer Operatoren, 47
lokale.c, 939
long Datentyp, 20
long long Datentyp, 20
longjmp(), 923
lottoza2.c, 233
lottoza3.c, 234
lottozah.c, 231
lrint(), 123
lround(), 123
lseek(), 868
lshift.c, 62
lsystem1.c, 408
lsystem2.c, 410

machebi.c, 609, 1032
main(), 894
Makro, 95, 499
Makroaufruf, 95
malloc(), 662
manhat2.c, 224
manhatan.c, 219
mannweib.c, 184
Marke, 297
Maschinensprache, 462
matadd.c, 531
matadd2.c, 558
mathverg.c, 127
Matrix, 527
Matrizen, 527
mauscursor.c, 336
mausdemo.c, 335
Mausprogrammierung, 334
max.c, 502
max2zahl.c, 501
mblen(), 949
mbrlen(), 950
mbrtowc(), 950
mbsinit(), 950
mbsrtowcs(), 951
mbstowcs(), 949
mbtowc(), 949
Mehrdimensionale Arrays, 527
meincd.c, 910, 1053
memchr(), 594
memcmp(), 593
memcpy(), 592
memcpy.c, 592
memmove(), 592

memmove.c, 592
memset(), 593
menue.c, 14, 967
menue1.c, 178
menue2.c, 179
mfunk1.c, 124
mfunk2.c, 125
mineins.c, 176
mineins2.c, 177
minmax.c, 253
mittwert.c, 218
mkdir(), 882
mktime(), 288
modf(), 121
modification time, 881
Modul, 423, 467
  bitumdre.c, 493
  fehler.c, 479
  paritaet.c, 480
  paturing.c, 478
  zahlwort.c, 496
MODULA-2, 465
Modularer Entwurf, 470
Modultechnik, 461, 466
monataus.c, 617
mouse_button(), 334
mouse_getpos(), 334
mouse_hide(), 334
mouse_inwindow(), 335
mouse_left(), 334
mouse_mid(), 334
mouse_right(), 334
mouse_setcursor(), 335
mouse_setpos(), 334
mouse_setwindow(), 335
mouse_show(), 334
mouse_visible(), 334
moverel(), 314
moveto(), 314
mrz_dez.c, 142
mrz_dez2.c, 143
Multibyte-Zeichen, 948
Muster, 318
mycat.c, 867
mywhich.c, 903, 1052

nachvor.c, 87
namdatei.c, 820
namfile.c, 830
namlist1.c, 745
namlist2.c, 753
namlist3.c, 762
namsort1.c, 659
namsort2.c, 663
namsort3.c, 665
namsort4.c, 683
namsort5.c, 684
namsort6.c, 685
namsort7.c, 686
namsort8.c, 686
namsort9.c, 687
Nassi-Shneiderman-Diagramm, 159, 166, 190, 201, 245, 267
NDEBUG, 495
nearbyint(), 123
Negations-Operator, 47
nextafter(), 123
nexttoward(), 123
Nicht-lokaler-Sprung, 923
noexstat.c, 895
nreingab.c, 448
Null-Direktive #, 513
nulleins.c, 569, 1030
numausg.c, 859, 1048
numrueck.c, 688, 1040
numsort1.c, 667
numsort2.c, 672
numsort3.c, 676
numsort4.c, 682
nytimes.c, 43, 974

O_APPEND Konstante, 861, 866
O_CREAT Konstante, 861
O_EXCL Konstante, 861
O_RDONLY Konstante, 861
O_RDWR Konstante, 861
O_SYNC Konstante, 861, 867
O_TRUNC Konstante, 861
O_WRONLY Konstante, 861
oel.c, 311
oelbohr.c, 259
okdezhex.c, 983
Oktal
  Konstanten, 27
  System, 25
  Zahlen, 25
open(), 860
OPEN_MAX Konstante, 860
opendir(), 884
Operation, 200
Operationen mit Zeigern, 551
Operationen und Datentypen, 73
Operator
  <<, 61
  >>, 61
  − >, 725
  | , 52, 55
  ||, 48
  ++, 68
  –, 68
  #, 503
  ##, 503
  &, 52, 53
  &&, 48
  !, 47
  ^, 52, 57
  Bit, 52
  | , 56
  ++, 74
  --, 74
  Assoziativität, 72
  Auswertungszeitupnkt, 74
  cast-, 142
  Dekrement, 68
  Inkrement, 68
  Komma, 199, 207
  Postfix, 68
  Präfix, 68
  Priorität, 71, 186, 200, 1057
  sizeof, 131, 536
  und Datentypen, 73
  AND, 48
  arithmetisch, 39
  Auswertung, 39
  Logischer, 47
  Negations-, 47
  OR, 48
  Pfeil-, 725
  Priorität, 39, 60, 64
  Punkt, 695
  relationaler, 44
  Shift, 61
  Tilde, 52
  Vergleichs-, 44
  Zuweisung, 35
  Zuweisungs-, 65
Operatoren, 39, 44, 47, 48, 52, 61, 65, 68, 71, 72, 186
  Gleichheits-, 44
  Relationale, 44
opshift.c, 66
OR-Operator, 48
outtextxy(), 307

pap.c, 726
Parameter, 351, 376
  Struktur, 719
paritaet.c, 480
PASCAL, 465
paturing.c, 478
perror(), 854
pfeilfix.c, 733
Pfeiloperator, 725
pflanze1.c, 413, 1017
piesarc.c, 331
pieslice(), 330
piregen.c, 344, 1002
Pixel, 311
pixel.c, 312
polydraw.c, 396, 1015
Polygon
  ausgefullt zeichnen, 319
  zeichnen, 319
polygon.c, 322
Positionieren
  absolut, 314
  relativ, 314
Postfix-Operator, 68
postprae.c, 74
potenz.c, 353, 357
potenz3.c, 359
potenz4.c, 360
pow(), 121
Präfix-Operator, 68
Präprozessor, 487
primfakt.c, 264, 994
primzahl.c, 278, 999
printf(), 101, 820, 1059
printf1.c, 103
printf2.c, 105
printf3.c, 106
printf4.c, 107
printf5.c, 109
Priorität, 39, 60, 64, 71, 186, 200, 1057
Privatisierungseffekt, 427
Problemorientierte Sprachen, 464
Programm
  add2.c, 119
  add2alt.c, 365
  add2neu.c, 364

addalt.c, 362
addiere.c, 91
addmult.c, 69
addneu.c, 363
adress1.c, 799
adturing.c, 485, 1020
aepfel.c, 264, 993
anhaeng.c, 873, 1049
arg_env.c, 909
argreih.c, 386
argtest.c, 645
argtest2.c, 646
argtest3.c, 647
argvor.c, 385
aribit.c, 64, 975
armstron.c, 272, 995
arr_swap.c, 625
arryfeh2.c, 525
arryfehl.c, 523
arrygros.c, 522
arryinitc99.c, 612
arrynoname.c, 615
arryok.c, 524
arrysize.c, 536
arrystatic.c, 627
artizahl.c, 717
atexit.c, 898
attribut.c, 879
autoadr.c, 444
autoggt.c, 436
autokauf.c, 162
basket.c, 345, 1004
baum.c, 886
benzinv.c, 81, 980
binatext.c, 838
bis100.c, 278, 996
bitand.c, 54
bitfeld.c, 790
bitfeld2.c, 792
bitgleit.c, 58
bitor.c, 56
bitumdre.c, 493
bitxor.c, 57
block.c, 430
block3.c, 432, 1019
bonbon.c, 247
bonbon2.c, 247
booltyp.c, 49
briefmar.c, 230
bruchrec.c, 704, 1041
bubble.c, 560
buchstat.c, 858
buchstnr.c, 519
buchzaeh.c, 574
bucstrei.c, 608, 1031
bytezahl.c, 816
c_worte.c, 719, 1043
caprecap.c, 112, 982
cast.c, 144
chinzahl.c, 621, 1034
clock1.c, 283
complex.c, 952
const.c, 458
constzei.c, 460, 1020
copyzeit.c, 833
dataus.c, 819
datbytes.c, 842
datediff.c, 722, 1044
dateiart.c, 877
dateiauf.c, 811
datuminfo.c, 292
datzeile.c, 843
datzeit.c, 881
dauerkal.c, 621, 1034
demere.c, 374, 1010
devnr.c, 891
dezumwa2.c, 545
dezumwan.c, 520
dhondt.c, 642, 1036
dirdemo.c, 883
dodon.c, 526, 1025
domino.c, 243, 992
dreichi1.c, 634
dreichi2.c, 638
dreieck.c, 404
dreizeic.c, 86
dualaddi.c, 485, 1022
dualbcd.c, 616
dualwand.c, 413, 1017
dualzahl.c, 217
dynarr.c, 622
e_reihe.c, 455
eifrau1.c, 277
eifrau2.c, 277
eimalei.c, 212
einaus.c, 307
einauscp.c, 823
einkompl.c, 53
einles.c, 433
einzeich.c, 85
Ende, 894, 897
endlos.c, 814
engel.c, 14, 967
englzah1.c, 629
englzah2.c, 631
entferbi.c, 609, 1033
enumdemo.c, 805
Environment, 900
envlist1.c, 900
envlist2.c, 901
erdumf.c, 81, 979
errodemo.c, 855
erst.c, 7
execdemo.c, 909
Exit-Status, 894
exstat.c, 896
faktor.h, 981
fakul.c, 399
fakul2.c, 401
farbstuf.c, 332
fehlausg.c, 394
fehler.c, 856
fehlhand.c, 856
fenv.c, 956
fibonaci.c, 405
figur.c, 319
firmaelt.c, 696, 701
firmver2.c, 737
firmverw.c, 710
float.c, 514, 1023
floataus.c, 41
floatint.c, 141
fordrueb.c, 242
free1.c, 679
free2.c, 680
free3.c, 681
fremdspe.c, 576
freopen.c, 846
funcbez.c, 398
funkproj.c, 338
funkzgr1.c, 418
funkzgr2.c, 419
geheim.c, 449
geldstap.c, 82, 981
geldstck.c, 128, 983
georeih4.c, 236
georeih5.c, 237
georeihe.c, 205
georeihr3.c, 209
gerade.c, 509
ggtkgv.c, 269
ggtrekur.c, 412, 1016
gleiteig.c, 155, 987
gleitpkt.c, 786
gleizae2.c, 241
gleizae3.c, 241
gleizae4.c, 242
gleizaeh.c, 240
goldbac1.c, 284
goldbac2.c, 285
goldbac3.c, 286
groesse.c, 324, 880
guelt1.c, 440
guelt2.c, 442
guelt3.c, 442
hamming1.c, 238
hamming2.c, 239
harmon.c, 243, 992
haus.c, 202
hebrae.c, 534
heron.c, 249
heron2.c, 250
heron3.c, 250
hexafloat1.c, 153
hexafloat2.c, 155
hexd.c, 829
hexextra.c, 826
hexokdua.c, 214
hexziff1.c, 192
hexziff2.c, 195
hintquad.c, 235
hochdrei.c, 507
idealgew.c, 187, 990
inline.c, 397
incpout.c, 870
indatei.c, 816
iniconst.c, 620
inkdek.c, 71, 977
inkrdekr.c, 69
intaus.c, 40
intcatch.c, 915
inttypes.c, 961
inttypes2.c, 961
invest.c, 529
isnrlow.c, 587
jaeger1.c, 261
jaeger2.c, 262
josephus.c, 779, 1045
just.c, 325
kandproz.c, 145, 986
karten.c, 329

keinadr.c, 118
kettenpi.c, 44, 974
kleigro1.c, 93
kleigro2.c, 94
kleigro3.c, 97
kleigros.c, 252
koch.c, 346, 414, 1007, 1018
komment.c, 14, 968
kompzahl.c, 720
konto.c, 88
konto2.c, 89
kosmisch.c, 643, 1037
kreis.c, 78, 198, 991
kubikzah.c, 557
kugel.c, 80, 979
kugzyvol.c, 415
laugsaeu.c, 805, 1047
ldat.c, 885
leerpara.c, 376
life.c, 538, 1026
limits.c, 514, 1023
linie.c, 315
logbit.c, 64, 977
lokale.c, 939
lottoza2.c, 233
lottoza3.c, 234
lottozah.c, 231
lshift.c, 62
lsystem1.c, 408
lsystem2.c, 410
machebi.c, 609, 1032
manhat2.c, 224
manhatan.c, 219
mannweib.c, 184
matadd.c, 531
matadd2.c, 558
mathverg.c, 127
mauscursor.c, 336
mausdemo.c, 335
max.c, 502
max2zahl.c, 501
meincd.c, 910, 1053
memcpy.c, 592
memmove.c, 592
menue.c, 14, 967
menue1.c, 178
menue2.c, 179
mfunk1.c, 124
mfunk2.c, 125
mineins.c, 176
mineins2.c, 177
minmax.c, 253
mittwert.c, 218
monataus.c, 617
mrz_dez.c, 142
mrz_dez2.c, 143
mycat.c, 867
mywhich.c, 903, 1052
nachvor.c, 87
namdatei.c, 820
namfile.c, 830
namlist1.c, 745
namlist2.c, 753
namlist3.c, 762
namsort1.c, 659
namsort2.c, 663
namsort3.c, 665
namsort4.c, 683
namsort5.c, 684
namsort6.c, 685
namsort7.c, 686
namsort8.c, 686
namsort9.c, 687
noexstat.c, 895
nreingab.c, 448
nulleins.c, 569, 1030
numausg.c, 859, 1048
numrueck.c, 688, 1040
numsort1.c, 667
numsort2.c, 672
numsort3.c, 676
numsort4.c, 682
nytimes.c, 43, 974
oel.c, 311
oelbohr.c, 259
okdezhex.c, 983
ophift.c, 66
pap.c, 726
pfeilfix.c, 733
pflanze1.c, 413, 1017
piesarc.c, 331
piregen.c, 344, 1002
pixel.c, 312
polydraw.c, 396, 1015
polygon.c, 322
postprae.c, 74
potenz.c, 353, 357
potenz3.c, 359
potenz4.c, 360
primfakt.c, 264, 994
primzahl.c, 278, 999
printf1.c, 103
printf2.c, 105
printf3.c, 106
printf4.c, 107
printf5.c, 109
qsort1.c, 562
quader.c, 98
quadrat.c, 226
quadrat2.c, 227
quadzahl.c, 548
quadzeic.c, 347, 1009
rabatt.c, 377
rabatt2.c, 378
rdonly.c, 879
reaktest.c, 920, 1054
reaktion.c, 309
realloc.c, 675
rechne1.c, 639
rechne2.c, 640
rechnen.c, 169
rechnen2.c, 185
rechner1.c, 926
rechner2.c, 929
rechnung.c, 181, 989
regel1.c, 366
regel1a.c, 133, 367
regel1b.c, 135
regel1c.c, 135
regel3.c, 136
regel4.c, 138
regel5.c, 140
romzahl.c, 370
rueckwae.c, 402
sandmann.c, 327
scanf1.c, 113
scanf2.c, 115
scanf3.c, 115
scanf4.c, 117
scanf5.c, 120
schalt.c, 181, 988
sincos.c, 129, 984
skatkart.c, 802
skilang.c, 769
sort1.c, 552
sparrate.c, 374, 1012
sparswei.c, 141, 985
speikla1.c, 456, 1020
spiegel.c, 568, 1029
spiel21.c, 575
spieltor.c, 446
sprintf.c, 601
sscanf.c, 600, 835
Start, 894
static1.c, 450
static2.c, 450
stdint.c, 958
strcat.c, 579
strchr.c, 583
strcmp.c, 581
strcpya.c, 572
strcpyz.c, 573
strein1.c, 603
strein2.c, 603
strein3.c, 604
strein4.c, 604
striche.c, 357
strlen.c, 582
strncmp(), 582
strncmp.c, 582
strncpy.c, 578
strrchr.c, 584
strtok(), 589
strtok.c, 589
strtol.c, 595
structinit99.c, 706
structnoname.c, 707
struoper.c, 703
struvararr.c, 782
struzgr1.c, 723
struzgr2.c, 726
struzgr3.c, 736
suchdiv1.c, 563
suchdiv2.c, 565
suchdiv3.c, 567
suchtext.c, 585
systdemo.c, 905
tabelle.c, 678
tausch.c, 36, 379
tausch2.c, 382
tausch3.c, 424
textaus1.c, 110
textaus2.c, 110
tmpnam.c, 852
treesize.c, 892, 1050
typedef1.c, 796
typgroes.c, 132
Umgebung, 900
union.c, 784
unturing.c, 483
va_args.c, 506

vchiffre.c, 372
vergl.c, 864
vertausc.c, 167
vfprintf(), 834
vieladd1.c, 391
vieladd2.c, 391
vieladd3.c, 392
vielmax.c, 396, 1015
vokzaeh2.c, 607
vokzaehl.c, 586, 587
vollkom2.c, 281
vollkomm.c, 279
vormakro.c, 512
vumrech.c, 101, 981
wachsen.c, 526, 1026
wctrans.c, 945
wctype.c, 944
wegein.c, 454
weghaupt.c, 453
weihbaum.c, 227
welchdat.c, 291
welchtag.c, 290
wertber.c, 148, 986
wortlen.c, 608, 1032
wortstat.c, 780, 1046
wurzstruct.c, 708
wurzzahl.c, 618
wz.c, 648
wz2.c, 655
zahlrat.c, 271, 995
zahlsys.c, 657, 1038
zahltab.c, 221
zahltab2.c, 222
zeitrech.c, 388, 1013
zgrarry1.c, 606
zgrarry2.c, 606
ziffadd1.c, 294
ziffadd2.c, 294, 999
zins.c, 387
zufzahl2.c, 256
zufzahl3.c, 257
zufzahl4.c, 258
zusop.c, 67, 977
zusop2.c, 76, 978
zweit.c, 11
Programm beenden, 897
Programmablaufplan
do. . . while, 267
for, 201
if (einsetig), 166
if (zweiseitig), 159
while, 245
programmglobal, 427
Programmiersprache
höhere, 464
problemorientiert, 464
Projektion von Koordinaten, 337
Prototypen, 361
Prozeß, 893
Ende, 894, 897
Environment, 900
Exit-Status, 894
Startup, 894
Umgebung, 900
Prozeduren, 357, 359
Prozedurtechnik, 465
Pseudocode, 180
Puffer
leeren, 851
Pufferung, 848
keine, 849
Voll-, 849
voreingestellt, 849
Zeilen-, 849
Punkt Operator, 695
putc(), 815
putchar(), 814
putenv(), 902
putimage(), 327
putpixel(), 311
puts(), 604, 818
putwc(), 946
putwchar(), 946

qsort(), 562, 741
qsort1.c, 562
quader.c, 98
quadrat.c, 226
quadrat2.c, 227
quadzahl.c, 548
quadzeic.c, 347, 1009

Römische Zahlen, 370
rabatt.c, 377
rabatt2.c, 378
rdonly.c, 879
read(), 863
readdir(), 884
reaktest.c, 920, 1054
reaktion.c, 309
realloc(), 672
realloc.c, 675
Rechenprogramm, 169
rechne1.c, 639
rechne2.c, 640
rechnen.c, 169
rechnen2.c, 185
rechner1.c, 926
rechner2.c, 929
rechnung.c, 181, 989
Rechteck
ausgefüllt zeichnen, 318
zeichnen, 317
rectangle(), 317
regel1.c, 366
regel1a.c, 133, 367
regel1b.c, 135
regel1c.c, 135
regel3.c, 136
regel4.c, 138
regel5.c, 140
register, 455, 931
Rekursive Funktion, 399
Rekursive Strukturen, 745
Relationale Operatoren, 44
Relativ Positionieren, 314
remainder(), 123
remove(), 848
remquo(), 123
rename(), 848
Reservierung
Speicher, 659
restrict, 571
return, 357
rewind(), 846
rewinddir(), 884
rint(), 123
rmdir(), 882
romzahl.c, 370
round(), 123
rueckwae.c, 402

S_IRGRP Konstante, 862, 878
S_IROTH Konstante, 862, 878
S_IRUSR Konstante, 862, 878
S_IRWXG Konstante, 862, 878
S_IRWXO Konstante, 862, 878
S_IRWXU Konstante, 862, 878
S_ISGID Konstante, 862, 878
S_ISUID Konstante, 862, 878
S_ISVTX Konstante, 862, 878
S_IWGRP Konstante, 862, 878
S_IWOTH Konstante, 862, 878
S_IWUSR Konstante, 862, 878
S_IXGRP Konstante, 862, 878
S_IXOTH Konstante, 862, 878
S_IXUSR Konstante, 862, 878
sandmann.c, 327
scalbln(), 123
scalbn(), 123
scanf(), 113, 603, 820, 1061
scanf1.c, 113
scanf2.c, 115
scanf3.c, 115
scanf4.c, 117
scanf5.c, 120
schalt.c, 181, 988
Schnittstellen, 366
Schriftart, 324
Schrittweise Verfeinerung, 470
sector(), 330
SEEK_CUR Konstante, 841, 869
SEEK_END Konstante, 841, 869
SEEK_SET Konstante, 841, 869
selbst.h, 97, 98
self-typing, 26
setbuf(), 849
setcolor(), 314
setenv(), 902
setfillstyle(), 318
setjmp(), 923
setlinestyle(), 315
setlocale(), 936
setpalette(), 332
setrgbpalette(), 332
settextjustify(), 324
settextstyle(), 324
setvbuf(), 849
setviewport(), 333
setwritemode(), 332
Shift-Operator, 61
short Datentyp, 19
SIG_DFL Signal, 914
SIG_ERR Konstante, 914
SIG_IGN Signal, 914
SIGABRT Signal, 917, 920
SIGFPE Signal, 917
SIGILL Signal, 917
SIGINT Signal, 917
Signal
SIG_DFL, 914

SIG_IGN, 914
SIGABRT, 917, 920
SIGFPE, 917
SIGILL, 917
SIGINT, 917
SIGSEGV, 917
SIGTERM, 917
signal(), 914
Signale, 913
Signalkonzept, 913
Signalnamen, 916
Signalnummern, 916
signed Datentyp, 19
SIGSEGV Signal, 917
SIGTERM Signal, 917
sin(), 121
sincos.c, 129, 984
sinh(), 121
sizeof, 131, 536
skatkart.c, 802
skilang.c, 769
sleep(), 920
snprintf(), 836
sort1.c, 552
Sortier-Algorithmus, 560
Sortierte
Liste, 753
sparrate.c, 374, 1012
sparswei.c, 141, 985
Speicher
Dynamische Freigabe, 659
Dynamische Reservierung, 659
Speicherort, 431
speikla1.c, 456, 1020
spiegel.c, 568, 1029
spiel21.c, 575
spieltor.c, 446
Sprache
Assembler, 462
Maschinen-, 462
maschinenorientiert, 462
sprintf(), 303, 601, 836
sprintf.c, 601
Sprung
Nicht-lokal, 923
sqrt(), 121
sscanf(), 600, 835
sscanf.c, 600, 835
Stack, 378, 401, 431
Standard-E/A-Funktionen, 807
Standard-Headerdateien, 498
Standardausgabe, 813, 823, 860
Standardbibliothek, 83, 351
Standardeingabe, 813, 823, 860
Standardfehlerausgabe, 813, 823, 860
Startup-Routine, 894
stat Struktur, 875, 876
stat(), 876
static, 445, 931
static1.c, 450
static2.c, 450
stderr Konstante, 813, 823
stdin Konstante, 813, 823
stdint.c, 958
stdout Konstante, 813
stdout Kontante, 823
Steuerzeichen
\\, 101, 1059
\', 101, 1059
\", 101, 1059
\a, 101, 1059
\b, 101, 1059
\e, 101, 1059
\f, 101, 1059
\ooo, 101, 1059
\r, 101, 1059
\t, 101, 1059
\v, 101, 1059
\x, 101, 1059
strcat(), 579
strcat.c, 579
strchr(), 583
strchr.c, 583
strcmp(), 581
strcmp.c, 581
strcoll(), 591
strcpy(), 573, 578
strcpya.c, 572
strcpyz.c, 573
strcspn(), 588
stream, 807
strein1.c, 603
strein2.c, 603
strein3.c, 604
strein4.c, 604
strerror(), 590, 855
strftime(), 288
striche.c, 357
String
Konkatenation, 109
Lesen (formatiert), 835, 837
Schreiben (formatiert), 836, 837
Stringkonkatenation, 109
Strings, 569
strlen(), 582
strlen.c, 582
strncat(), 580
strncmp(), 582
strncmp.c, 582
strncpy(), 578
strncpy.c, 578
strpbrk(), 586
strrchr(), 584
strrchr.c, 584
strspn(), 587
strstr(), 585
strtod(), 598
strtof(), 599
strtoimax(), 960
strtok(), 589
strtok.c, 589
strtol(), 594
strtol.c, 595
strtold(), 599
strtoll(), 597
strtoul(), 598
strtoull(), 598
strtoumax(), 960
struct, 689
structinit99.c, 706
structnoname.c, 707
Struktogramm
do. . . while-Anweisung, 267
for-Anweisung, 201
if-Aweisung (einseitig), 166
if-Aweisung (zweiseitig), 159
switch-Anweisung, 190
while-Anweisung, 245
Struktur
Bitfelder, 788
Array, 709
Funktion, 719
Initialisierung, 705
rekursiv, 745
Zeiger, 722
Strukturarray, 709
dynamisch, 737
Strukturen, 689
struoper.c, 703
struvararr.c, 782
struzgr1.c, 723
struzgr2.c, 726
struzgr3.c, 736
strxfrm(), 591
Subroutinentechnik, 462
suchdiv1.c, 563
suchdiv2.c, 565
suchdiv3.c, 567
Suche
Binär, 563, 726
suchtext.c, 585
switch, 189
swprintf(), 946
swscanf(), 946
systdemo.c, 905
system(), 905

tabelle.c, 678
tan(), 121
tanh(), 121
tausch.c, 36, 379
tausch2.c, 382
tausch3.c, 424
Technik
Modul-, 466
Prozedur-, 465
Subroutinen-, 462
Unterprogramm, 462
Temporäre Dateien, 851
Text
Breite, 324
Font, 324
Größe, 324
Höhe, 324
Schreibrichtung, 324
Schriftart, 324
Zeichensatz, 324
textaus1.c, 110
textaus2.c, 110
textheight(), 324
Textmodus, 837
textwidth(), 324
tgamma(), 123
Tilde Operator, 52
time(), 289
TMP_MAX Konstante, 852
tmpfile(), 852
tmpnam(), 851
tmpnam.c, 852
tolower(), 93

Top-Down, 470
toupper(), 93
towlower(), 943
towupper(), 943
Transformation von Koordinaten, 337
treesize.c, 892, 1050
TRUE, 46
true Konstante, 46
trunc(), 124
turing.h, 476
Turingmaschinen, 472
Typ-Qualifizierer, 457
typedef, 795
typedef1.c, 796
typgroes.c, 132

Umgebung, 900
Umlenkung
  E/A, 823
Umwandlung
  Dezimal in Dualzahl, 18
  Dual-in Hexadzimalzahl, 26
  Dual-in Oktalzahl, 25
  Gleitpunkt in Dualzahl, 152
ungetc(), 826
ungetwc(), 946
uninitialisierte Variable, 75
union, 783
union.c, 784
Unions, 783
unlink(), 848
unsetenv(), 902
unsigned Datentyp, 19, 20
Unterprogrammtechnik, 462
unturing.c, 483

va_arg(), 390
va_args.c, 506
va_copy(), 390
va_end(), 390
va_list, 390
va_start(), 390
Variable initialisieren, 75
Variablen, 29
  Definition, 32
  Deklaration, 32
  Initialisierung, 38
  Name, 30
  nicht vorbesetzt, 75
  uninitialisiert, 75
  Vereinbarung, 32
  Vertauschen, 36, 167
vchiffre.c, 372
Vektor, 517
vergl.c, 864
Vergleichsoperatoren, 44
Verkettete Liste, 745, 761
Verschiebechiffre, 372
vertausc.c, 167
Vertauschen von Variablen, 36, 167
Verzeichnis, 875, 882
vfprintf(), 834
vfscanf(), 837
vfwprintf(), 946
vfwscanf(), 946
vieladd1.c, 391
vieladd2.c, 391
vieladd3.c, 392
vielmax.c, 396, 1015
void, 357, 376
vokzaeh2.c, 607
vokzaehl.c, 586, 587
volatile, 457, 459, 931
Voll-Pufferung, 849
vollkom2.c, 281
vollkomm.c, 279
Vollkommene Zahlen, 279, 281
Vordefinierte Makros, 511
Voreingestellte Pufferung, 849
vormakro.c, 512
vprintf(), 834
vscanf(), 837
vsnprintf(), 837
vsprintf(), 836
vsscanf(), 837
vswprintf(), 946
vswscanf(), 946
vumrech.c, 101, 981
vwprintf(), 946
vwscanf(), 946

wachsen.c, 526, 1026
wcrtomb(), 951
wcscat(), 947
wcschr(), 947
wcscmp(), 947
wcscoll(), 947
wcscpy(), 947
wcscspn(), 947
wcsftime(), 948
wcslen(), 947
wcsncat(), 947
wcsncmp(), 947
wcsncpy(), 947
wcspbrk(), 947
wcsrtombs(), 951
wcsspn(), 947
wcsstr(), 947
wcstod(), 948
wcstof(), 948
wcstoimax(), 960
wcstok(), 947
wcstol(), 948
wcstold(), 948
wcstoll(), 948
wcstombs(), 950
wcstoul(), 948
wcstoull(), 948
wcstoumax(), 960
wcsxfrm(), 947
wctob(), 951
wctomb(), 949
wctrans.c, 945
wctype.c, 944
wegein.c, 454
weghaupt.c, 453
weihbaum.c, 227
welchdat.c, 291
welchtag.c, 290
wertber.c, 148, 986
Wertebereich von Datentypen, 21, 1058
while, 245
Wide-Character-Funktionen, 942
wmemchr(), 947
wmemcmp(), 947
wmemcpy(), 947
wmemmove(), 947
wmemset(), 947
Wochentag eines Datums, 290
Working-Directory, 882
wortlen.c, 608, 1032
wortstat.c, 780, 1046
wprintf(), 946
write(), 866
wscanf(), 946
wurzstruct.c, 708
wurzzahl.c, 618
wz.c, 648
wz2.c, 655

Zahlen
  zu große, 22
  zu kleine, 22
  Hexadezimale, 25
  Oktale, 25
zahlrat.c, 271, 995
zahlsys.c, 657, 1038
zahltab.c, 221
zahltab2.c, 222
zahlwort.c, 496
Zeichenfarbe
  erfragen, 314
  festlegen, 314
Zeichensatz, 324
Zeichnen
  zweidimensionalen Balken, 318
  ausgefüllte Ellipse, 318, 330
  ausgefüllter Kreis, 330
  ausgefülltes Polygon, 319
  ausgefülltes Rechteck, 318
  dreidimansionalen Balken, 318
  Ellipse, 318
  Kreis, 317
  Kreisbogen, 317
  Linie, 314, 315
  Polygon, 319
  Rechteck, 317
Zeiger
  auf Funktionen, 415
  auf Zeiger, 623
  char, 569
  Funktionsadressen, 639
  Operationen mit, 551
  Pfeiloperator, 725
  Struktur, 722
  und Arrays, 539
Zeigerarrays, 623
Zeilen-Pufferung, 849
Zeilenummerierung, 512
Zeitangaben, 286
Zeiten einer Datei, 881
zeitrech.c, 388, 1013
zgrarry1.c, 606
zgrarry2.c, 606
ziffadd1.c, 294
ziffadd2.c, 294, 999
zins.c, 387
zu große Zahlen, 22

zu kleine Zahlen, 22
zufzahl2.c, 256
zufzahl3.c, 257
zufzahl4.c, 258
Zugriffrechte, 878
zusop.c, 67, 977
zusop2.c, 76, 978
Zuweisung, 35
Zuweisungsoperator, 35, 65
zweidimensionale Arrays, 527
Zweier-Komplement, 16, 17, 52
Zweiseitige if-Anweisung, 159
zweit.c, 11