

Inhaltsverzeichnis

| | |
|---|-----------|
| Über den Autor | 9 |
| Einführung | 23 |
| Zu diesem Buch | 23 |
| Icons in diesem Buch | 26 |
| Weiteres Material zum Buch | 26 |
| Wie geht's weiter? | 27 |
| | |
| Teil I | |
| Einführung in die C++-Programmierung | 29 |
| Kapitel 1 | |
| Ihr erstes C++-Programm | 31 |
| Was ist ein Programm? | 31 |
| Code::Blocks installieren | 33 |
| Windows | 33 |
| Ubuntu Linux | 36 |
| Macintosh | 37 |
| Ihr erstes C++-Programm erstellen | 41 |
| Ein Projekt anlegen | 41 |
| Den C++-Code eingeben | 43 |
| Schummeln | 45 |
| Ihr Programm erstellen | 46 |
| Ihr Programm ausführen | 47 |
| Kommentierte Programme lesen | 48 |
| Das gemeinsame Gerüst aller C++-Programme | 48 |
| Sourcecode durch Kommentare erläutern | 49 |
| Programme bestehen aus C++-Anweisungen | 50 |
| Deklarationen schreiben | 51 |
| Ausgaben erzeugen | 51 |
| Ausdrücke berechnen | 52 |
| Das Ergebnis eines Ausdrucks speichern | 52 |
| Den Rest des Programms untersuchen | 52 |

Kapitel 2**Variablen deklarieren**

| | |
|--|----|
| Variablen deklarieren | 55 |
| Verschiedene Variablentypen deklarieren | 56 |
| Überblick über die Einschränkungen der Ganzzahlen in C++ | 57 |
| Das Rundungsproblem lösen | 58 |
| Die Einschränkungen von Fließkommazahlen | 59 |
| Variablentypen deklarieren | 60 |
| Konstantentypen | 62 |
| Wertebereiche von numerischen Typen | 63 |
| Sonderzeichen | 64 |
| Zeichen mit doppelter Spurbreite | 65 |
| Sind diese Berechnungen wirklich logisch? | 66 |
| Mixed-Mode-Ausdrücke | 66 |
| Automatische Deklarationen | 68 |

Kapitel 3**Mathematische Operationen ausführen**

| | |
|---|----|
| Einfache binäre Operationen ausführen | 69 |
| Ausdrücke zerlegen | 71 |
| Die Reihenfolge von Operationen bestimmen | 71 |
| Unäre Operationen ausführen | 72 |
| Mit Zuweisungsoperatoren arbeiten | 74 |

Kapitel 4**Logische Operationen ausführen**

| | |
|--|----|
| Warum brauchen wir überhaupt logische Operationen? | 77 |
| Mit den einfachen logischen Operatoren arbeiten | 78 |
| Logische Werte speichern | 79 |
| Mit logischen int-Variablen arbeiten | 81 |
| Vorsicht bei logischen Operationen mit Fließkommavariablen | 81 |
| Binärzahlen repräsentieren | 83 |
| Das dezimale Zahlensystem | 83 |
| Andere Zahlensysteme | 83 |
| Das binäre Zahlensystem | 84 |
| Bitweise logische Operationen ausführen | 85 |
| Die bitweisen Operatoren | 86 |
| Die bitweisen Operatoren verwenden | 87 |
| Ein einfacher Test | 87 |

Kapitel 5**Den Programmablauf steuern**

| | |
|---|-----|
| Den Programmablauf mit Verzweigungsbefehlen steuern | 91 |
| Schleifen in einem Programm ausführen | 93 |
| Eine Schleife ausführen, während eine Bedingung true ist | 94 |
| Die Autoinkrement- und Autodekrement-Funktionen verwenden | 96 |
| Mit der for-Schleife arbeiten | 97 |
| Die gefürchtete Endlosschleife vermeiden | 100 |
| Bereichsbasierte for-Schleifen | 101 |
| Spezielle Schleifenkontrollen anwenden | 102 |
| Steuerbefehle verschachteln | 105 |
| Die Mehrwege-Verzweigung: »switch« | 107 |

Teil II**Ein brauchbarer C++-Programmierer werden**

109

Kapitel 6**Funktionen erstellen**

111

| | |
|--|-----|
| Eine Funktion schreiben und verwenden | 111 |
| Unsere erste Funktion definieren | 114 |
| Die Funktion sumSequence() definieren | 114 |
| Die Funktion sumSequence() aufrufen | 114 |
| Teile und herrsche | 115 |
| Die Details von Funktionen verstehen | 115 |
| Einfache Funktionen verstehen | 116 |
| Funktionen mit Argumenten verstehen | 117 |
| Funktionsnamen überladen | 120 |
| Funktionsprototypen definieren | 121 |
| Standardargumente | 123 |
| Argumente als Wert oder als Referenz übergeben | 124 |
| Variable Speichertypen | 126 |

Kapitel 7**Sequenzen in Arrays speichern**

127

| | |
|---|-----|
| Wozu werden Arrays benötigt? | 127 |
| Ein Array verwenden | 129 |
| Ein Array initialisieren | 132 |
| Mit einem zu großen Indexwert auf ein Array zugreifen | 133 |
| Arrays und bereichsbasierte for-Schleifen | 133 |
| Arrays von Arrays definieren und verwenden | 134 |
| Arrays von Zeichen verwenden | 135 |
| Ein Array von Zeichen erstellen | 135 |
| Einen String von Zeichen erstellen | 136 |

| | |
|--------------------------------|-----|
| Zeichenstrings manipulieren | 138 |
| Ein paar Bibliotheksfunktionen | 140 |
| Mehr Platz für Wide-Strings | 142 |

Kapitel 8**Ein erster Blick auf C++-Zeiger**

| | |
|---|-----|
| Variablengröße | 145 |
| Was ist in einer Adresse enthalten? | 147 |
| Adressoperatoren | 147 |
| Zeigervariablen verwenden | 149 |
| Verschiedene Typen von Zeigern verwenden | 150 |
| Zeiger an Funktionen übergeben | 151 |
| Übergabe als Wert | 151 |
| Zeigerwert übergeben | 152 |
| Übergabe per Referenz | 152 |
| Konstante Irritationen | 153 |
| Einen Speicherblock namens Heap nutzen | 155 |
| Begrenzter Gültigkeitsbereich | 156 |
| Das Problem des Gültigkeitsbereichs untersuchen | 157 |
| Mit dem Heap eine Lösung finden | 158 |

Kapitel 9**Ein zweiter Blick auf C++-Zeiger**

| | |
|--|-----|
| Operationen für Zeigervariablen definieren | 161 |
| Arrays im Licht von Zeigervariablen noch einmal betrachten | 162 |
| Operatoren auf die Adresse eines Arrays anwenden | 163 |
| Zeigeroperationen auf einen String ausdehnen | 165 |
| Gründe für die zeigerbasierte String-Manipulation | 167 |
| Operatoren auf andere Zeigertypen als char anwenden | 167 |
| Einen Zeiger mit einem Array vergleichen | 168 |
| Wann gibt's einen Zeiger nicht? | 170 |
| Arrays von Zeigern deklarieren und verwenden | 171 |
| Arrays von Zeichenstrings nutzen | 172 |
| Auf die Argumente von main() zugreifen | 174 |

Kapitel 10**Der C++-Präprozessor**

| | |
|---------------------------------------|-----|
| Was ist ein Präprozessor? | 179 |
| Einbinden von Dateien | 180 |
| Die #Definition von Dingen | 183 |
| Und wenn ich nichts #definieren will? | 185 |
| Ein paar Alternativen aufzählen | 187 |
| Dinge mit einem »ja aber« einbinden | 188 |
| Vordefinierte Objekte | 189 |
| Typendefinitionen mit typedef | 191 |

Teil III**Eine Einführung in Klassen****193****Kapitel 11****Ein Blick auf die objektorientierte Programmierung****195**

| | |
|---|-----|
| Die Abstraktion und der Mikrowellenherd | 195 |
| Nachos prozedural zubereiten | 196 |
| Nachos objektorientiert zubereiten | 197 |
| Klassenbildung und Mikrowellen | 197 |
| Wozu Klassen bilden? | 198 |

Kapitel 12**Klassen in C++****201**

| | |
|---|-----|
| Einführung der Klasse | 201 |
| Das Format einer Klasse | 201 |
| Auf die Elemente einer Klasse zugreifen | 202 |
| Objekte aktivieren | 203 |
| Reale Objekte simulieren | 203 |
| Wozu dienen Elementfunktionen? | 204 |
| Eine Elementfunktion hinzufügen | 205 |
| Eine Elementfunktion aufrufen | 206 |
| Von einer Elementfunktion auf andere Elemente zugreifen | 209 |
| Auflösung des Gültigkeitsbereichs | 209 |
| Eine Elementfunktion in der Klasse definieren | 211 |
| Elementfunktionen separat definieren | 213 |
| Elementfunktionen überladen | 215 |

Kapitel 13**Objekte mit Zeigern manipulieren****217**

| | |
|---|-----|
| Arrays aus Objekten deklarieren | 217 |
| Zeiger auf Objekte deklarieren | 218 |
| Einen Objektzeiger dereferenzieren | 219 |
| Der Pfeiloperator | 220 |
| Objekte an Funktionen übergeben | 221 |
| Eine Funktion mit einem Objektwert aufrufen | 221 |
| Eine Funktion mit einem Objektzeiger aufrufen | 222 |
| Eine Funktion mit dem Referenzoperator aufrufen | 224 |
| Wozu Zeiger oder Referenzen? | 225 |
| Zurück zum Heap | 226 |
| Viele Objekte auf dem Heap anlegen | 227 |
| Wenn der Speicher für Sie reserviert wird | 227 |
| Verknüpfungen mit verketteten Listen erstellen | 228 |
| Andere Operationen auf eine verkettete Liste anwenden | 229 |
| Das LinkedListData-Beispielprogramm | 230 |

| | |
|--|-----|
| Ein Hoffnungsstrahl: Eine Liste von Containern, die mit der C++-Library verknüpft sind | 233 |
|--|-----|

Kapitel 14**Nicht stören: Elemente schützen**

| | |
|---|-----|
| Elemente schützen | 235 |
| Wozu benötigen Sie geschützte Elemente? | 235 |
| Wie geschützte Elemente funktionieren | 236 |
| Argumente für den Einsatz geschützter Elemente | 237 |
| Den internen Status einer Klasse schützen | 237 |
| Eine Klasse mit eingeschränkter Schnittstelle verwenden | 238 |
| Nicht-Elementfunktionen Zugriff auf geschützte Elemente geben | 239 |

Kapitel 15**Objekte konstruieren und zerstören**

| | |
|------------------------------------|-----|
| Objekte erstellen | 243 |
| Konstruktoren verwenden | 244 |
| Ein einzelnes Objekt konstruieren | 244 |
| Mehrere Objekte konstruieren | 246 |
| Ein Duplex konstruieren | 247 |
| Einen Destruktor analysieren | 249 |
| Warum Sie den Destruktor benötigen | 249 |
| Mit Destruktoren arbeiten | 249 |

Kapitel 16**Konstruktoren mit Argumenten**

| | |
|--|-----|
| Konstruktoren mit Argumenten ausstatten | 255 |
| Einen Konstruktor verwenden | 256 |
| Konstruktoren überladen | 258 |
| Standardmäßige Standardkonstruktoren | 262 |
| Klassenelemente konstruieren | 263 |
| Ein komplexes Datenelement konstruieren | 264 |
| Elementinitialisierung und Konstruktor im Wettbewerb | 268 |
| Ein konstantes Datenelement konstruieren | 270 |
| Die Reihenfolge der Konstruktion | 270 |
| Lokale Objekte werden der Reihenfolge nach konstruiert | 271 |
| Statische Objekte werden nur einmal konstruiert | 271 |
| Alle globalen Objekte werden vor main() konstruiert | 273 |
| Globale Objekte werden in keiner bestimmten Reihenfolge konstruiert | 273 |
| Elemente werden in der Reihenfolge konstruiert, in der sie deklariert werden | 274 |
| Destruktoren werden in der umgekehrten Reihenfolge der Konstruktionen aufgerufen | 275 |

| | |
|---|------------|
| Arrays konstruieren | 275 |
| Konstruktoren als Form der Konvertierung | 276 |
| Kapitel 17 | |
| Der Copy/Move-Konstruktor | 279 |
| Ein Objekt kopieren | 279 |
| Warum Sie Copy-Konstruktoren brauchen | 279 |
| Die Copy-Konstruktoren verwenden | 280 |
| Der automatische Copy-Konstruktor | 282 |
| Flache oder tiefe Kopien erstellen | 284 |
| Temporäre Objekte | 288 |
| Temporäre Objekte dauerhaft vermeiden | 290 |
| Der Move-Konstruktor | 291 |
| Kapitel 18 | |
| Statische Elemente | 295 |
| Ein statisches Element definieren | 295 |
| Warum Sie statische Elemente benötigen | 295 |
| Statische Elemente verwenden | 296 |
| Statische Datenelemente referenzieren | 297 |
| Verwendungsmöglichkeiten statischer Datenelemente | 298 |
| Statische Elementfunktionen deklarieren | 299 |
| Was ist überhaupt this? | 301 |
| Teil IV | |
| Vererbung | 305 |
| Kapitel 19 | |
| Eine Klasse erben | 307 |
| Brauche ich Vererbung? | 308 |
| Wie funktioniert die Vererbung einer Klasse? | 309 |
| Eine Unterklassie verwenden | 311 |
| Eine Unterklassie konstruieren | 312 |
| Eine Unterklassie zerstören | 313 |
| Konstruktoren erben | 313 |
| Die HAS_A-Beziehung | 314 |
| Kapitel 20 | |
| Virtuelle Elementfunktionen | 315 |
| Warum Sie Polymorphismus brauchen | 318 |
| Wie funktioniert Polymorphismus? | 319 |
| Wann ist eine Funktion virtuell und wann nicht? | 320 |
| Virtuelle Betrachtungen | 322 |

| | | |
|---|--|------------|
| Kapitel 21 | | |
| Klassen faktorisieren | | 325 |
| Das Faktorisieren | | 325 |
| Abstrakte Klassen implementieren | | 329 |
| Das Konzept der abstrakten Klasse | | 330 |
| Wie man aus einer abstrakten Klasse eine wirkliche Klasse macht | | 331 |
| Abstrakte Klassen übergeben | | 331 |
| Teil V | | |
| Optionale Funktionen und Sicherheit | | 333 |
| Kapitel 22 | | |
| Ein neuer Zuweisungsoperator | | 335 |
| Operatoren mit Funktionen vergleichen | | 335 |
| Einen neuen Operator einfügen | | 336 |
| Flache Kopien zu erstellen ist ein tiefes Problem | | 336 |
| Den Zuweisungsoperator überladen | | 338 |
| Den Indexoperator überladen | | 343 |
| Der Move-Konstruktor und der Move-Zuweisungsoperator | | 343 |
| Kapitel 23 | | |
| Mit Stream-I/O arbeiten | | 351 |
| Wie funktioniert Stream-I/O? | | 347 |
| Default-Stream-Objekte | | 348 |
| Die fstream-Unterklassen | | 349 |
| Vielfältiges Öffnen | | 350 |
| Datei, was machst Du gerade? | | 351 |
| Kann ich ein Beispiel sehen? | | 352 |
| Andere Methoden der Stream-Klassen | | 355 |
| Streams direkt lesen und schreiben | | 357 |
| Das Format kontrollieren | | 358 |
| Was ist mit endl? | | 360 |
| Positionierung des Zeigers innerhalb einer Datei | | 361 |
| Die stringstream-Unterklassen verwenden | | 361 |
| Manipulatoren manipulieren | | 364 |
| Kapitel 24 | | |
| Fehler verarbeiten – Ausnahmen | | 367 |
| Warum brauche ich einen neuen Fehlermechanismus? | | 369 |
| Den Ausnahmemechanismus untersuchen | | 370 |
| Welche Dinge sind »throw«-fähig? | | 373 |
| Einfach mal durchreichen | | 376 |

| | |
|---|------------|
| Kapitel 25 | |
| Mehrfachvererbung | 377 |
| Wie funktioniert die Mehrfachvererbung? | 377 |
| Vererbungsmehrdeutigkeiten beseitigen | 379 |
| Virtuelle Vererbung hinzufügen | 380 |
| Objekte der Mehrfachvererbung erzeugen | 386 |
| Eine gegenteilige Meinung | 387 |
| Kapitel 26 | |
| C++-Templates durchleuchten | 389 |
| Eine Funktion in einem Template verallgemeinern | 390 |
| Klassentemplates | 392 |
| Tipps für die Verwendung von Templates | 396 |
| Externe Template-Instanziierungen | 396 |
| Eine Initialisierungsliste implementieren | 397 |
| Kapitel 27 | |
| Die Standard Template Library als Standard verwenden | 399 |
| Der string-Container | 400 |
| Die list-Container | 404 |
| Listen durchwandern | 405 |
| Operationen auf ganzen Listen | 407 |
| Beispiel, bitte! | 407 |
| Kapitel 28 | |
| Den Code vor Hackern schützen | 411 |
| Die Motive des Hackers verstehen | 411 |
| Code-Einschleusung verstehen | 414 |
| Ein Beispiel für SQL-Injection untersuchen | 414 |
| Code-Injection unterbinden | 416 |
| Puffer aus Spaß und aus Gewinnsucht zum Überlauf bringen | 417 |
| Beispiel, bitte! | 417 |
| Einen Funktionsaufruf aufstapeln | 420 |
| BufferOverflow hacken | 424 |
| Pufferüberlauf vermeiden – erster Ansatz | 428 |
| Pufferüberlauf vermeiden – zweiter Ansatz | 431 |
| Ein weiterer Grund für die string-Klasse | 434 |
| Warum nicht immer string-Funktionen verwenden? | 436 |

Teil VI**Der Top-Ten-Teil****439****Kapitel 29****Zehn Methoden, um Fehler in einem Programm zu vermeiden****441**

| | |
|--|-----|
| Aktivieren Sie alle Warnungen und Fehlermeldungen | 441 |
| Pflegen Sie einen sauberen und konsistenten Programmierstil | 442 |
| Schränken Sie die Sichtbarkeitsbereiche ein | 442 |
| Kommentieren Sie Ihren Code | 444 |
| Durchlaufen Sie jeden Pfad mindestens einmal im Einzelschrittmodus | 444 |
| Vermeiden Sie es, Operatoren zu überladen | 445 |
| Mit dem Heap arbeiten | 445 |
| Fehler mit Ausnahmen behandeln | 445 |
| Deklarieren Sie Destruktoren virtuell | 446 |
| Vermeiden Sie die Mehrfachvererbung | 447 |

Kapitel 30**Zehn Methoden, um Programme vor Hackern zu schützen****449**

| | |
|---|-----|
| Machen Sie keine Annahmen über die Benutzereingaben | 449 |
| Gehen Sie charmant mit Scheitern um | 450 |
| Erstellen Sie ein Programmprotokoll | 451 |
| Setzen Sie einen klugen Entwicklungsprozess ein | 453 |
| Implementieren Sie eine gute Versionsverwaltung | 453 |
| Authentifizieren Sie Ihre Benutzer sicher | 455 |
| Verwalten Sie Remotesitzungen | 457 |
| Verschleiern Sie Ihren Code | 458 |
| Signieren Sie Ihren Code mit einem digitalen Zertifikat | 461 |
| Verwenden Sie wo immer nötig sichere Verschlüsselung | 462 |

Stichwortverzeichnis**465**