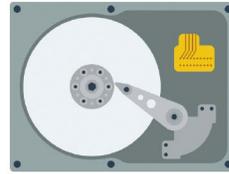


Thomas Theis



DATENSTRÖME

DATENTYPEN



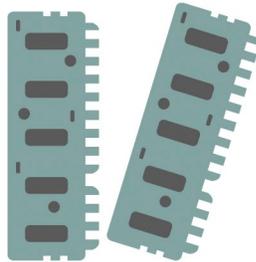
DATENBANKEN

Einstieg in C++

Ideal zum Programmieren lernen



STREAMS



STRUKTUREN



FUNKTIONEN

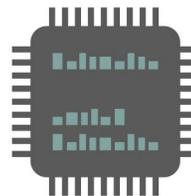


DATENAUSTAUSCH



OBJEKTE

VERERBUNG



FEHLERBEHANDLUNG



CONTAINER

Ohne
Vorkenntnisse
einsteigen

- ▶ C++-Programmierung verständlich erklärt
- ▶ Alle Sprachgrundlagen und wichtigen Programmier Techniken
- ▶ Mit Übungsaufgaben und Musterlösungen



Alle Beispielprojekte zum Download



Rheinwerk
Computing

Kapitel 1

Eine erste Einführung

Die Sprache C++ ist weit verbreitet, wird vielfältig genutzt und zählt schon lange zu den wichtigsten Programmiersprachen.

Nachdem wir geklärt haben, wie wir mit C++ arbeiten und welcher C++-Standard für uns wichtig ist, schreiben wir bereits das erste Programm.

1.1 Was machen wir mit C++?

Wir werden mit der Sprache C++ objektorientiert programmieren, also mit *Klassen, Objekten, Eigenschaften* und *Methoden* und nach den Konzepten der *Datenkapselung*, der *Vererbung* und der *Polymorphie*.

Objektorientierung

Als Vorbereitung dazu werden wir mit C++ zunächst klassisch prozedural arbeiten, also mit *Datentypen, Operatoren, Kontrollstrukturen* und *Funktionen* starten. Auf diese Weise erhalten Sie eine solide Grundlage, die Ihnen anschließend den Einstieg in die Objektorientierung erleichtert. Alle genannten Begriffe lernen Sie ausführlich kennen.

Grundlagen

Das Verständnis der Theorie wird Ihnen anhand von circa 200 vollständigen lauffähigen Beispielprogrammen erleichtert. Mithilfe von vielen eingestreuten Übungsaufgaben können Sie Ihren wachsenden Kenntnisstand prüfen. Alle Beispielprogramme und alle Lösungen zu den Übungsaufgaben finden Sie im Downloadbereich zum Buch unter der Adresse www.rheinwerk-verlag.de/5179.

Beispiele und
Übungen

Es gibt zahlreiche Bibliotheken, die die Möglichkeiten von C++ erweitern. Die Sprache C++ ist ursprünglich als Weiterentwicklung der Sprache C entstanden. Aufgrund der Verwandtschaft ist es auch in C++ möglich, hardware-orientierte schnelle Programme zu entwickeln.

Bibliotheken

1.2 Was benötige ich zum Programmieren?

Eingabe, Terminal Sie können Ihre C++-Programme bereits mit einem einfachen Texteditor erstellen, sie mithilfe des Befehls `g++` übersetzen und anschließend ausführen. Unter *Windows* machen Sie das in der *Eingabeaufforderung*, unter *Ubuntu Linux* und unter *macOS* in einem *Terminal*. Die dazu notwendigen Schritte werden in Anhang A, »Installationen«, beschrieben. Viele Texteditoren bieten ein Syntax-Highlighting für C++, also ein farbliches Hervorheben der Sprachelemente.

IDE Wesentlich mehr Komfort bietet eine integrierte Entwicklungsumgebung (engl.: *Integrated Development Environment*, kurz IDE). Für die Betriebssysteme *Windows*, *Ubuntu Linux* und *macOS* gibt es eine Reihe von frei verfügbaren IDEs, die u. a. Folgendes bieten:

- Editor** ▶ einen *Editor*, mit dessen Hilfe Sie den Programmcode schreiben
- Compiler** ▶ einen *Compiler*, mit dessen Hilfe Sie den Programmcode in die Sprache übersetzen, die der jeweilige Rechner versteht
- Debugger** ▶ einen *Debugger*, der Ihnen bei der Fehlersuche hilft

Die Installation und die Nutzung verschiedener IDEs, z. B. von *Eclipse*, *Qt Creator*, *Xcode*, *Code::Blocks* und *Orwell Dev C++* wird ebenfalls in Anhang A, »Installationen«, beschrieben.

1.3 Die Entwicklung von C++

Standards Die Sprache C++ wurde im Jahr 1979 von Bjarne Stroustrup entwickelt. Über die Jahre hinweg hat C++ viele Neuerungen erfahren. Im Jahr 1998 erschien die standardisierte Version C++98. Weitere Verbesserungen sind mit den Standards C++03, C++11, C++14 und C++17 in den Jahren 2003, 2011, 2014 und 2017 eingeflossen.

C++20 Aktuell (im Mai 2020) ist die Entwicklung der Version C++20 bereits weit fortgeschritten. Der zugehörige Entwurf C++2a, der bereits einige Möglichkeiten von C++20 bietet, kann z. B. in den folgenden Konstellationen genutzt werden:

- ▶ mit dem *GCC-Compiler* in der Version 9.3.0 in einem Terminal unter *Ubuntu Linux*
- ▶ mit der Programmsammlung *MinGW*, die den *GCC-Compiler* in der Version 9.2.0 nutzt, in einer Eingabeaufforderung unter *Windows*

- ▶ mit dem *Clang-Compiler* in der Version 9.1.0 in einem Terminal unter *macOS*

Die Programme dieses Buchs wurden primär unter diesen drei Umgebungen entwickelt und getestet. Lassen Sie sich nicht irritieren, falls vereinzelt eine Warnung beim Bearbeiten eines Programms in einer IDE erscheint: Nicht alle IDEs setzen die C++-Standards zu 100 % um.

Falls Sie weitere Informationen zur Sprache C++ und der Zuordnung ihrer Elemente zu den verschiedenen Versionen benötigen, empfehle ich die C++-Referenz unter der folgenden Adresse: <http://cppreference.com>.

C++-Referenz

1.4 So sieht das erste Programm aus

In diesem Abschnitt lernen Sie das erste C++-Programm kennen. Es geht zunächst nur um das Verständnis für den Aufbau des Programms. Es wird eine Reihe von Begriffen eingeführt. Ihre Bedeutung wird erläutert, zudem werden sie Ihnen im Verlauf des Buchs immer vertrauter.

Aufbau

Das nachfolgende Programm gibt den Text *Hallo Welt* auf dem Bildschirm aus, gefolgt von einem Zeilenumbruch:

Hallo Welt

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hallo Welt" << endl;
}
```

Listing 1.1 Datei »hallo.cpp«

Die entscheidende Zeile des Programms beginnt mit `cout`. Das Objekt `cout` dient zur Ausgabe von Daten auf dem Bildschirm. Der Operator `<<` versorgt das Objekt `cout` mit den Informationen, die angezeigt werden.

cout

Zunächst wird der Text *Hallo Welt* ausgegeben. Texte werden in doppelten Anführungsstrichen notiert. Nach dem Text folgt der *Manipulator* `endl`, der einen Zeilenumbruch erzeugt. Manipulatoren dienen zur Gestaltung der Ausgabe. Mehr dazu in Abschnitt 2.5, »Zahlen formatieren mit Manipulatoren«. Der Begriff `endl` steht abkürzend für *End of Line* (dt.: Ende der Zeile).

endl

main() Ein einfaches Programm besteht aus einer oder mehreren Anweisungen innerhalb der Funktion `main()`, die der Reihe nach ausgeführt werden. Jede Anweisung wird durch ein Semikolon am Ende begrenzt. Die Zeile mit `cout` beinhaltet eine Anweisung.

{...} Der Aufbau der Funktion `main()` beginnt mit `int main()`. Die Bedeutung von `int` und den runden Klammern werden Sie noch kennenlernen. Nach den runden Klammern folgen die Anweisungen der Funktion `main()` innerhalb der geschweiften Klammern `{}`. Sie erreichen sie über die Tastenkombinationen `[Alt Gr] + [7]` bzw. `[Alt Gr] + [0]`. Es empfiehlt sich, die Anweisungen wie im vorliegenden Fall einzurücken. Sie erhöhen damit die Lesbarkeit Ihrer Programme.

iostream, std Die Bibliothek `iostream` ermöglicht die Ausgabe von Daten auf dem Bildschirm und die Eingabe von Daten über die Tastatur. Sie muss mithilfe von `#include <iostream>` eingebunden werden. Das Objekt `cout` und der Manipulator `endl` stammen aus dem Namensraum (engl.: *namespace*) `std`. Seine Benutzung muss mithilfe der Anweisung `using namespace std;` bekannt gemacht werden.



Hinweis

In Anhang A, »Installationen«, wird die Nutzung der verschiedenen IDEs erläutert. Sie geben das Programm innerhalb der IDE ein, speichern es in der Datei `hallo.cpp`, übersetzen es und führen es aus.

1.5 Kommentieren Sie Ihre Programme

Erläuterung Es kann sein, dass Sie sich selbst nach längerer Zeit wieder mit einem Ihrer Programme beschäftigen oder eines Ihrer Programme an einen anderen Entwickler weitergeben. In diesen Fällen sind Kommentare innerhalb Ihrer Programme sehr nützlich. Sie werden nicht übersetzt und dienen nur zur Erläuterung des Ablaufs.

Im nachfolgenden Programm sehen Sie einige Kommentare:

```
#include <iostream>
using namespace std;

int main()
{
```

```
/* Ein Kommentar
   über mehrere Zeilen */
cout << "Hallo Welt" << endl;

// Ein einzeiliger Kommentar
cout << "Das ist die zweite Zeile" << endl;

cout << "Ende" << endl; // Ende des Programms
}
```

Listing 1.2 Datei »kommentar.cpp«

Sie können einen längeren Kommentar über mehrere Zeilen notieren. Er muss innerhalb der Zeichenkombinationen `/*` und `*/` stehen. Ein kurzer Kommentar nach der Zeichenkombination `//` erstreckt sich nur bis zum Ende der Zeile. Ein Kommentar kann auch nach einer Anweisung in derselben Zeile beginnen.

Das Programm gibt drei Zeilen aus:

```
Hallo Welt
Das ist die zweite Zeile
Ende
```

Kapitel 3

Mehrere Zweige in einem Programm

Ein Programm kann unterschiedliche Anweisungen durchlaufen, abhängig davon, ob eine Bedingung wahr ist oder nicht.

In den bisherigen Programmen werden alle Anweisungen der Reihe nach ausgeführt. Mithilfe von *Kontrollstrukturen* haben wir die Möglichkeit, unsere Programme zu strukturieren und ihren Ablauf zu kontrollieren. Zu den Kontrollstrukturen zählen die *Verzweigungen* und die *Schleifen*.

Kontrollstruktur

3.1 Zwei Zweige mit »if« und »else«

Wir beginnen mit einer einfachen Verzweigung. Im nachfolgenden Programm erfolgen unterschiedliche Ausgaben in Abhängigkeit von der Eingabe des Benutzers:

**Einfache
Verzweigung**

```
#include <iostream>
using namespace std;
int main()
{
    double preis;
    cout << "Preis: ";
    cin >> preis;

    if (preis > 1.5)
    {
        cout << "Ein teurer Artikel" << endl;
    }
    else
    {
        cout << "Ein preiswerter Artikel" << endl;
    }
}
```

```

        cout << "Den nehmen wir" << endl;
    }
}

```

Listing 3.1 Datei »verzweigung.cpp«

Wahrheitswert Das Schlüsselwort `if` leitet eine Verzweigung ein. Innerhalb von runden Klammern folgt eine Bedingung, die mithilfe eines Vergleichsoperators gebildet wird: *Falls der Preis über 1.50 € liegt*. Das Ergebnis der Bedingung ist ein *Wahrheitswert*, also wahr oder falsch.

Anweisungsblock Wird die Bedingung erfüllt, ergibt sich der Wahrheitswert wahr (engl.: *true*), und es wird der Anweisungsblock nach dem `if` ausgeführt. Ein Anweisungsblock besteht aus einer oder mehreren Anweisungen in den geschweiften Klammern `{}`. Wird die Bedingung nicht erfüllt, ergibt sich der Wahrheitswert falsch (engl.: *false*), und es wird der Anweisungsblock nach dem Schlüsselwort `else` ausgeführt.

Ein möglicher Verlauf des Programms:

```

Preis: 1.2
Ein preiswerter Artikel
Den nehmen wir

```

Es kann auch anders aussehen, wie die nachfolgende Ausgabe zeigt:

```

Preis: 1.8
Ein teurer Artikel

```

if und else Eine Verzweigung muss ein `if` und kann ein `else` beinhalten. Ein `else` steht nie allein, sondern gehört immer zu einem vorhergehenden `if`.

Einzelne Anweisung Falls innerhalb eines Anweisungsblocks nur eine Anweisung steht, so können die geschweiften Klammern auch weggelassen werden. Es verbleibt dann nur noch die einzelne Anweisung. Der entsprechende Abschnitt des obigen Programms sähe dann so aus:

```

if (preis > 1.5)
    cout << "Ein teurer Artikel" << endl;
else ...

```

Hinweis

Die Anweisungen in einem Anweisungsblock sind gegenüber den geschweiften Klammern eingerückt. Damit erhöht sich die Lesbarkeit eines Programms. Sie sollten das auch machen, falls es nur eine einzelne Anweisung innerhalb der Verzweigung gibt. Editoren für C++ erkennen Kontrollstrukturen meist selbstständig und rücken automatisch ein.



Lesbarkeit

3.2 Bedingungen benötigen Vergleiche

Innerhalb der Bedingung einer Verzweigung können verschiedene *Vergleichsoperatoren* eingesetzt werden. Im nachfolgenden Programm werden Zahlenwerte auf unterschiedliche Art miteinander verglichen:

Zahlen vergleichen

```

#include <iostream>
using namespace std;
int main()
{
    double preisApfel = 1.45, preisBirne = 0.85, preisKiwi = 1.45;
    int anzahlApfel = 3, anzahlBirne = 5, anzahlKiwi = 3;

    if(preisApfel > preisBirne)
        cout << "Apfel kostet mehr als Birne" << endl;

    if(preisBirne < preisKiwi)
        cout << "Birne kostet weniger als Kiwi" << endl;

    if(anzahlBirne >= anzahlKiwi)
        cout << "Mindestens so viele Birnen wie Kiwis" << endl;

    if(anzahlKiwi <= anzahlBirne)
        cout << "Maximal so viele Kiwis wie Birnen" << endl;

    if(anzahlApfel == anzahlKiwi)
        cout << "Anzahl gleich" << endl;
}

```

```

if(anzahlBirne != anzahlKiwi)
    cout << "Anzahl unterschiedlich" << endl;
}

```

Listing 3.2 Datei »operator_vergleich.cpp«

Tabelle der Vergleichsoperatoren

Bei den Verzweigungen wird jeweils nur eine Anweisung ausgeführt, daher werden hier keine geschweiften Klammern benötigt. Die Vergleichsoperatoren sehen Sie in Tabelle 3.1.

Operator	Bedeutung
>	größer als
<	kleiner als
>=	größer als oder gleich
<=	kleiner als oder gleich
==	gleich
!=	ungleich

Tabelle 3.1 Vergleichsoperatoren

Operator == Beachten Sie die doppelten Gleichheitszeichen des Operators ==. Falls Sie bei einem Vergleich (bewusst oder versehentlich) ein einfaches Gleichheitszeichen setzen, wird stattdessen eine Zuweisung durchgeführt. Das führt zu einem anderen Ergebnis.

Die Ausgabe des Programms:

```

Apfel kostet mehr als Birne
Birne kostet weniger als Kiwi
Mindestens so viele Birnen wie Kiwis
Maximal so viele Kiwis wie Birnen
Anzahl gleich
Anzahl unterschiedlich

```



Fließkommazahlen
vergleichen

Hinweis

Fließkommazahlen werden nicht mathematisch genau gespeichert. Der Zahlenwert 1.5 könnte z. B. als 1.500001, als 1.500000 oder auch als 1.499999 gespeichert werden. Daher sollten nur ganze Zahlen auf Gleichheit oder Ungleichheit verglichen werden. Eine geeignete Möglichkeit für

einen Vergleich von Fließkommazahlen finden Sie in Abschnitt 9.4.5, »Betrag und Vergleich«.

3.3 Mehr als zwei Zweige

Falls es mehr als zwei mögliche Fälle gibt, können Sie eine *geschachtelte Verzweigung* oder auch eine *mehrfache Verzweigung* einsetzen.

Nehmen wir an, für die Beurteilung des Preises eines Artikels wird zwischen vier Fällen unterschieden, siehe Tabelle 3.2. Vier Fälle

Untergrenze	Obergrenze	Beurteilung
(keine)	< 0.50 €	sehr preiswert
>= 0.50 €	< 1.00 €	preiswert
>= 1.00 €	< 2.00 €	teuer
>= 2.00 €	(keine)	sehr teuer

Tabelle 3.2 Vier verschiedene Fälle

Der erste Teil des Programms mit der geschachtelten Verzweigung:

Geschachtelt

```

#include <iostream>
using namespace std;
int main()
{
    double preis;
    cout << "Preis: ";
    cin >> preis;

    if(preis >= 1.0)
    {
        if(preis >= 2.0)
            cout << "sehr teuer" << endl;
        else
            cout << "teuer" << endl;
    }
    else
    {

```

```

    if(preis < 0.5)
        cout << "sehr preiswert" << endl;
    else
        cout << "preiswert" << endl;
}
...

```

Listing 3.3 Datei »verzweigung_mehrere.cpp«, Teil 1 von 2

Reihenfolge der Prüfung Zu Beginn der geschachtelten Verzweigung wird geprüft, ob der Preis mindestens 1.00 € beträgt. Falls ja, kann die Beurteilung nur noch *teuer* oder *sehr teuer* lauten. Innerhalb des Anweisungsblocks nach dem `if` wird geprüft, welcher dieser beiden Fälle zutrifft.

Falls der Preis unter 1.00 € liegt, kommen nur noch die Beurteilungen *preiswert* oder *sehr preiswert* in Betracht. Das wird genauer innerhalb des Anweisungsblocks nach dem `else` untersucht.

Mehrfach Der zweite Teil des Programms mit der mehrfachen Verzweigung:

```

...
    if(preis < 0.5)
        cout << "sehr preiswert" << endl;
    else if(preis < 1.0)
        cout << "preiswert" << endl;
    else if(preis < 2.0)
        cout << "teuer" << endl;
    else
        cout << "sehr teuer" << endl;
}

```

Listing 3.4 Datei »verzweigung_mehrere.cpp«, Teil 2 von 2

Reihenfolge der Prüfung Zu Beginn der mehrfachen Verzweigung wird mithilfe des ersten `if` geprüft, ob der Preis unter 0.50 € liegt. Trifft das nicht zu, wird mithilfe des zweiten `if` geprüft, ob der Preis unter 1.00 € liegt. Es muss nicht mehr geprüft werden, ob er mindestens 0.50 € beträgt. Trifft auch das nicht zu, wird mithilfe des dritten `if` geprüft, ob der Preis unter 2.00 € liegt. Auch hier muss nicht mehr geprüft werden, ob er mindestens 1.00 € beträgt. Trifft das ebenfalls nicht zu, folgt das `else` zum dritten `if`.

Ein möglicher Verlauf des Programms, abhängig von der Eingabe:

```

Preis: 0.9
preiswert
preiswert

```

Sie sehen, es gibt mehrere Arten, das Problem zu lösen. In der Praxis entscheiden Sie, welche Form für Sie lesbarer und übersichtlicher ist.

Hinweis

Sie können die Leistung eines Programms, in dem Werte häufig mithilfe von mehreren Bedingungen untersucht werden, steigern. Dazu sollten Sie die Bedingung als Erste prüfen, die erfahrungsgemäß am häufigsten zutrifft. Trifft sie zu, müssen die anderen Bedingungen anschließend nicht mehr durchlaufen werden.



Leistung steigern

3.4 Wie kann ich Bedingungen kombinieren?

Sie können mehrere Bedingungen mithilfe der folgenden logischen Operatoren miteinander verknüpfen und gemeinsam auswerten: `&&` für die *logische Und-Verknüpfung*, `||` für die *logische Oder-Verknüpfung* und `!` für die *logische Verneinung*.

Im nachfolgenden Programm werden sie genutzt:

```

#include <iostream>
using namespace std;
int main()
{
    double preisApfel = 1.45, preisBirne = 0.85, preisBanane = 0.75;

    if(preisBirne < 1.0 && preisBanane < 1.0)
        cout << "Beide Artikel sind preiswert" << endl;

    if(preisApfel >= 1.0 || preisBirne >= 1.0)
        cout << "Mindestens einer der Artikel ist teuer" << endl;
}

```

Logische Operatoren

```

    if(!(preisBanane >= 1.0))
        cout << "Artikel ist nicht teuer" << endl;
}

```

Listing 3.5 Datei »operator_logisch.cpp«

- Operator &&** Bei der Verknüpfung mithilfe des logischen Operators && liefert die Gesamtbedingung nur dann das Ergebnis *wahr*, falls beide Teilbedingungen den Wahrheitswert *wahr* liefern.
- Operator ||** Bei der Verknüpfung mithilfe des logischen Operators || liefert die Gesamtbedingung bereits dann das Ergebnis *wahr*, falls mindestens eine der beiden Teilbedingungen den Wahrheitswert *wahr* liefert.
- Vorrang** Die Vergleichsoperatoren haben Vorrang vor den logischen Operatoren && und ||. Daher werden die Teilbedingungen als Erste ausgewertet und müssen nicht in runde Klammern gesetzt werden.
- Operator !** Der logische Operator ! dreht den Wahrheitswert einer Bedingung um: Aus *wahr* wird *falsch*, aus *falsch* wird *wahr*. Der logische Operator ! hat allerdings Vorrang vor dem Vergleichsoperator. Daher müssen Sie die Bedingung in runde Klammern setzen.

Die Ausgabe des Programms:

```

Beide Artikel sind preiswert
Mindestens einer der Artikel ist teuer
Artikel ist nicht teuer

```

**Hinweise:**

Achten Sie darauf, nur sinnvolle Verknüpfungen zu formulieren. Die Bedingung `(preis >= 20 && preis <= 30)` liefert nur für die Werte zwischen 20 und 30 (jeweils einschließlich) den Wahrheitswert *wahr*. Die Bedingung `(preis >= 20 || preis <= 30)` liefert für jeden beliebigen Wert den Wahrheitswert *wahr*. Die Bedingung `(preis <= 20 && preis >= 30)` liefert für keinen Wert den Wahrheitswert *wahr*.

Verkürzung Falls bei einer Verknüpfung mithilfe des logischen Operators && die erste Teilbedingung bereits den Wahrheitswert *falsch* liefert, wird die zweite Teilbedingung gar nicht mehr überprüft, weil die Gesamtbedingung nicht mehr *wahr* werden kann.

Entsprechend gilt: Falls bei einer Verknüpfung mithilfe des logischen Operators || die erste Teilbedingung bereits den Wahrheitswert *wahr* liefert,

wird die zweite Teilbedingung gar nicht mehr überprüft, weil die Gesamtbedingung bereits *wahr* ist.

3.5 Zweige zusammenfassen mit »switch« und »case«

Mithilfe einer switch-case-Verzweigung haben Sie die Möglichkeit, einzelne Zweige zusammenzufassen und eine mehrfache Verzweigung übersichtlich zu gestalten.

**Mehrfache
Verzweigung**

Im nachfolgenden Programm gibt der Benutzer einen Monat als Zahl von 1 bis 12 ein. Anschließend wird die Bezeichnung des zugehörigen Quartals ausgegeben:

```

#include <iostream>
using namespace std;
int main()
{
    int monat;
    cout << "Monat: ";
    cin >> monat;

    switch(monat)
    {
        case 1:
        case 2:
        case 3:
            cout << "1. Quartal" << endl;
            break;
        case 4:
        case 5:
        case 6:
            cout << "2. Quartal" << endl;
            break;
        case 7:
        case 8:
        case 9:
            cout << "3. Quartal" << endl;
            break;
    }
}

```

```

    case 10:
    case 11:
    case 12:
        cout << "4. Quartal" << endl;
        break;
    default:
        cout << "Das ist kein Monat" << endl;
}
}

```

Listing 3.6 Datei »verzweigung_switch.cpp«

switch, case Nach dem Schlüsselwort `switch` muss eine ganzzahlige Variable in runden Klammern notiert werden. Ihr Wert wird in dem folgenden Block in geschweiften Klammern untersucht. Jeder mögliche Fall wird mithilfe des Schlüsselworts `case` formuliert.

break Bei der Ausführung des Programms werden die Fälle der Reihe nach durchlaufen. Sobald ein Fall zutrifft, werden alle danach folgenden Anweisungen bis zum nächsten Vorkommen des Schlüsselworts `break` ausgeführt. Damit wird der Block abgebrochen, und der Programmablauf wird danach fortgesetzt.

Ein Beispiel: Der Benutzer gibt die Zahl 7 für den Monat Juli ein, also trifft `case 7` zu. Die folgende Anweisung ist die Ausgabe `3. Quartal`. Anschließend folgt ein `break`, und die Verzweigung ist beendet.

default Nach dem Schlüsselwort `default` können weitere Anweisungen folgen. Sie werden durchgeführt, falls keiner der zuvor notierten Fälle zutrifft.

Eine mögliche Ausgabe des Programms:

```

Monat: 7
3. Quartal

```

3.6 Was ist mit dem Rest?

Operator % Einen Rechenoperator habe ich bisher nur kurz erwähnt. Der Operator `%`, auch *Modulo-Operator* genannt, berechnet den Rest bei der Division von zwei ganzen Zahlen. In Abschnitt 2.2, »Rechnen mit Operatoren«, haben wir bereits gesehen, dass bei einer solchen Division die Nachkommastellen abgeschnitten werden, das Ergebnis also nicht mathematisch genau ist.

Im folgenden Programm gibt der Benutzer eine ganze Zahl ein. Mithilfe des Modulo-Operators wird der Rest einer Division durch 3 berechnet. Außerdem wird untersucht, ob die Zahl ohne Rest durch 3 teilbar ist.

Modulo

```

#include <iostream>
using namespace std;
int main()
{
    int anzahl;
    cout << "Anzahl: ";
    cin >> anzahl;

    int rest = anzahl % 3;
    cout << "Rest bei Teilung durch 3: " << rest << endl;

    if(anzahl % 3 == 0)
        cout << anzahl << " ist durch 3 teilbar" << endl;
    else
        cout << anzahl << " ist nicht durch 3 teilbar" << endl;
}

```

Listing 3.7 Datei »operator_modulo.cpp«

Eine mögliche Ausgabe könnte wie folgt aussehen:

```

Anzahl: 13
Rest bei Teilung durch 3: 1
13 ist nicht durch 3 teilbar

```

Hinweis

Rechenoperatoren haben Vorrang vor Vergleichsoperatoren. Daher müssen innerhalb der Bedingung nach dem `if` keine zusätzlichen runden Klammern notiert werden. An dieser Stelle weise ich noch einmal auf die *doppelten* Gleichheitszeichen beim Vergleichsoperator `==` hin.



3.7 Welcher Operator hat Vorrang?

Innerhalb von Kontrollstrukturen und Rechenausdrücken werden gleichzeitig viele Operatoren unterschiedlichen Typs eingesetzt. Dabei ist die Rei-

henfolge der Auswertung wichtig. Bestimmte Operatoren haben Vorrang vor bestimmten anderen Operatoren.

Rangfolge In Tabelle 3.3 sehen Sie eine Rangfolge der bisher genutzten Operatoren. In der obersten Tabellenzelle steht der Operator mit dem höchsten Rang. Operatoren innerhalb derselben Tabellenzelle haben denselben Rang.

Operator	Erläuterung
x++ x--	nachstehendes Inkrement und Dekrement
! ++x --x	logisches Nicht, voranstehendes Inkrement und Dekrement
* / %	Multiplikation, Division, Modulo
< <= > >=	kleiner als (oder gleich), größer als (oder gleich)
== !=	gleich, ungleich
&&	logisches Und
	logisches Oder
?: += -= *= /=	bedingter Ausdruck, (kombinierte) Zuweisung

Tabelle 3.3 Rangfolge der bisher genutzten Operatoren



Hinweis

Beinhaltet eine Verzweigung sowohl den logischen Operator `&&` als auch den logischen Operator `||`, hat der Teilausdruck mit `&&` gemäß Tabelle 3.3 Vorrang vor dem Teilausdruck mit `||`. Zur deutlicheren Darstellung empfehle ich dennoch, den Teilausdruck mit `&&` in Klammern zu setzen. So vermeiden Sie auch eine entsprechende Warnung bei der Nutzung des *Clang-Compilers* unter *macOS*.

3.8 Übungen

Einfache
Verzweigung

Übung 3A: Entwickeln Sie ein Programm in der Datei `u_verzweigung.cpp`. Der Benutzer wird dazu aufgefordert, zwei Zahlen einzugeben. Die Zahlen

können Nachkommastellen haben. Sie sollen in absteigender Reihenfolge wieder ausgegeben werden.

Zu jeder Übung finden Sie eine mögliche Lösung im Downloadbereich zum Buch unter der Adresse www.rheinwerk-verlag.de/5179.

Ein Aufruf des fertigen Programms könnte wie folgt aussehen:

Erste Zahl eingeben: 2.5

Zweite Zahl eingeben: 6.1

Ihre Zahlen:

6.1

2.5

Übung 3B: Entwickeln Sie ein Programm zur Prüfung einer Datumsangabe in der Datei `u_datum.cpp`. Der Benutzer soll die drei Bestandteile eines Datums (Tag, Monat, Jahr) einzeln eingeben. Anschließend wird ermittelt, ob es sich um ein richtiges oder ein falsches Datum handelt.

Mehrere
Bedingungen

Gehen Sie bei der Entwicklung in den nachfolgend beschriebenen Schritten vor. Testen Sie Ihr Programm nach jedem Schritt:

- ▶ Falls der eingegebene Wert für den Tag nicht zwischen 1 und 31 liegt, handelt es sich um ein falsches Datum.
- ▶ Falls der eingegebene Wert für den Monat nicht zwischen 1 und 12 liegt, handelt es sich um ein falsches Datum.
- ▶ Untersuchen Sie den eingegebenen Wert für den Monat. Geben Sie den Wert aus, den der letzte Tag des betreffenden Monats hat. Schaltjahre sollen zunächst nicht berücksichtigt werden. Es gibt also nur drei mögliche Fälle.
- ▶ Falls der eingegebene Wert für den Tag größer als der letzte Tag des betreffenden Monats ist, handelt es sich um ein falsches Datum.
- ▶ Untersuchen Sie den eingegebenen Wert für das Jahr. Geben Sie aus, ob es sich um ein Schaltjahr handelt. Die vereinfachte Regel für ein Schaltjahr lautet: Falls sich der Wert ohne Rest durch 4 teilen lässt, handelt es sich um ein Schaltjahr.
- ▶ Kombinieren Sie die bisherigen Schritte miteinander. Falls der Wert für den Tag größer als der letzte Tag des betreffenden Monats ist (mit Berücksichtigung der Regel für ein Schaltjahr), handelt es sich um ein falsches Datum.

- Erweitern Sie das Programm. Die vollständige Regel für ein Schaltjahr lautet: Falls sich der Wert ohne Rest durch 4 teilen lässt, aber nicht ohne Rest durch 100 teilen lässt, handelt es sich um ein Schaltjahr. Es handelt sich aber auch um ein Schaltjahr, falls sich der Wert ohne Rest durch 400 teilen lässt.

Ein Aufruf des fertigen Programms könnte wie folgt aussehen:

Tag des Datums eingeben: 29
 Monat des Datums eingeben: 2
 Jahr des Datums eingeben: 2000
 Letzter Tag: 29
 Richtiges Datum

Bewertung **Übung 3C:** Ändern Sie das Programm für das Kopfrechentraining aus der Datei `u_kopf_eingabe.cpp`. Das Programm in der Datei `u_kopf_verzweigung.cpp` soll die Eingabe des Benutzers mit »Falsch« oder »Richtig« bewerten. In jedem Fall soll das richtige Ergebnis angezeigt werden.

Ein Aufruf des fertigen Programms könnte wie folgt aussehen:

Aufgabe: $24 + 33 = 58$
 Falsch: $24 + 33 = 57$

Mehrere Operatoren **Übung 3D:** Ändern Sie das Programm für das Kopfrechentraining aus der Datei `u_kopf_verzweigung.cpp`. Mithilfe des Programms in der Datei `u_kopf_verzweigung_mehrfach.cpp` soll der Benutzer nicht nur die Addition, sondern auch die drei anderen Grundrechenarten trainieren.

Unterschiedliche Zahlenbereiche Mithilfe eines Zufallsgenerators soll eine Zahl für den Operator ermittelt werden. Wird die Zahl 0 ermittelt, wird eine Addition durchgeführt, bei einer 1 eine Subtraktion, bei einer 2 eine Multiplikation und bei einer 3 eine Division. Für die ersten beiden Rechenarten sollen wie bisher zufällige Zahlen zwischen 20 und 40 ermittelt werden, für die beiden anderen Rechenarten zufällige Zahlen zwischen 10 und 15.

Trick für Division Bei der Division soll es nur ganzzahlige Ergebnisse geben. Dazu ein kleiner Trick: Das Ergebnis einer Division ergibt sich aus dem Dividenten, geteilt durch den Divisor. Ermitteln Sie zufällige Zahlen für das Ergebnis und den Divisor, und berechnen Sie den Dividenten durch die Multiplikation dieser beiden Werte. Ein Beispiel: Aus dem zufälligen Ergebnis 8 und dem zufälligen Divisor ergibt sich der Divident 40. Dem Benutzer wird dann folgende Aufgabe gestellt: $40 / 5 = \dots$

Ein Aufruf des fertigen Programms könnte wie folgt aussehen:

Aufgabe: $8 * 12 = 95$
 Falsch: $8 * 12 = 96$

3.9 Wie speichere ich Wahrheitswerte?

Sie können an dieser Stelle bereits mit dem nächsten Kapitel fortfahren und die weiteren Abschnitte dieses Kapitels bei Bedarf später bearbeiten. Sie dienen der Ergänzung Ihres Wissens.

Der Datentyp `bool` dient zur Speicherung von *Wahrheitswerten*. Variablen des Datentyps `bool`, auch *boolesche Variablen* genannt, können entweder den Wert `true` oder den Wert `false` haben.

Boolesche Variable

Wahrheitswerte können aus mehreren Quellen stammen:

- Sie können das Ergebnis einer Bedingung sein.
- Es kann sich um einen der beiden Werte `true` oder `false` handeln.
- Sie könnten einer booleschen Variablen sogar Zahlenwerte zuweisen. Dabei findet eine implizite (d. h. automatisierte) Konvertierung in einen booleschen Wert statt. Der Zahlenwert 0 entspricht dem booleschen Wert `false`. Alle anderen Zahlenwerte entsprechen dem Wert `true`.

0 = false

Im ersten Teil des nachfolgenden Programms werden einer booleschen Variablen Wahrheitswerte und Zahlenwerte zugewiesen:

```
#include <iostream>
using namespace std;
int main()
{
    bool x;
    x = true;   cout << "true: " << x << endl;
    x = false; cout << "false: " << x << endl;
    x = 1;     cout << "1:    " << x << endl;
    x = 0;     cout << "0:    " << x << endl;
    x = -1.5;  cout << "-1.5: " << x << endl;
    ...
}
```

Listing 3.8 Datei »datentyp_bool.cpp«, Teil 1 von 2

Die Ausgabe des ersten Programnteils:

```
true: 1
false: 0
1: 1
0: 0
-1.5: 1
```

Ausgabe 0 oder 1 Der Wert einer booleschen Variablen wird als 0 oder 1 ausgegeben. Der *Clang-Compiler* unter *macOS* akzeptiert zwar die implizite Konvertierung eines *double*-Werts in einen booleschen Wert, gibt aber eine Warnung aus.

Im zweiten Teil des Programms werden die Ergebnisse von Bedingungen verschiedenen booleschen Variablen zugewiesen. Anschließend werden diese Variablen in Verzweigungen genutzt:

```
...
double preisApfel = 1.45, preisBirne = 0.85, preisBanane = 0.75;
bool apfelTeuer = preisApfel >= 1.0;
bool birneTeuer = preisBirne >= 1.0;
bool bananeTeuer = preisBanane >= 1.0;

if(apfelTeuer || birneTeuer)
    cout << "Mindestens einer der Artikel ist teuer" << endl;
if(!bananeTeuer)
    cout << "Artikel ist nicht teuer" << endl;
if(!birneTeuer && !bananeTeuer)
    cout << "Beide Artikel sind nicht teuer" << endl;
}
```

Listing 3.9 Datei »datentyp_bool.cpp«, Teil 2 von 2

Die Reihenfolge bei den Zuweisungen: Zunächst wird die Bedingung ausgewertet und liefert *true* oder *false*. Anschließend wird dieser Wert einer booleschen Variablen zugewiesen.

Operator ! Ähnlich wie Sie es bei den logischen Verknüpfungen in Abschnitt 3.4, »Wie kann ich Bedingungen kombinieren?«, sehen: Der logische Operator **!** dreht den Wahrheitswert einer booleschen Variablen um.

Die Ausgabe des zweiten Programnteils:

```
Mindestens einer der Artikel ist teuer
Artikel ist nicht teuer
Beide Artikel sind nicht teuer
```

Hinweis

Zur effektiven Speicherung größerer Mengen von Wahrheitswerten bietet C++ den sogenannten *Bitset* an, siehe Abschnitt 13.12, »Eine Menge von Bits«.



3.10 Die Kurzform: der bedingte Ausdruck

Ein *bedingter Ausdruck* bietet die Möglichkeit, Verzweigungen zu verkürzen. Das folgende Programm zeigt einige Einsatzsituationen:

```
#include <iostream>
using namespace std;
int main()
{
    double preis = 1.45;
    int preisKategorie;

    preisKategorie = preis >= 1.0 ? 2 : 1;
    cout << "Kategorie: " << preisKategorie << " von 2" << endl;

    preisKategorie = preis >= 1.0 ? (preis >= 2.0 ? 3 : 2) : 1;
    cout << "Kategorie: " << preisKategorie << " von 3" << endl;

    cout << (preis >= 1.0 ? "Ein teurer Artikel"
            : "Ein preiswerter Artikel") << endl;
}
```

Listing 3.10 Datei »ausdruck_bedingt.cpp«

Der Preis eines Artikels wird in eine bestimmte Kategorie eingeordnet.

Im ersten Beispiel wird der Variablen *preisKategorie* der Wert 2 zugewiesen, falls der Preis mindestens 1.00 € beträgt, ansonsten der Wert 1. In

Operator ? :

einem bedingten Ausdruck wird mit dem Operator `?` : gearbeitet. Vor dem Fragezeichen steht eine Bedingung. Falls die Bedingung den Wahrheitswert `wahr` ergibt, liefert der bedingte Ausdruck den Zahlenwert nach dem Fragezeichen, ansonsten liefert er den Zahlenwert nach dem Doppelpunkt.

Geschachtelt Im zweiten Beispiel sehen Sie einen geschachtelten bedingten Ausdruck. Im äußeren Ausdruck wird zunächst geprüft, ob der Preis mindestens 1.00 € beträgt. Trifft das zu, wird im inneren Ausdruck geprüft, ob er mindestens 2.00 € beträgt. Trifft das auch zu, wird der Zahlenwert 3 geliefert und zugewiesen, ansonsten der Zahlenwert 2.

Liegt der Preis unter 1.00 €, wird der Zahlenwert 1 ermittelt und zugewiesen. Der innere Ausdruck steht zur besseren Übersicht in runden Klammern.

Ausdruck ausgeben Der bedingte Ausdruck im dritten Beispiel liefert als Wert einen Text, der mithilfe von `cout` ausgegeben wird. Hier sind die runden Klammern für die richtige Reihenfolge der Bearbeitung notwendig.

Die Ausgabe des Programms:

Kategorie: 2 von 2

Kategorie: 2 von 3

Ein teurer Artikel

Auf einen Blick

1	Eine erste Einführung	19
2	Arbeiten mit Zahlen und Operatoren	25
3	Mehrere Zweige in einem Programm	49
4	Teile von Programmen wiederholen	67
5	Programme aufteilen in Funktionen	79
6	Große Datenmengen speichern in Feldern	109
7	Arbeiten mit Zeichen und Texten	133
8	Daten in Strukturen zusammenfassen	157
9	Vorhandene Funktionen nutzen	175
10	Eigene Klassen entwerfen	237
11	Vererbung und Polymorphie	269
12	Datenströme verarbeiten	289
13	Container sind vielfältige Datenstrukturen	319
14	Mehr zu eigenen Klassen	383
15	Präprozessor-Anweisungen	409
16	Grafische Benutzeroberflächen mit der Qt-Bibliothek	417
17	Datenbanken mit SQLite verwalten	449

Inhalt

Materialien zum Buch	17
1 Eine erste Einführung	19
1.1 Was machen wir mit C++?	19
1.2 Was benötige ich zum Programmieren?	20
1.3 Die Entwicklung von C++	20
1.4 So sieht das erste Programm aus	21
1.5 Kommentieren Sie Ihre Programme	22
2 Arbeiten mit Zahlen und Operatoren	25
2.1 Wie speichere ich Zahlen?	25
2.2 Rechnen mit Operatoren	27
2.3 Fehler suchen	29
2.4 Wie können Daten eingegeben werden?	31
2.5 Zahlen formatieren mit Manipulatoren	33
2.6 Zuweisungen kürzer schreiben	34
2.7 Übung	36
2.8 Mehr über die Speicherung von Zahlen	37
2.8.1 Ganzzahlige Datentypen	37
2.8.2 Datentypen für Fließkommazahlen	40
2.9 Feste Werte in Konstanten speichern	41
2.10 Konstanten in Enumerationen zusammenfassen	42
2.11 Zahlensysteme	43

2.12	Initialisierung	45
2.13	Wie erzeuge ich zufällige Zahlen?	47
2.14	Übung	48
3	Mehrere Zweige in einem Programm	49
3.1	Zwei Zweige mit »if« und »else«	49
3.2	Bedingungen benötigen Vergleiche	51
3.3	Mehr als zwei Zweige	53
3.4	Wie kann ich Bedingungen kombinieren?	55
3.5	Zweige zusammenfassen mit »switch« und »case«	57
3.6	Was ist mit dem Rest?	58
3.7	Welcher Operator hat Vorrang?	59
3.8	Übungen	60
3.9	Wie speichere ich Wahrheitswerte?	63
3.10	Die Kurzform: der bedingte Ausdruck	65
4	Teile von Programmen wiederholen	67
4.1	Regelmäßige Wiederholungen mit »for«	67
4.2	Wiederholungen für einen Bereich	70
4.3	Bedingte Wiederholungen mit »do-while«	71
4.4	Besser vorher prüfen mit »while«	72
4.5	Wiederholungen abbrechen oder fortsetzen	73
4.6	Die Wiederholung der Wiederholung	75
4.7	Übungen	76

5	Programme aufteilen in Funktionen	79
5.1	So schreibe ich eine eigene Funktion	79
5.2	Wie übergebe ich Daten?	81
5.2.1	Ein weiterer Zugriff über eine Referenz	83
5.2.2	Übergabe an eine Referenz	84
5.2.3	Übergabe an eine konstante Referenz	86
5.3	Wie erhalte ich ein Ergebnis zurück?	87
5.4	Mehr Ordnung im Programm	88
5.5	Statische Variablen behalten ihren Wert	90
5.6	Fehler suchen	91
5.7	Übungen	92
5.8	Standardwerte vorgeben	93
5.9	Beliebig viele Parameter	94
5.10	Funktionen mehrfach definieren	96
5.11	Funktionen, die sich selbst aufrufen	98
5.12	Anonyme Funktionen	99
5.12.1	So schreibe ich eine anonyme Funktion	100
5.12.2	Daten von außerhalb erfassen	101
5.12.3	Wie übergebe ich Parameter?	102
5.12.4	Wie erhalte ich ein Ergebnis zurück?	104
5.13	Funktionen als Parameter	105
6	Große Datenmengen speichern in Feldern	109
6.1	Wie werden Felder unterschieden?	109
6.2	Einfache Felder mit fester Größe	110
6.3	Intelligente Felder mit fester Größe	111
6.4	Ausnahmen behandeln	113

6.5	Einfache und intelligente Zeiger	115
6.5.1	Einfache Zeiger	116
6.5.2	Die Operatoren »new« und »delete«	118
6.5.3	Ein intelligenter und eindeutiger Zeiger	119
6.5.4	Das Funktionstemplate »make_unique«	120
6.5.5	Weitere intelligente Zeiger	121
6.6	Intelligente Felder mit variabler Größe	122
6.7	Felder initialisieren	124
6.8	Felder als Parameter	125
6.9	Daten in mehreren Dimensionen speichern	127
6.9.1	Ein zweidimensionales Feld mit fester Größe	127
6.9.2	Ein zweidimensionales Feld mit variabler Größe	129
6.10	Übungen	131
7	Arbeiten mit Zeichen und Texten	133
7.1	Einzelne Zeichen	133
7.2	Einfache Zeichenketten	134
7.3	Intelligente Zeichenketten: Strings	137
7.3.1	Strings erzeugen	137
7.3.2	Strings ändern	139
7.3.3	Strings vergleichen und durchsuchen	141
7.4	Wie wandle ich Zahlen in Strings um?	143
7.5	Wie verarbeite ich Eingaben?	144
7.6	Felder von Zeichenketten	147
7.7	Suchen und Ersetzen	149
7.7.1	Funktionen zum Suchen und Ersetzen	149
7.7.2	Beginn und Ende	151
7.7.3	Auswahl und Bereiche	152
7.7.4	Gruppen, Anzahl und Sonderzeichen	153
7.8	Übungen	154

8	Daten in Strukturen zusammenfassen	157
8.1	Wie speichere ich zusammengehörige Daten?	157
8.2	Besser einen Typ definieren	159
8.3	Strukturen und Felder	161
8.4	Strukturen und Funktionen	165
8.4.1	Das Hauptprogramm	166
8.4.2	Die Größe des Felds festlegen	167
8.4.3	Die Eingabefunktionen	168
8.4.4	Die Ausgabefunktionen	169
8.5	Eine Hierarchie von Strukturen	170
8.6	Übung	172
9	Vorhandene Funktionen nutzen	175
9.1	Umgang mit Datum und Uhrzeit	175
9.1.1	Datum und Uhrzeit ausgeben	176
9.1.2	Zeit messen	178
9.1.3	Zeitangaben erzeugen und berechnen	179
9.2	Bessere zufällige Zahlen	181
9.2.1	Mersenne-Twister	181
9.2.2	Binomialverteilung	182
9.2.3	Übung	184
9.3	Mehrere Threads zur gleichen Zeit	184
9.3.1	Ablauf von Threads	185
9.3.2	Parameter für Threads	187
9.3.3	Gemeinsame Daten	189
9.3.4	Gemeinsame Daten schützen	191
9.4	Nützliche mathematische Funktionen	193
9.4.1	Winkelfunktionen	193
9.4.2	Funktionen zum Runden	195
9.4.3	Verschiedene mathematische Funktionen	197
9.4.4	Prüffunktionen	198

9.4.5	Betrag und Vergleich	199
9.4.6	Numerische Funktionen	201
9.5	Übungen	202
9.6	Rechnen mit komplexen Zahlen	203
9.6.1	Erstellung von komplexen Zahlen	203
9.6.2	Spezielle Funktionen	205
9.6.3	Operatoren	207
9.6.4	Mathematische Funktionen	209
9.7	Daten mit dem Betriebssystem austauschen	210
9.7.1	Wie lauten die Parameter?	210
9.7.2	Berechnung der Summe der Parameter	212
9.7.3	Wie nutze ich die Rückgabe eines Programms?	213
9.7.4	Systemkommandos ausführen	215
9.7.5	Ausgabe umlenken	216
9.7.6	Eingabe umlenken	217
9.8	Zugriff auf Dateien und Verzeichnisse	219
9.8.1	Pfad-Objekte	220
9.8.2	Rechte für den Zugriff ermitteln	224
9.8.3	Rechte für den Zugriff ändern	226
9.8.4	Verzeichnisse und Verzeichnishierarchien	227
9.8.5	Dateien ändern	229
9.8.6	Verzeichnisse ändern	231
9.8.7	Verzeichnisse rekursiv ändern	233
10	Eigene Klassen entwerfen	237
10.1	Klassen umfassen Eigenschaften und Methoden	237
10.1.1	Die Definition der Klasse	238
10.1.2	Objekte als Instanzen einer Klasse	239
10.2	Schützen Sie die Daten	240
10.3	Wie erzeuge und lösche ich Objekte?	244
10.4	Statische Elemente einer Klasse	248
10.5	Wie überlade ich Operatoren?	252

10.6	Objekte und Felder	256
10.7	Objekte ausgeben	260
10.8	Eigenschaften können Objekte sein	262
10.9	Übungen	264
11	Vererbung und Polymorphie	269
11.1	Basisklasse und abgeleitete Klassen	269
11.2	Welche Elemente sind an welcher Stelle erreichbar?	276
11.3	Konstruktoren in abgeleiteten Klassen	277
11.4	Was bedeutet Polymorphie?	281
11.5	Erben von mehreren Klassen	284
12	Datenströme verarbeiten	289
12.1	Sequenzielles Schreiben und Lesen	289
12.1.1	Schreiben in eine Datei	290
12.1.2	Pfadangaben	292
12.1.3	Lesen aus einer Datei	292
12.1.4	Mehrmals öffnen und schließen	295
12.2	Schreiben und Lesen an beliebiger Stelle	296
12.2.1	Formatiertes Schreiben in eine Datei	297
12.2.2	Lesen an beliebiger Stelle einer Datei	299
12.2.3	Schreiben an beliebiger Stelle in eine Datei	302
12.3	Wie leiten Sie Datenströme?	305
12.3.1	String-Streams auf dem Bildschirm ausgeben	305
12.3.2	String-Streams von Tastatur lesen	306
12.4	Wie ist das CSV-Format zum Austausch aufgebaut?	308
12.4.1	Im CSV-Format in eine Datei schreiben	308
12.4.2	Eine Datei im CSV-Format lesen	312
12.5	Übung	315

13	Container sind vielfältige Datenstrukturen	319
13.1	Wie durchlaufe ich Container?	320
13.2	Intelligente Felder mit fester Größe	322
13.3	Intelligente Felder mit variabler Größe	326
13.4	Eine Warteschlange mit zwei Enden	330
13.5	Daten in Listen verketteten	334
13.5.1	Einfach verkettete Listen	334
13.5.2	Doppelt verkettete Listen	339
13.6	Drei einfache Container	344
13.6.1	Stapel	345
13.6.2	Einfache Warteschlangen	347
13.6.3	Prioritäts-Warteschlangen	349
13.7	Zwei nützliche Typen	351
13.7.1	Paare	351
13.7.2	Tupel	353
13.8	Eine Menge von Elementen	355
13.8.1	Sortiert und einzigartig	355
13.8.2	Nicht sortiert und einzigartig	360
13.8.3	Sortiert und nicht einzigartig	360
13.8.4	Nicht sortiert und nicht einzigartig	361
13.9	Schlüssel und Werte in einer Map	362
13.9.1	Sortiert und einzigartig	362
13.9.2	Nicht sortiert und einzigartig	367
13.9.3	Sortiert und nicht einzigartig	368
13.9.4	Nicht sortiert und nicht einzigartig	369
13.10	Algorithmen für Bereiche	369
13.11	Mengenlehre	374
13.12	Eine Menge von Bits	378
13.12.1	Funktionen für Bitsets	378
13.12.2	Operatoren für Bitsets	380

14	Mehr zu eigenen Klassen	383
14.1	Objekte initialisieren, kopieren und erzeugen	383
14.2	Klassen können Freunde haben	386
14.2.1	Befreundete Funktionen	386
14.2.2	Befreundete Klassen	387
14.3	Namen müssen eindeutig sein	389
14.4	Fehler behandeln mit Ausnahmen	392
14.4.1	Ausnahmen in Schleifen	392
14.4.2	Eigene Ausnahmen werfen	393
14.4.3	Ausnahme in Funktion behandeln	395
14.4.4	Eigene Ausnahmeklasse definieren	397
14.5	Innere Klassen	400
14.6	Templates sind Vorlagen	402
14.6.1	Templates für Funktionen	402
14.6.2	Templates für Klassen	405
15	Präprozessor-Anweisungen	409
15.1	Einbinden von Dateien	409
15.2	Definitionen und Makros	410
15.3	Definitionen und Verzweigungen	412
15.4	Eine Systemweiche	414
16	Grafische Benutzeroberflächen mit der Qt-Bibliothek	417
16.1	Die erste GUI-Anwendung	418
16.1.1	Klasse und Dateien	418
16.1.2	Projekt- und Codefenster	420
16.1.3	»Design«-Ansicht	420
16.1.4	Projekt ausführen	421

16.1.5	Ein Label einfügen und gestalten	422
16.1.6	Weitere Widgets einfügen und gestalten	424
16.1.7	Code und Verbindung erstellen	424
16.1.8	Registerkarte »Signals und Slots«	428
16.2	Ein einfacher Kopfrechentrainer	429
16.2.1	Die Erstellung der GUI	430
16.2.2	Die Deklaration der Klasse	431
16.2.3	Die Definition ohne Prüfung	431
16.2.4	Die Definition mit Prüfung	433
16.3	Ein erweiterter Kopfrechentrainer	434
16.3.1	Ein Projekt kopieren	435
16.3.2	Die Erweiterung der GUI	436
16.3.3	Die Erweiterung der Deklaration	437
16.3.4	Die Erweiterung des Konstruktors	437
16.3.5	Die Erweiterung des Prüfvorgangs	438
16.3.6	Die Erweiterung der Aufgabenstellung	439
16.4	Weitere Widgets	441
16.4.1	Beschreibung der Widgets	441
16.4.2	Die Erstellung der GUI	442
16.4.3	Die Deklaration der Klasse	442
16.4.4	Der Konstruktor der Klasse	443
16.4.5	Die Ereignismethoden der Klasse	445
17	Datenbanken mit SQLite verwalten	449
17.1	Der Aufbau einer Datenbank	449
17.2	Wie erzeuge ich Datenbank und Tabelle?	450
17.2.1	Die Erstellung der GUI	450
17.2.2	Die Deklaration der Klasse	451
17.2.3	Konstruktor und Destruktor der Klasse	452
17.2.4	Die Ereignismethode der Klasse	454
17.3	Wie speichere ich Daten in einer Tabelle?	455
17.4	So zeige ich alle Daten einer Tabelle an	457
17.5	Wie wähle ich bestimmte Daten aus?	460
17.5.1	Vergleichsoperatoren	461

17.5.2	Logische Operatoren	462
17.5.3	Vergleichsoperator »LIKE«	462
17.6	Der Benutzer wählt Daten aus	464
17.7	Daten sollten sortiert werden	465
17.8	Wie ändere ich Daten?	466
17.9	Vorsicht beim Löschen von Daten	468
17.10	Eine Datenbank mit mehreren Tabellen	469
17.10.1	Inhalt der Datenbank	469
17.10.2	Das Datenbankmodell	470
17.10.3	Der Aufbau des Qt-Projekts	472
17.10.4	Das Erstellen und Füllen der Tabellen	472
17.10.5	Abfragen über einzelne Tabellen	473
17.10.6	Abfragen über zwei verbundene Tabellen	474
17.10.7	Abfragen über drei verbundene Tabellen	475
17.10.8	Gruppieren und Summieren	476
17.11	Projekt Vokabeln	476
17.11.1	Die Benutzung des Programms	477
17.11.2	Der Aufbau des Qt-Projekts	481
17.11.3	Die Deklaration der Klasse	482
17.11.4	Der Konstruktor der Klasse	484
17.11.5	Das Aktivieren und Deaktivieren von Widgets	486
17.11.6	Die Auswahl der Sprachkombination	487
17.11.7	Der Vokabeltest wird gestartet	489
17.11.8	Die Übersetzung wird geprüft	491
17.11.9	Der Vokabeltest wird vorzeitig beendet	492
17.11.10	Die Neueingabe von Vokabeln wird gestartet	492
17.11.11	Die List Widgets werden parallel markiert	494
17.11.12	Eine neue Vokabel wird gespeichert	494
17.11.13	Eine Vokabel wird gelöscht	496
17.11.14	Die Neueingabe von Vokabeln wird beendet	496
17.11.15	Mögliche Erweiterungen des Programms	497
17.12	Übung	497
17.12.1	Benutzeroberfläche	498
17.12.2	Bedienung	498
17.12.3	Zugriff auf Date Edit Widget	500
17.12.4	Eintrag im List Widget aufsplitten	500

Anhang

A	Installationen	501
A.1	Installationen unter Windows	501
A.2	Installationen unter Ubuntu Linux	512
A.3	Installationen unter macOS	516
A.4	Ubuntu Linux unter Windows installieren	520
B	Hilfestellungen und Übersichten	525
B.1	Windows – einige Tastenkombinationen	525
B.2	Unix-Befehle	525
B.3	Schlüsselwörter der Sprache C++	529
	Index	531