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++?

Objektorientierung

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*.

Grundlagen

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.

Beispiele und Übungen

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*.

Bibliotheken

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.

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.

C++-Referenz

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*.

1.4 So sieht das erste Programm aus

Aufbau

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.

Hallo Welt

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

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

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

Listing 1.1 Datei »hallo.cpp«

cout

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.

endl

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).

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«

/* */ und // 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

Zahlen vergleichen

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

```
#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.

Vier Fälle

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

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

Geschachtelt

Der erste Teil des Programms mit der geschachtelten Verzweigung:

```
#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

Leistung steigern

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.

3.4 Wie kann ich Bedingungen kombinieren?

Logische Operatoren

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;
```

```
   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«

Mehrfache Verzweigung

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

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 Programmlauf 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.

Modulo

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.


```
#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
```

Mehrere Bedingungen

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.

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 Dividenden, geteilt durch den Divisor. Ermitteln Sie zufällige Zahlen für das Ergebnis und den Divisor, und berechnen Sie den Dividenden durch die Multiplikation dieser beiden Werte. Ein Beispiel: Aus dem zufälligen Ergebnis 8 und dem zufälligen Divisor ergibt sich der Dividend 40. Dem Benutzer wird dann folgende Aufgabe gestellt: 40 / 5 = ...

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.

Boolesche Variable

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.

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.
- **0 = false** 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*.

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 Programmteils:

```
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 Programmteils:

```
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.

Operator ? :

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

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
```