Kapitel 2
Unser erstes Programm

Anhand des ersten Programms lernen Sie die Entwicklungsumgebung und die Grundbegriffe eines C-Programms kennen.

Hallo Welt

Bei unserem ersten Programm handelt es sich um einen Klassiker, mit dem viele Entwickler in vielen Programmiersprachen starten. Auf dem Bildschirm wird der folgende Text ausgegeben:

```
Hallo Welt
```

Sehen Sie als Entwickler diese Ausgabe vor sich, wissen Sie, dass Sie den ersten wichtigen Schritt bewältigt haben.

2.1 Wie gebe ich das Programm ein?

Code::Blocks

Starten Sie die Entwicklungsumgebung *Code::Blocks*. Die Installation unter *Windows* beschreibe ich in Abschnitt A.1, unter *Ubuntu Linux* in Abschnitt A.2 und unter *macOS* in Abschnitt A.3. Die Entwicklungsumgebung bietet unter anderem einen Editor, in dem Sie den Code Ihrer Programme eingeben können.

Neue Datei

Wählen Sie den Menüpunkt File • New • Empty File. Dies geht auch einfacher mithilfe der Tastenkombination Strg+⇧+N. Im rechten Teil erscheint ein neues Fenster, das zunächst den Titel Unbenannt1 hat. Geben Sie darin den Code des folgenden Listings ein (siehe auch Abbildung 2.1):

```
#include <stdio.h>
int main()
{
   printf("Hallo Welt\n");
   return 0;
}
```

Listing 2.1 Datei »hallo.c«

Sonderzeichen

Die vielen Sonderzeichen stellen eine erste Hürde dar. Die geschweiften Klammern { und } sowie das Backslash-Zeichen \ können Sie mithilfe der Tastenkombinationen Alt Gr+{, Alt Gr+} und Alt Gr+\ eingeben.

Klammern, Einrückungen

Nach der Eingabe einer geöffneten Klammer erscheint in vielen Entwicklungsumgebungen die passende schließende Klammer von selbst. Falls es sich um eine geschweifte Klammer handelt, wird die nächste Zeile automatisch eingerückt. Nach einer kurzen Eingewöhnung werden Sie feststellen, dass sich dieses selbstständige Verhalten des Editors als hilfreich erweist und Sie bei der Erstellung von gut lesbaren Programmen unterstützt. Diese Unterstützung erhalten Sie auch in anderen Entwicklungsumgebungen.

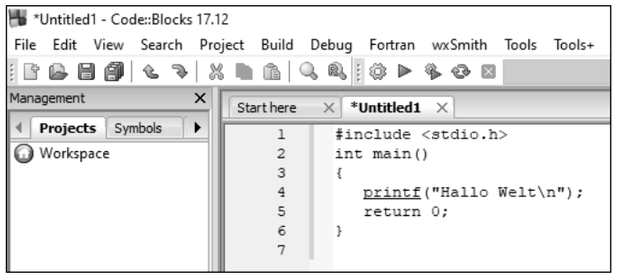

Abbildung 2.1 Erste Eingabe

Case sensitivity

In C wird nach Klein- und Großbuchstaben unterschieden. Dieses Verhalten wird auch *case sensitivity* genannt. Sie sollten also zum Beispiel `printf` nicht mit einem großen `P` schreiben. Dies wird in der Regel zu einem Fehler führen.

2.2 Was bedeuten die einzelnen Zeilen?

Bei der Erläuterung der Programmzeilen arbeiten wir uns, ein wenig ungewöhnlich, von innen nach außen vor.

printf()

In der Mitte des Programms finden Sie die Anweisung `printf("Hallo Welt\n");`. Sie dient zur Ausgabe des gewünschten Texts auf dem Bildschirm. Der Text wird in doppelte Anführungsstriche eingeschlossen. Das Sonderzeichen `\n` bewirkt einen anschließenden Zeilenumbruch. Das Steuerzeichen `n` steht für *new line*. Jede Anweisung muss mit einem Semikolon beendet werden.

Die restlichen Zeilen sehen momentan noch etwas abstrakt aus. Ihre Bedeutung wird aber im weiteren Verlauf des Buchs klarer. Sie dienen dazu, rund um die Anweisung `printf(...);` ein vollständiges C-Programm zu erstellen, das übersetzt und ausgeführt werden kann.

Die Anweisung `return 0;` liefert die Zahl 0 zurück. Für uns bedeutet das zunächst vereinfacht, dass kein Fehler aufgetreten ist.

main()

Beide genannten Anweisungen stehen innerhalb der Funktion `main()`. Jedes C-Programm besteht aus einer oder mehreren *Funktionen*, die unterschiedliche Namen haben und miteinander arbeiten. Die Funktion `main()` bildet stets den Startpunkt eines Programms. Nach dem Namen einer Funktion folgen stets runde Klammern. Alle Anweisungen einer Funktion müssen innerhalb eines *Blocks* stehen. Einen Block erkennen Sie an den geschweiften Klammern.

Die Angabe `int` besagt, dass die Funktion `main()` eine ganze Zahl (engl. *integer*) zurückliefert, in diesem Falle 0.

#include

Die Anweisung `#include` zu Beginn des Programms dient dazu, Funktionen aus bestimmten Bibliotheken für das Programm verfügbar zu machen. Über die sogenannte Header-Datei *stdio.h* erreichen Sie eine Bibliothek mit Standardfunktionen zur Ein- und Ausgabe. Die Abkürzung *stdio* steht dabei für ***st**andard **i**nput **o**utput*.

stdio.h

Über *stdio.h* wird unter anderem die Funktion `printf()` erreichbar gemacht. Ohne diese Header-Datei wären C-Programme nicht in der Lage, etwas auf dem Bildschirm auszugeben oder Angaben des Benutzers von der Tastatur einzulesen. Im weiteren Verlauf des Buchs werden wir noch Funktionen aus anderen Bibliotheken benötigen.

2.3 Das Programm wird gespeichert

Nach der Eingabe speichern Sie das Programm über den Menüpunkt FILE • SAVE FILE AS. Daraufhin erscheint das Dialogfeld SAVE FILE.

Dateiendung ».c«

Sie können mithilfe der verschiedenen Entwicklungsumgebungen sowohl in der Sprache C als auch in der Sprache C++ programmieren. Dateien mit Programmen in der Sprache C haben die Endung *.c*. Falls es sich um Programme in der Sprache C++ handelt, haben die Dateien die Endung *.cpp*. In diesem Buch programmieren wir in C, und unser Programm gibt ein »Hallo

Welt« aus, also geben Sie im Feld DATEINAME zum Beispiel den Namen »hallo.c« ein (siehe auch Abbildung 2.2).

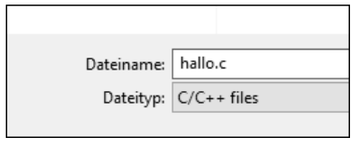

Abbildung 2.2 Dateinamen eingeben

C:\CProgramme

Wählen Sie ein Verzeichnis aus. Ich habe *C:\CProgramme* ausgewählt und beziehe mich auch in späteren Anleitungen darauf. Betätigen Sie die Schaltfläche SPEICHERN.

Einige nützliche Hinweise

Schließen

- Sie können die Datei schließen, und zwar über den Menüpunkt FILE • CLOSE FILE oder mit der Tastenkombination Strg+W.

Öffnen

- Sie können eine vorhandene Datei erneut öffnen: über den Menüpunkt FILE • OPEN oder mit der Tastenkombination Strg+O.
- Sie können an mehreren Programmen parallel arbeiten, falls Sie zum Beispiel einzelne Elemente eines Programms in ein anderes Programm übernehmen möchten. Jedes Programm erscheint rechts in einem eigenen Fenster.

2.4 Wie starte ich das fertige Programm?

Übersetzen

Ich gehe davon aus, dass Sie die Datei gespeichert und aktuell geöffnet haben. Sie können das Programm über den Menüpunkt BUILD • BUILD AND RUN (Funktionstaste F9) in die Sprache des Rechners übersetzen, zu einem lauffähigen Programm zusammensetzen und starten. Der Vorgang des Übersetzens wird auch *kompilieren* genannt. Der entsprechende Bestandteil der Entwicklungsumgebung nennt sich *Compiler*.

Syntax

Beim Kompilieren können *Syntaxfehler* festgestellt werden. Unter der *Syntax* eines Programms versteht man seinen Aufbau gemäß den Regeln der Programmiersprache. Falls das Programm keinen Syntaxfehler enthält, erscheint unten in der Registerkarte BUILD LOG eine Ausgabe wie in Abbildung 2.3. Unter anderem steht dort `0 error(s)` und `0 warning(s)`.

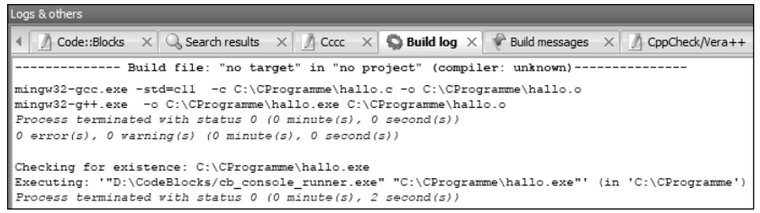

Abbildung 2.3 Erfolgreiche Übersetzung

Fehlermeldung

Sollte das Programm Syntaxfehler aufweisen, erscheinen eine oder mehrere Fehlermeldungen und es startet nicht. Zur Demonstration habe ich einmal das Semikolon am Ende der ersten Anweisung weggelassen. Die Fehlermeldung besagt, dass in Zeile 5 ein Semikolon vor `return` erwartet wird. Der Fehler, der in Zeile 4 gemacht wird, wird also erst in Zeile 5 bemerkt (siehe Abbildung 2.4). Nach Beseitigung des Fehlers muss das Programm erneut übersetzt und gestartet werden.

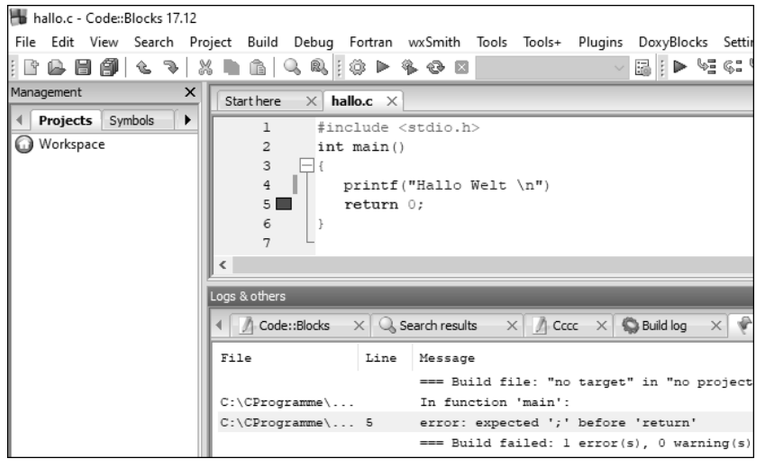

Abbildung 2.4 Übersetzung mit Fehlermeldung

Hinweis

Es ist häufig so, dass sich Fehler erst in späteren Programmzeilen auswirken. Falls Sie also nach der Ursache eines Fehlers suchen, beginnen Sie in der Zeile mit der Fehlermeldung und arbeiten sich anschließend Zeile für Zeile nach oben.

Nach einer erfolgreichen Übersetzung wird die Ausgabe des Programms in einem *Konsolenfenster* angezeigt (siehe Abbildung 2.5). Nach einem Tastendruck wird das Konsolenfenster wieder geschlossen.

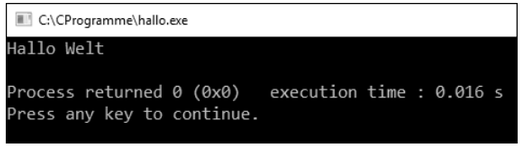

Abbildung 2.5 Ausgabe des Programms

Hinweis

Wir schreiben zunächst nur kleinere Programme. Größere Programme bestehen häufig aus mehreren Dateien und werden in einem zusammenhängenden Projekt gespeichert. Ausführliche Erläuterungen zur Erstellung von Projekten in den verschiedenen Entwicklungsumgebungen finden Sie in Anhang A.

2.5 Eine weitere Möglichkeit zum Starten

Diesen Abschnitt können Sie zunächst überspringen und später bei Bedarf lesen. Aber vielleicht würden Sie gerne einmal Folgendes ausprobieren:

- Ihr Programm direkt von der Kommandozeile aus aufrufen
- Ihr Programm auf einem anderen *Windows*-PC nutzen

Ausführbare Datei

Sie haben mithilfe der Entwicklungsumgebung aus dem C-Programm in der Datei *hallo.c* bereits ein ausführbares Programm in der Datei *hallo.exe* erstellt. Es befindet sich im selben Verzeichnis wie die Datei *hallo.c*, also in *C:\CProgramme*. Im Kommandozeilenmodus können Sie es direkt aus diesem Verzeichnis heraus aufrufen.

Hinweis

Falls Sie in einer der verschiedenen Entwicklungsumgebungen ein C-Projekt angelegt haben, finden Sie das ausführbare Programm in einem der Unterverzeichnisse des Projekts.

Kommandozeile

Unter *Windows* erreichen Sie den Kommandozeilenmodus zum Beispiel über die Tastenkombination ⊞+R und die anschließende Eingabe des Befehls `cmd`.

Ausführen

Wechseln Sie in das Verzeichnis *C:\CProgramme*, und zwar wie in Abbildung 2.6 mit dem Befehl `cd C:\CProgramme`. Lassen Sie sich mit dem Befehl `dir hallo*` alle Dateien im Verzeichnis anzeigen, deren Name mit `hallo` beginnt. Starten Sie das ausführbare Programm in der Datei *hallo.exe* – dazu dient der Befehl `hallo`. Es erscheint die Ausgabe des Programms: **`Hallo Welt`**.

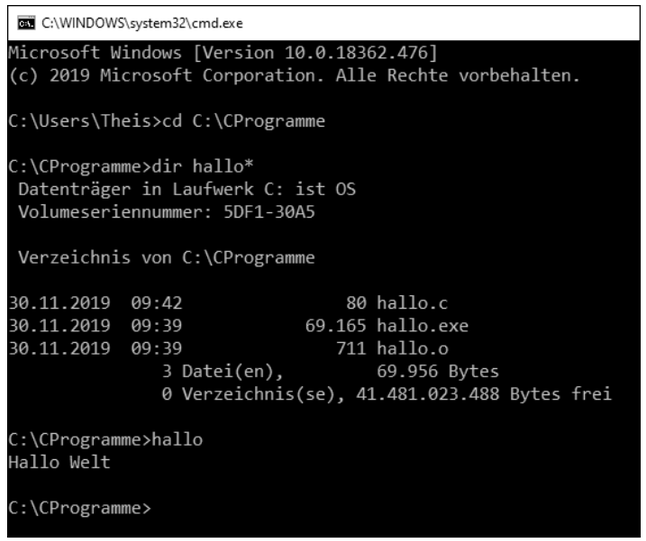

Abbildung 2.6 Eingabeaufforderung mit Programmaufruf

Programm kopieren

Sie können die Datei *hallo.exe* auf einen anderen *Windows*-PC kopieren, etwa mithilfe eines USB-Sticks. Anschließend können Sie das Programm auf dem anderen *Windows*-PC starten.

Codeseite

Standardmäßig ist unter *Windows* die Codeseite 850 eingestellt. Falls Sie die deutschen Umlaute und das scharfe S (ß) ausgeben möchten, sollten Sie im Kommandozeilenfenster mithilfe des Befehls `chcp 1252` auf die Codeseite 1252 umstellen.

2.6 Kommentare sind wichtig

Gute Entwickler kommentieren den Code ihrer Programme. Auf diese Weise erläutern sie den Zweck und den Aufbau wichtiger Bestandteile des Programms für sich selbst und für andere Entwickler.

Weitere Entwicklung

Häufig nimmt sich ein Entwickler ein Programm nach Wochen oder Monaten noch einmal vor, um es zu verbessern oder ein anderes Programm auf seiner Basis zu erschaffen. Oder er muss ein Programm bearbeiten, das ursprünglich von einem anderen Entwickler erstellt wurde. In diesen Fällen ist es sehr hilfreich, wenn die Programme gut kommentiert sind.

/* Kommentar */

Kommentare werden nicht übersetzt und sind nur in der Datei mit dem C-Quellcode sichtbar. Sie stehen zwischen den Zeichenfolgen /* und */ und können sich über mehrere Zeilen erstrecken.

// Kommentar

Seit dem Sprachstandard C99 sind auch einzeilige Kommentare möglich. Sie beginnen nach der Zeichenfolge // und erstrecken sich bis zum Ende der Zeile.

Ein Beispiel:

```
#include <stdio.h>
int main()
{
   /* Es gibt lange Kommentare.
      Diese stehen in mehreren Zeilen */
   printf("Es gibt auch lange"
      " Texte.\nDiese stehen"
      " auch in mehreren Zeilen.\n");

   return 0;  // Kommentar bis zum Ende der Zeile, seit C99
}
```

Listing 2.2 Datei »kommentar.c«

Zu Beginn der Funktion `main()` steht ein Kommentar über zwei Zeilen. Nach der Anweisung mit `return` steht ein einzeiliger Kommentar.

Langer Text

Gleichzeitig sehen Sie in diesem Programm, wie Sie einen längeren Ausgabetext unterteilen können. Die einzelnen Textstücke stehen jeweils innerhalb von doppelten Anführungsstrichen. Die Zeichenfolge `\n` sorgt für einen Zeilenumbruch. Die Ausgabe sieht wie folgt aus:


```
Es gibt auch lange Texte.
Diese stehen auch in mehreren Zeilen.
```


Hinweis

Kommentare geben Ihnen auch die Möglichkeit, bestimmte Teile eines Programms zeitweilig von der Übersetzung auszuschließen. Dies kann sich während der Entwicklung eines größeren Programms oder bei der Fehlersuche als nützlich erweisen.

2.7 Eine Übungsaufgabe

Schreiben Sie ein Programm in der Datei *u_name.c*. Es soll die folgende Ausgabe erzeugen:

```
Guten Morgen.
Mein Name ist Claus Clever.
```

Kommentieren Sie jede Zeile innerhalb der Funktion `main()`. Eine mögliche Lösung zu der Übungsaufgabe finden Sie in Abschnitt C.1. Denken Sie daran: Es kann zu jeder Aufgabe *beliebig viele* richtige Lösungen geben. Falls Ihre Lösung von meiner Lösung abweicht, überlegen Sie, welche Lösung für Sie verständlicher ist.

Kapitel 3
Verschiedene Arten von Daten

Sie speichern Zahlenwerte, geben sie auf dem Bildschirm aus, lesen sie von der Tastatur ein und rechnen mit ihnen. Außerdem lernen Sie, wie Sie Fehler leichter finden können.

Innerhalb eines Programms müssen unterschiedliche Daten ermittelt, gespeichert, weiterverarbeitet und ausgegeben werden. Dies können Zahlen, aber auch Texte sein.

Ein Einkauf

In vielen Beispielen dieses Buchs wird mit den Daten eines alltäglichen, leicht verständlichen Anwendungsbeispiels gearbeitet: Wir tätigen einen Einkauf in einem Geschäft. Schritt für Schritt lernen Sie weitere Programmierelemente kennen, die Ihnen die Möglichkeit geben, diesen Einkauf und die Erstellung der zugehörigen Rechnung immer realistischer zu gestalten.

Sie lernen zunächst die verschiedenen Programmierelemente kennen und setzen sie in einem anschaulichen Zusammenhang ein. Später, ab Kapitel 12, zeige ich Ihnen ihre vielfältigen Möglichkeiten.

3.1 Daten bekannt machen und speichern

Zahlenarten

Nehmen wir an, Sie kaufen zwei Äpfel. Jeder Apfel kostet 1,45 Euro, da es sich um besonders gute Äpfel aus zertifiziertem Bio-Anbau handelt. Damit haben wir unsere ersten Daten. Die Anzahl der gekauften Äpfel ist eine ganze Zahl. Beim Preis der Äpfel handelt es sich um eine Zahl mit Stellen nach dem Komma.

Datentypen

Einzelne Daten werden in *Variablen* gespeichert. Variablen können ihren Wert verändern. In C gibt es unterschiedliche Typen von Variablen. Jeder dieser sogenannten *Datentypen* hat einen eigenen Einsatzzweck und bietet bestimmte Vorteile. Wir beschränken uns zunächst auf die Datentypen `int` für ganze Zahlen und `double` für Zahlen mit Nachkommastellen.

Ein Beispiel:

```
#include <stdio.h>
int main()
{
   /* Deklaration */
   int anzahl;
   double preis;

   /* Zuweisung */
   anzahl = 2;
   preis = 1.45;

   return 0;
}
```

Listing 3.1 Datei »daten.c«

Deklaration

Eine *Deklaration* dient dazu, einer Variablen einen Namen zu geben und sie einem Datentyp zuzuordnen. Eine Variable kann erst nach einer Deklaration verwendet werden, daher stehen Deklarationen normalerweise zu Beginn.

int, double

In der Variablen `anzahl` vom Typ `int` kann eine ganze Zahl gespeichert werden, in der Variablen `preis` vom Typ `double` eine Zahl mit Nachkommastellen.

Zuweisung

Eine Variable kann per *Zuweisung* einen Wert erhalten. Eine Zuweisung wird mithilfe des Zeichens `=` notiert. Die Variable auf der linken Seite erhält damit den Wert, der auf der rechten Seite des Gleichheitszeichens steht.

Vor der ersten Zuweisung hat eine Variable einen zufälligen Wert. Im Verlauf des Programms kann sich der Wert einer Variablen beliebig oft ändern. Sie könnten nun schon das Programm aus Listing 3.1 übersetzen und ausführen. Allerdings erzeugt es noch keine Ausgabe auf dem Bildschirm.

Namensregeln

Einige Regeln für die Namen der Variablen:

- Es sind nur Buchstaben, Ziffern und der Unterstrich gestattet.
- Die deutschen Umlaute und das scharfe S (ß) gehören nicht zu den erlaubten Buchstaben.
- Der Name darf nicht mit einer Ziffer beginnen.

- Eines der *Schlüsselwörter* der Sprache C (siehe Abschnitt B.3) darf nicht als Name verwendet werden.

Aussagekräftige Namen

Erleichtern Sie sich die Entwicklung: Geben Sie den Variablen aussagekräftige Namen. Sie können den Preis eines eingekauften Artikels auch in der Variablen `x` speichern. In diesem Fall weiß aber später niemand mehr, was sich dahinter verbirgt. Falls Sie zwei verschiedene Preise speichern möchten, nehmen Sie zum Beispiel die Namen `preisApfel` und `preisBanane`. Diese gut lesbare Schreibweise nennt sich *camelCase*, wegen des »Kamelhöckers« mitten im Wort.

3.2 Wie gebe ich Daten auf dem Bildschirm aus?

Platzhalter %d und %f

Sie können zu jedem Zeitpunkt des Programms den aktuellen Wert einer Variablen auf dem Bildschirm ausgeben. Dazu werden sogenannte *Platzhalter* benötigt. Diese stehen bei der Ausgabe mithilfe der Funktion `printf()` genau an der Stelle des Textes, an der der Wert ausgegeben werden soll. Variablen vom Typ `int` können mit dem Platzhalter `%d` (oder auch mit dem Platzhalter `%i`) ausgegeben werden. Bei Variablen vom Datentyp `double` nimmt man den Platzhalter `%f`.

Ein Beispiel:

```
#include <stdio.h>
int main()
{
   /* Deklaration mit Initialisierung */
   int anzahl = 2;
   double preis = 1.45;

   /* Ausgabe */
   printf("Anzahl: %d\n", anzahl);
   printf("Preis: %f Euro\n", preis);
   printf("Preis: %.2f Euro\n", preis);

   return 0;
}
```

Listing 3.2 Datei »daten_aus.c«

Initialisierung

Diesmal erhalten die Variablen bereits bei der Deklaration einen Wert. Dieser Vorgang wird *Initialisierung* genannt und macht das Programm ein wenig kompakter. Die Werte der Variablen können sich nach wie vor im Verlauf des Programms ändern.

Das Programm hat die folgende Ausgabe:

```
Anzahl: 2
Preis: 1.450000 Euro
Preis: 1.45 Euro
```

Formatierte Ausgabe

Der Wert der Variablen `anzahl` steht am Ende der ersten Ausgabe. Der Wert der Variablen `preis` wird jeweils mitten in der zweiten und dritten Ausgabe eingebettet. Ohne genauere Angabe führt `%f` zu einer Formatierung der Ausgabe auf sechs Nachkommastellen. Durch die Angabe `%.2f` findet eine Formatierung auf zwei Nachkommastellen statt.

Dezimalpunkt

Beachten Sie die englische Schreibweise: Wir verwenden in C immer einen Dezimalpunkt anstelle des deutschen Dezimalkommas.

3.3 Wie kann der Benutzer seine Daten per Tastatur eingeben?

scanf()

Ein Programm wird erst interaktiv, wenn sein Benutzer die Möglichkeit hat, Eingaben zu machen. Dazu setzen Sie als Entwickler die Funktion `scanf()` ein. Was bewirkt sie?

- Das Programm hält an und wartet auf eine Eingabe. Sie sollten dem Benutzer vorher mitteilen, welche Information von ihm an dieser Stelle erwartet wird.
- Nach der Eingabe per Tastatur betätigt der Benutzer die Taste ↵. Erst danach läuft das Programm weiter, speichert die eingegebenen Daten in Variablen und verarbeitet sie im weiteren Verlauf des Programms.

Adressoperator

Auch bei `scanf()` werden Platzhalter benötigt, wie bei der Funktion `printf()`. Bei Variablen des Datentyps `int` können Sie weiterhin `%d` verwenden. Eine `double`-Variable muss allerdings mit `%lf` eingelesen werden, also mit dem kleinen Buchstaben `l` vor dem `f`. Außerdem muss vor dem Namen der Variablen das Zeichen `&` angegeben werden. Dabei handelt es sich um den sogenannten *Adressoperator*, der auf die Adresse der Variablen im Speicher verweist.

Ein Beispiel:

```
#include <stdio.h>
int main()
{
   /* Deklaration */
   int anzahl;
   double preis;

   /* Erste Eingabe */
   printf("Anzahl eingeben: ");
   scanf("%d", &anzahl);

   /* Zweite Eingabe */
   printf("Preis in Euro eingeben: ");
   scanf("%lf", &preis);

   /* Ausgabe */
   printf("Anzahl: %d\n", anzahl);
   printf("Preis: %.2f Euro\n", preis);

   return 0;
}
```

Listing 3.3 Datei »daten_ein.c«

Die Ausgabe des Programms könnte, mit einigen Beispieleingaben des Benutzers, wie folgt aussehen:

```
Anzahl eingeben: 3
Preis in Euro eingeben: 1.35
Anzahl: 3
Preis: 1.35 Euro
```

Die Funktion `scanf()` bietet einige Stolperfallen für den Benutzer:

- Er muss bei der Eingabe wiederum die englische Schreibweise beachten, also mit einem Dezimalpunkt statt einem Dezimalkomma. **Dezimalpunkt**
- Sollte er mehrere Eingaben auf einmal vornehmen, Buchstaben statt Ziffern eingeben oder ein Komma statt eines Punkts, führt dies leider zu einem unerwarteten Programmverlauf. **Falsche Eingaben**

Abfangen

Sie werden in Abschnitt 12.6 Möglichkeiten kennenlernen, mit denen Sie als Entwickler falsche Eingaben des Benutzers abfangen können. Zunächst gehen wir davon aus, dass er richtige Eingaben vornimmt.

Aber auch Sie als Entwickler könnten Fehler machen. Hier sind einige Tipps, die Sie beim Einsatz der Funktion `scanf()` beachten sollten:

- Innerhalb der Anführungsstriche sollte allein der Platzhalter notiert werden, also nur `%d` oder `%lf`, nicht etwa `%d\n` oder `%.2lf`.
- Vergessen Sie nicht, den Adressoperator `&` zu notieren.

Fehler bei der Ausführung

Das Gefährliche an den genannten Fehlern des Entwicklers: Es sind keine Syntaxfehler. Das Programm wird also *leider* übersetzt, ohne sichtbare Fehlermeldung. Erst während der Laufzeit des Programms treten Fehler auf, weil zum Beispiel eine Variable nicht den richtigen Eingabewert erhalten hat und in der Folge ein unverständliches Ergebnis auftritt.

Vielfalt

Sie werden sich fragen: Wieso meldet der Compiler diese Fehler nicht als Syntaxfehler? Der Grund liegt unter anderem in der Vielfalt der Programmierung mit C. Es gibt auch Operationen für andere Einsatzzwecke, die genau diese Schreibweise erfordern, die hier zunächst als falsch erscheint.

3.4 Berechnungen mit Operatoren

Grundrechenarten

In diesem Abschnitt führen wir einige Berechnungen im Bereich der Grundrechenarten durch. Wir benutzen die Operatoren `+` für die Addition, `-` für die Subtraktion, `*` für die Multiplikation und `/` für die Division.

Punkt vor Strich

Wie in der Mathematik gilt: *Punktrechnung vor Strichrechnung*. Die Operatoren `*` und `/` haben also Vorrang vor den Operatoren `+` und `-`. Sie können zusätzlich Klammern setzen, um die Reihenfolge der Berechnung zu beeinflussen. Sie finden in Abschnitt 12.1.9 eine Tabelle mit den Vorrangregeln für die Operatoren.

Zunächst das Programm:

```
#include <stdio.h>
int main()
{
   /* Deklaration, teilweise mit Initialisierung */
   int anzahlApfel = 2, anzahlBirne = 4;
```



```
   double preisApfel = 1.45, preisBirne = 0.85;
   double summeRechnung, preisMittel, preisDifferenz;

   /* Berechnung */
   summeRechnung = anzahlApfel * preisApfel
      + anzahlBirne * preisBirne;
   preisMittel = summeRechnung / (anzahlApfel + anzahlBirne);
   preisDifferenz = preisApfel - preisBirne;

   /* Ausgabe */
   printf("Summe der Rechnung: %.2f Euro\n", summeRechnung);
   printf("Mittlerer Preis: %.2f Euro\n", preisMittel);
   printf("Preis-Differenz: %.2f Euro\n", preisDifferenz);

   return 0;
}
```

Listing 3.4 Datei »daten_rechnen.c«

Was passiert bei diesem Einkaufsvorgang?

- Es werden zwei verschiedene Artikel eingekauft, in unterschiedlicher Anzahl. Daher werden weitere Variablen benötigt. Mehrere Variablen desselben Typs können Sie durch Komma getrennt innerhalb einer Anweisung deklarieren und auch initialisieren. **Mehrere Deklarationen**
- Es werden Variablen für die Ergebnisse der Rechenoperationen deklariert.
- Zur Berechnung der Summe der Rechnung werden die Einzelsummen pro Artikel ermittelt. Anschließend werden diese Einzelsummen addiert. **Summe**
- Der mittlere Preis eines Artikels dieses Einkaufsvorgangs ergibt sich, indem wir die Summe der Rechnung durch die Gesamtanzahl der Artikel teilen. Wir müssen Klammern setzen, da ansonsten die Division Vorrang hätte. **Mittelwert**
- Als Letztes wird die preisliche Differenz der beiden Artikel ermittelt. **Differenz**

Es ergibt sich die folgende Ausgabe:

```
Summe der Rechnung: 6.30 Euro
Mittlerer Preis: 1.05 Euro
Preis-Differenz: 0.60 Euro
```

Daten umwandeln

Eine Rechenoperation, an der eine ganze Zahl und eine Zahl mit Nachkommastellen beteiligt sind, ergibt in C eine Zahl mit Nachkommastellen. Ein Beispiel: Bei der Berechnung des mittleren Preises pro Artikel wird eine `double`-Variable durch die Summe von zwei `int`-Variablen geteilt. Es ergibt sich ein `double`-Wert.

Ganzzahldivision

Hinweis

Falls an einer Division zwei ganze Zahlen beteiligt sind, ergibt sich wiederum eine ganze Zahl. Sollten sich Nachkommastellen ergeben, werden diese abgeschnitten. Ein Beispiel: 11 durch 4 ergibt den Wert 2 und nicht etwa 2,75, wie es mathematisch richtig wäre.

Als Abhilfe kann zum Beispiel Folgendes dienen:

- Sie schreiben `11.0 / 4` oder `11 / 4.0`, geben also eine der beiden Zahlen als `double`-Wert an.
- Sie schreiben `1.0 * 11 / 4`, multiplizieren also einen `double`-Wert mit der ersten Zahl.

3.5 Entwicklung eines Programms

Teile eines Programms

Bei der Entwicklung Ihrer eigenen Programme sollten Sie Schritt für Schritt vorgehen. Stellen Sie zuerst einige Überlegungen an, wie das gesamte Programm aufgebaut sein sollte, und zwar auf Papier. Aus welchen Teilen sollte es nacheinander bestehen? Versuchen Sie nicht, das gesamte Programm mit all seinen komplexen Bestandteilen auf einmal zu schreiben! Dies ist der größte Fehler, den Einsteiger (und manchmal auch Fortgeschrittene) machen können.

Einfache Version

Schreiben Sie zunächst eine einfache Version des ersten Programmteils. Anschließend testen Sie sie. Erst nach einem erfolgreichen Test fügen Sie den folgenden Programmteil hinzu. Nach jeder Änderung testen Sie wiederum. Sollte sich ein Fehler zeigen, so wissen Sie, dass er aufgrund der letzten Änderung aufgetreten ist. Nach dem letzten Hinzufügen haben Sie eine einfache Version Ihres gesamten Programms.

Komplexere Version

Nun ändern Sie einen Teil Ihres Programms in eine komplexere Version ab. Auf diese Weise machen Sie Ihr Programm Schritt für Schritt komplexer, bis Sie schließlich das gesamte Programm so erstellt haben, wie es Ihren anfänglichen Überlegungen auf Papier entspricht.

Änderungen

Manchmal ergibt sich während der praktischen Programmierung noch die eine oder andere Änderung gegenüber Ihrem Entwurf. Das ist kein Problem, solange sich nicht der gesamte Aufbau ändert. Sollte dies allerdings der Fall sein, so kehren Sie noch einmal kurz zum Papier zurück und überdenken den Aufbau. Das bedeutet nicht, dass Sie die bisherigen Programmzeilen löschen müssen, sondern möglicherweise nur ein wenig ändern und anders anordnen.

Einzelne Schritte

Schreiben Sie Ihre Programme übersichtlich. Falls Sie gerade überlegen, wie Sie drei, vier bestimmte Schritte Ihres Programms auf einmal machen können: Machen Sie daraus einfach einzelne Anweisungen, die der Reihe nach ausgeführt werden. Dies vereinfacht eine eventuelle Fehlersuche. Falls Sie (oder jemand anders) Ihr Programm später einmal ändern oder erweitern möchten, gelingt der Einstieg in den Aufbau des Programms wesentlich schneller.

Sie können die Funktion `printf()` und das nachfolgend beschriebene Debugging zur Kontrolle von Werten und zur Suche von logischen Fehlern einsetzen. Zusätzlich können Sie einzelne Zeilen Ihres Programms als Kommentar kennzeichnen, um festzustellen, welcher Teil des Programms fehlerfrei läuft und welcher Teil demnach fehlerbehaftet ist.

3.6 Fehler suchen

Debugging

Sollte eines Ihrer Programme unerwartete Ergebnisse haben, so müssen Sie sich auf die Suche nach dem Fehler machen und diesen beseitigen. Die verschiedenen Entwicklungsumgebungen bieten Ihnen einige Möglichkeiten für dieses sogenannte *Debugging*. Ich beschreibe diese Vorgänge anhand der Entwicklungsumgebung *Code::Blocks*. In den anderen Umgebungen funktioniert die Fehlersuche auf vergleichbare Art und Weise.

Werte kontrollieren

Das Debugging möchte ich Ihnen zu diesem frühen Zeitpunkt vorstellen, damit Sie sich an die Handhabung gewöhnen. Sie können Ihr Programm schrittweise ablaufen lassen, um die Werte bestimmter Variablen zu unterschiedlichen Zeitpunkten des Programms zu kontrollieren. Das Programm muss dazu in einem Projekt eingebunden sein. Zur Erstellung eines Projekts in *Code::Blocks* verweise ich auf Abschnitt A.1.2.

Haltepunkt

Klicken Sie einmal auf den grauen Balken vor einer Programmzeile. Es wird ein roter Punkt angezeigt. Die Zeile dient nun als *Haltepunkt* (engl. *break-*

point). Das Gleiche erreichen Sie über den Menüpunkt DEBUG • TOGGLE BREAKPOINT oder die Funktionstaste F5. Sie können mehrere Haltepunkte setzen. Ein erneuter Klick in derselben Programmzeile entfernt den Haltepunkt. In Abbildung 3.1 sehen Sie im Projekt *daten_rechnen* in der C-Datei *daten_rechnen.c* zwei Haltepunkte.

Bei der normalen Ausführung des Programms mithilfe des Menüpunkts BUILD • BUILD AND RUN (Funktionstaste F9) ändert sich durch die Haltepunkte noch nichts.

```
10      /* Berechnung */
11      summeRechnung = anzahlApfel * preisApfel
12         + anzahlBirne * preisBirne;
13      preisMittel = summeRechnung / (anzahlApfel + anzahlBirne);
14      preisDifferenz = preisApfel - preisBirne;
15
16      /* Ausgabe */
17      printf("Summe der Rechnung: %.2f Euro\n", summeRechnung);
18      printf("Mittlerer Preis: %.2f Euro\n", preisMittel);
19      printf("Preis-Differenz: %.2f Euro\n", preisDifferenz);
20
```

Abbildung 3.1 Zwei Haltepunkte

Programm hält an

Falls Sie aber den Menüpunkt DEBUG • START / CONTINUE wählen (Funktionstaste F8), hält das Programm vor dem Ausführen der Zeile 11 mit dem ersten Haltepunkt an. Falls es schon Ausgaben gegeben hat, so stehen diese bereits im Ausgabefenster.

Watch

Mithilfe des Menüpunkts DEBUG • DEBUGGING WINDOWS • WATCHES können Sie ein WATCH-Fenster einblenden, in dem die aktuellen Werte der Variablen angezeigt werden (siehe Abbildung 3.2).

Watches (new)

Function arguments	
⊟ Locals	
anzahlApfel	2
anzahlBirne	4
preisApfel	1.45
preisBirne	0.84999999999999998
summeRechnung	1.7911146116081608e-307
preisMittel	2.2818816367587726e+255
preisDifferenz	-5.2860771615310328e+208

Abbildung 3.2 Aktuelle Werte

Zeile für Zeile

Die ersten vier Variablen haben ihre Werte bereits zu Beginn erhalten. Zum Zeitpunkt des Anhaltens vor Zeile 11 haben die drei restlichen Variablen

noch keinen Wert erhalten, daher haben sie zufällige Werte. Mithilfe des Menüpunkts DEBUG • NEXT LINE (Funktionstaste F7) können Sie das Programm Zeile für Zeile durchlaufen und die Änderungen der Werte im WATCH-Fenster beobachten. Diese Vorgehensweise wird auch *Einzelschrittverfahren* genannt.

Kritische Stellen

Falls Sie den Menüpunkt DEBUG • START / CONTINUE (Funktionstaste F8) erneut ausführen, hält das Programm erst wieder vor dem nächsten Haltepunkt an, sofern vorhanden. Auf diese Weise können Sie kritische Stellen im Programm ansteuern, ohne jeden Schritt einzeln zu durchlaufen.

Beenden

Die Ausführung des Menüpunkts DEBUG • STOP DEBUGGER (Tastenkombination ⇧+F8) beendet den Debugger.

3.7 Eine Übungsaufgabe

Schreiben Sie ein Programm in der Datei *u_daten.c*. Es soll die folgende Ausgabe ermöglichen (hier mit Eingaben des Benutzers):

```
Erster Artikel, Anzahl: 5
Erster Artikel, Preis in Euro: 2.10

Zweiter Artikel, Anzahl: 2
Zweiter Artikel, Preis in Euro: 1.60

Summe ohne Rabatt: 13.70 Euro
Summe mit Rabatt: 10.96 Euro
```

Rabatt berechnen

Der Benutzer soll also Anzahl und Preis für zwei verschiedene eingekaufte Artikel eingeben. Anschließend wird die Summe der Rechnung ermittelt. Außerdem wird ein Rabatt von 20 % gewährt. Als Letztes soll die Summe einschließlich des eingerechneten Rabatts ausgegeben werden. Wählen Sie geeignete Variablennamen, und kommentieren Sie Ihr Programm ausreichend. Eine mögliche Lösung finden Sie in Abschnitt C.2.

Kapitel 4
Verschiedene Fälle in einem Programm

Ein Programm kann unterschiedliche Anweisungen durchlaufen, abhängig vom Wahrheitswert einer oder mehrerer Bedingungen.

In den bisherigen Programmen werden alle Anweisungen Zeile für Zeile ausgeführt. Ein Programm soll aber auch in der Lage sein, auf unterschiedliche Situationen zu reagieren. Sie als Entwickler können nicht wissen, was der Benutzer eingibt oder welche Inhalte eine Datei bietet, die Daten für Ihr Programm bereitstellt.

Verzweigung

In diesen Fällen sollte Ihr Programm verzweigen. Es sollte abhängig von der Situation unterschiedliche Anweisungen durchlaufen. Eine Verzweigung wird mithilfe einer Bedingung formuliert. Eine Bedingung hat einen Wahrheitswert. Dieser ist entweder *wahr* oder *falsch*.

4.1 Eine einfache Bedingung mit »if«

Wahr

Sie können eine Verzweigung durch das Schlüsselwort `if` einleiten. Nach dem `if` folgt in runden Klammern eine Bedingung. Ist diese Bedingung *wahr*, wird die folgende Anweisung oder der folgende Block von Anweisungen ausgeführt. Eine Bedingung bilden Sie mithilfe eines Vergleichsoperators.

Anweisungsblock

Einen Block von Anweisungen erzeugen Sie mithilfe von geschweiften Klammern. Jede Funktion, also auch die Funktion `main()`, enthält einen solchen Block von Anweisungen.

Ein Beispiel:

```
#include <stdio.h>
int main()
{
   double preis = 0.85;
```

```
   /* Bedingte Ausführung eines Blocks von Anweisungen */
   if(preis > 0.99)
   {
      printf("Ein teurer Artikel\n");
      printf("Brauchen wir den wirklich?\n");
   }

   /* Bedingte Ausführung einer einzelnen Anweisung */
   if(preis < 1.0)
      printf("Ein billiger Artikel\n");

   return 0;
}
```

Listing 4.1 Datei »fall_if.c«

Vergleichsoperatoren

In Listing 4.1 werden die beiden Vergleichsoperatoren > (größer als) und < (kleiner als) verwendet. Liegt der Preis über 99 Cent, handelt es sich um einen teuren Artikel und es wird der Block mit den beiden Anweisungen zur Ausgabe durchgeführt. Liegt der Preis unterhalb von einem Euro, ist der Artikel billig und es wird die einzelne Anweisung ausgeführt.

Das Programm erzeugt die folgende Ausgabe:

```
Ein billiger Artikel
```

Ändern Sie einmal den Preis so ab, dass die andere Ausgabe erfolgt.

Einrückung

Eine Empfehlung: Erhöhen Sie die Lesbarkeit Ihrer Programme, indem Sie die Anweisung(en) innerhalb einer Verzweigung einrücken, wie Sie dies im vorliegenden Programm sehen. Viele Editoren nehmen eine solche Einrückung bereits automatisch vor.

> **Hinweis**
>
> Achten Sie darauf, nach der Bedingung kein Semikolon zu notieren. Dies würde die Verzweigung unmittelbar beenden. Der Compiler meldet diesen Fehler nicht. Die anschließende Anweisung wäre nicht mehr mit der Verzweigung verbunden und würde stets ausgeführt.

4.2 Welche Bedingungen gibt es?

`> < >= <= == !=`

Beim Vergleich von Zahlen stehen Ihnen sechs verschiedene Vergleichsoperatoren zur Verfügung. Neben > und < gibt es vier Operatoren, die jeweils aus zwei Zeichen bestehen: >= (größer als oder gleich), <= (kleiner als oder gleich), == (gleich) und != (ungleich).

Alle Vergleichsoperatoren werden in folgendem Beispiel genutzt:

```
#include <stdio.h>
int main()
{
   double preisApfel = 1.45, preisBirne = 0.85, preisKiwi = 1.45;

   /* Größer */
   if(preisApfel > preisBirne)
      printf("Apfel ist teurer als Birne\n");

   /* Kleiner */
   if(preisBirne < preisKiwi)
      printf("Birne ist billiger als Kiwi\n");

   /* Größer oder gleich */
   if(preisApfel >= preisKiwi)
      printf("Apfel ist mindestens so teuer wie Kiwi\n");

   /* Kleiner oder gleich */
   if(preisBirne <= preisKiwi)
      printf("Birne ist hoechstens so teuer wie Kiwi\n");

   /* Gleich */
   if(preisApfel == preisKiwi)
      printf("Apfel und Kiwi haben denselben Preis\n");

   /* Ungleich */
   if(preisBirne != preisKiwi)
      printf("Birne und Kiwi haben unterschiedliche Preise\n");

   return 0;
}
```

Listing 4.2 Datei »fall_vergleich.c«

In Listing 4.2 werden die Preise von drei unterschiedlichen Artikeln miteinander verglichen. Das Programm erzeugt die folgende Ausgabe:

```
Apfel ist teurer als Birne
Birne ist billiger als Kiwi

Apfel ist mindestens so teuer wie Kiwi
Birne ist hoechstens so teuer wie Kiwi

Apfel und Kiwi haben denselben Preis
Birne und Kiwi haben unterschiedliche Preise
```

Bei den hier gewählten Preisen werden alle Bedingungen erfüllt und es erfolgen alle Ausgaben. Ändern Sie einmal einzelne Preise ab, und überlegen Sie, welche Ausgaben danach erfolgen sollen und welche nicht. Starten Sie erst anschließend zur Bestätigung das Programm.

= und ==

Hinweis

Achten Sie auf den Unterschied zwischen einer Zuweisung mit `=` und einem Vergleich mit `==`. Der Compiler übersetzt auch die Anweisung `if(preis = 1.5)` ohne Fehlermeldung. Allerdings haben Sie damit nicht geprüft, ob der Preis 1,50 Euro beträgt, sondern den Preis mit 1,50 Euro festgelegt. Dies führt nicht zu den erwarteten Ergebnissen.

4.3 Zwei Möglichkeiten, mit »if« und »else«

Nicht wahr

Sie können eine Verzweigung durch das Schlüsselwort `else` erweitern. Ist die Bedingung nach dem `if` *nicht wahr*, wird die Anweisung oder der Block von Anweisungen ausgeführt, der dem `else` folgt.

Ein Beispiel:

```
#include <stdio.h>
int main()
{
   double preis = 0.85;

   /* Falls Bedingung zutrifft */
   if(preis > 0.99)
      printf("Ein teurer Artikel\n");
```



```
   /* Ansonsten, falls also Bedingung nicht zutrifft */
   else
   {
      printf("Ein billiger Artikel\n");
      printf("Den nehmen wir\n");
   }

   return 0;
}
```

Listing 4.3 Datei »fall_else.c«

Das Programm erzeugt in jedem Fall eine Ausgabe. Hier ist es die folgende:

```
Ein billiger Artikel
Den nehmen wir
```

Hinweis

Achten Sie auch darauf, nach dem `else` kein Semikolon zu notieren. Dies würde diesen Zweig der Verzweigung unmittelbar beenden. Der Compiler meldet diesen Fehler ebenfalls nicht. Die anschließende Anweisung wäre nicht mehr mit der Verzweigung verbunden und würde stets ausgeführt.

4.4 Wie kann ich Bedingungen kombinieren?

Verknüpfung

Der Ablauf innerhalb einer Verzweigung kann auch von mehreren Bedingungen abhängig sein, die miteinander verknüpft werden. Die wichtigste Frage ist: Sollen alle Bedingungen *wahr* sein, oder reicht es aus, wenn eine der Bedingungen *wahr* ist?

Und &&

- Falls alle Bedingungen zutreffen sollen, verknüpfen Sie sie mithilfe des Operators `&&`. Er steht für das logische *Und*.

Oder ||

- Reicht dagegen eine Bedingung aus, nutzen Sie den Operator `||`. Er steht für das logische *Oder*.

Nicht !

Es gibt noch einen dritten logischen Operator: Das Zeichen `!` steht für das logische *Nicht*. Mit diesem Operator wird der Wahrheitswert einer Bedingung umgedreht: Aus *wahr* wird *falsch* und umgekehrt. Manche Verzweigung lässt sich mithilfe dieses Operators einfacher formulieren.

Ein Beispiel:

```
#include <stdio.h>
int main()
{
   double preisApfel = 1.45, preisBirne = 0.85, preisBanane = 0.75;

   /* Logisches Und */
   if(preisBirne < 1.0 && preisBanane < 1.0)
      printf("Beide Artikel sind billig\n");

   /* Logisches Oder */
   if(preisApfel > 0.99 || preisBirne > 0.99)
      printf("Mindestens einer der Artikel ist teuer\n");

   /* Logisches Nicht */
   if(!(preisBanane > 0.99))
      printf("Artikel ist nicht teuer\n");

   return 0;
}
```

Listing 4.4 Datei »fall_kombi.c«

In Listing 4.4 werden die Preise von drei unterschiedlichen Artikeln miteinander verglichen. Das Programm erzeugt die folgende Ausgabe:

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

Bei den hier gewählten Preisen werden alle Bedingungen erfüllt und es erfolgen alle Ausgaben. Ändern Sie einmal einzelne Preise ab, und überlegen Sie, welche Ausgaben danach erfolgen sollen und welche nicht. Starten Sie erst anschließend zur Bestätigung das Programm.

Vergleich vor Verknüpfung

Die Vergleichsoperatoren haben Vorrang vor den logischen Operatoren && und ||. Das bedeutet, dass zunächst der Wahrheitswert der einzelnen Bedingungen ermittelt wird. Erst anschließend werden die Bedingungen verknüpft.

Vorrangregeln

Allerdings hat der Operator ! für das logische *Nicht* Vorrang vor den Vergleichsoperatoren. Daher wird die Bedingung in Klammern gesetzt, bevor ihr Wahrheitswert umgekehrt wird. Sie finden in Abschnitt 12.1.9 eine Tabelle mit den Vorrangregeln für die Operatoren.

4.5 Eine Übungsaufgabe

Schreiben Sie ein Programm in der Datei *u_fall.c*. Es soll die folgende Ausgabe ermöglichen (hier mit Eingaben des Benutzers):

```
Anzahl: 5
Preis in Euro: 3.20
Summe der Rechnung: 12.80 Euro
```

Der Benutzer soll Anzahl und Preis für einen eingekauften Artikel eingeben. Anschließend wird die Summe der Rechnung ermittelt. Falls diese Summe 10 Euro überschreitet, soll ein Rabatt von 20 % gewährt werden; ansonsten soll kein Rabatt gewährt werden. Wählen Sie geeignete Variablennamen, und kommentieren Sie Ihr Programm ausreichend. Eine mögliche Lösung finden Sie in Abschnitt C.3.