Errata

Seite 100, Aufgabe A 4.2

Hier hat sich anstelle der korrekten Zahl "24" zweimal eine "22" hereingemogelt.

Korrekt muss es heißen: 40-16 = **24**, und **24** – 16 = 8.

Seite 131, Abbildung 6.5

Auch hier hat es eine Zahlenvertauschung in den Druck geschafft:

Richtig ist: 7 = 0*7 +**7**, und dementsprechend lautet die Zahl mit Siebenerbasis:

76510.

Seite 167, Kapitel 6.8, Listing 6.14

Ausgabe der Zeichenkette "Rindfleischetikettierungsüber (..) hat 63 Zeichen".

Hier könnten je nach verwendetem System einen anderen Wert erhalten, beispielsweise 65 oder auch 67.

Hier sollte als Test besser der folgende Text mit 65 Zeichen und ohne Umlaute verwendet werden:

Rindfleischetikettierungsueberwachungsaufgabenuebertragungsgesetz

Zeichen wie Umlaute, die nicht im ASCII-Zeichensatz enthalten sind, werden ja nach System und Compiler unterschiedlich behandelt. Insbesondere kann der Compiler solche Zeichen in der Zeichenkette mit mehr als einem einzelnen Zeichen codieren. Dann kann ein Zeichen mehreren `char` entsprechen und wird entsprechend gezählt, was dann zu den abweichenden Ergebnissen führt. Diese sogenannte Multi-Byte-Codierung soll im Buch allerdings nicht behandelt werden.

Seite 187, Abbildung 7.2

Die in der Abbildung angegebene Funktion `umkehren` erhält die Anzahl der umzukehrenden Elemente als `anz` übergeben, verwendet diese Variable aber nicht. Stattdessen wird in der Schleife immer der Wert konstante Wert 9 verwendet. In diesem speziellen Beispiel funktioniert das auch fehlerfrei, der Schleifenkopf sollte aber den übergebenen Wert verwenden und sieht dann so aus:

```
for (v = 0, h = anz – 1; v < h; v++, h--)
```

Seite 228, 236 und 238

In allen drei Fällen ist das Ergebnis der Suche nach Minimum und Maximum für das Maximum

Maximum: 31 anstatt von Maximum: 32

Seite 233, Listing 8.10

In den Schleifenkopf in der Funktion `zaehle` hat sich ein Dereferenzierungsoperator eingeschlichen, der dort nicht hingehört. Statt

for(i = 0, a = 0; *string[i] != 0; i++)

muss es lauten:

for (i = 0, a = 0; string[i] != 0; i++)

Seite 390, Abbildung 13.45

Die beiden Elemente in der Abbildung sind vertauscht. Die Sortierung arbeitet im dritten Schritt auf die drittletzte Stelle und im vierten Schritt auf die viertletze Stelle. In der Abbildung ist links der vierte Schritt der Sortierung und rechts der dritte Schritt dargestellt.

Seite 490, Tabelle 15.2

In der letzten Zeile der Tabelle in der zweiten Spalte muss der Wert wie in der ersten Spalte 1000000 lauten, dem abgedruckten Wert 100000 fehlt eine Null.

Seite 501, Tabelle 16.2

Die letzten vier Zeilen der Tabelle sind eine Wiederholung der ersten vier Zeilen und können komplett entfallen.

Seite 521, Abbildung 17.14

Auch die Verbindung von Dresden nach Leipzig müsste einen Verbindungspfeil, haben, der von Dresden nach Leipzig zeigen würde.

Seite 757, 20.9.1 Menge

Das angegeben Beispiel zur Differenzmenge enthält ein Element zu viel. Bei den Mengen A = { 1, 2, 4 } und B = { 1, 4 , 6, 7 } besteht die Menge C, zu der alle Elemente die zu A aber nicht zu B gehören nur aus der 2, es gilt damit C = A – B = { 2 }

Seite 764, Abbildung 20.9

Hier taucht die falsche Differenzmenge noch einmal auf. Auch hier gilt wieder A -= B // { 2 }. Die interne Darstellung ist damit statt

01100000
00100000

Seite 807, Kapitel 22.1.2

Als Überschrift für das Kapitel und Bildunterschrift für die Abbildung 22.3 wäre „Vererbung über mehrere Generationen“ treffender, das der Begriff „Wiederholte Vererbung“ einen Aspekt der Mehrfachvererbung beschreibt.

Seite 957, Listing 24.4

In dem Listing findet sich eine falsche Signatur für die Methode `push`. In Deklaration muss es lauten

int push(float element);

in der Definition

int floatStack::push(float element)

Seite 968, Listing 24.11

Die schließende geschweifte Klammer der Funktion `main` fehlt im Listing.

Seite 968-972

Im C++-Standard ist für die Klasse `std::exception` kein Konstruktor vorgesehen, der eine Zeichenkette übernimmt. Allerdings gibt es Compiler, die eine solche Implementierung bereitstellen und ein solcher kam auch bei der Erstellung des Beispiels zum Einsatz.

Für die abgeleitete Klassen `std::runtime_error` oder `std::logic_error` ist ein solcher Konstruktor auch im Standard vorgesehen.

In der Funktion `test3` muss daher entsprechend geändert werden, beispielsweise auf:

```
void test3( int  i )
   {
   switch( i % 3 ){
      case 0:
         return;
      case 1:
         cout << „AUSNAHME1 wird geworfen\n“;
         throw runtime_error( „AUSNAHME1“ );
      case 2:
         cout << „AUSNAHME2 wird geworfen\n“;
        throw logic_error( „AUSNAHME2“ );
      }
   }
```

Die Implementierung der Klasse `std::exception` gemäß des Standards gibt mit der Methode `what` auch lediglich ihren Typ aus, da kein entsprechender Ausgabetext vorgesehen ist. Damit ändert sich dann demnach dann auch die Ausgabe des Programms, beispielsweise, wenn eine geworfene, abgeleitete Ausnahme als ihre Basisklasse gefangen und behandelt wird.