# A

# Anhang

Im Anhang finden Sie die folgenden Themen:

- Empfehlungen zur Programmierung
- C++-Schlüsselwörter
- Die ASCII-Tabelle
- Rangfolge der Operatoren
- Compilerbefehle
- Lösungen zu den Aufgaben
- Hinweise zur Installation der Software von der DVD

## A.1 Programmierhinweise

Im Text sind Tips und Hinweise zur Programmierung vorhanden, von denen einige hier in zusammengefasster Darstellung erscheinen.

1. **Programme werden für Menschen geschrieben!**
   Nur lesbare und verständliche Programme sind wartbar. Ein Programm wird nur einmal geschrieben, aber mehrfach gelesen. Schwer verständliche Programme bergen überdies die Gefahr einer erhöhten Fehlerwahrscheinlichkeit. Die Bedeutung von Kommentaren und der Strukturierung des Programmcodes sollte man nicht unterschätzen! Außerdem sollte es bei etwas größeren Programmen getrennt vom Code eine problembezogene (objektorientierte Analyse) und eine programmbezogene (objektorientierter Entwurf) Dokumentation geben.

Das Einhalten von Programmierrichtlinien unterstützt das Schreiben gut lesbarer Programme. Es gibt sehr einige dieser Richtlinien, die sich in großen Teilen ähneln. Deshalb sei hier nur auf die wohl am besten bekannten »Ellemtel«-Regeln [HeNy], die JSF AV C++ Coding Standards [JSF] und auf den CERT C++ Secure Coding Standard [CERT] hingewiesen. Ein einfaches Beispiel für solche Regeln sind Vorschriften für die Schreibweise, etwa:

- Die Namen von eigenen Klassen sollen stets mit einem Großbuchstaben beginnen (im Gegensatz zu denen der C++-Standardbibliothek).
- Die Namen von Variablen und Funktionen beginnen mit einem Kleinbuchstaben.
- Konstantennamen sind vollständig groß zu schreiben, z.B. `FAKTOR`.
- Worttrennungen sind durch Wechsel in der Groß-/Kleinschreibung oder durch einen Unterstrich zu kennzeichnen, z.B. `anzahlDerObjekte` oder `anzahl_der_objekte`.
- Der Name einer Header-Datei soll dem Namen der Klasse entsprechen, die in dieser Datei deklariert wird.

Weitere Empfehlungen sind die nachfolgend aufgezählten Punkte, auch finden Sie in diesem Buch viele weitere Hinweise an den thematisch entsprechenden Stellen.

2. **Trennung von Schnittstellen und Implementation**
   Die Trennung von Schnittstellen und Implementation ist ein wichtiges Mittel, um Software wartbar und wieder verwendbar zu gestalten. Üblich sind
   - die Trennung von Funktionsprototyp und Funktionsdefinition sowie
   - die Definition einer gemeinsamen Schnittstelle für Klassen mit Hilfe einer abstrakten Klasse.
3. **Konstruktion von Schnittstellen**
   Empfehlungen zur Konstruktion von Schnittstellen sind in Abschnitt 20.1 auf Seite 557 ff. zusammengefasst. Falls nicht ausgeschlossen ist, dass von einer Klasse geerbt wird, müssen alle Methoden, die dabei überschrieben werden könnten, `virtual` sein. Siehe dazu auch Punkt 12 unten.
4. **Datenkapselung**
   Der Zugriff auf die Daten von Objekten sollte restriktiv gehandhabt werden. Die Erleichterung des Zugriffs mit `friend` oder `public`-Datenbereichen muss begründet sein. Verzichten Sie nach Möglichkeit auf globale Daten und Funktionen.
5. **One Definition Rule**
   Jede Variable, Funktion, Struktur, Konstante und so weiter in einem Programm hat *genau eine* Definition.
6. **Zeiger in Klassen**
   Zeiger in einer Klasse, die auf dynamisch erzeugte Objekte verweisen, erfordern für die Klasse in der Regel je einen besonderen Kopierkonstruktor, Zuweisungsoperator und Destruktor.
7. **Kopierkonstruktor, Zuweisungsoperator, Destruktor**
   Wenn einer der drei für eine Klasse `X` geschrieben werden muss, sind meistens auch die anderen beiden notwendig.
   - Der Kopierkonstruktor soll das zu kopierende Objekt nicht verändern und es daher als konstante Referenz übergeben:

```
X::X(const X&);                    // Kopierkonstruktor
```

- Der Zuweisungsoperator soll `*this` als Referenz (`X&`) zurückgeben, damit die Verkettung von Zuweisungen möglich ist.
- Der Zuweisungsoperator soll bei dynamischen, also mit `new` erzeugten Teilen des Objekts, die folgende Struktur aufweisen, wenn die linke und die rechte Seite der Zuweisung vom selben statischen Typ sind:

```
X& X::operator=(X obj) { // Übergabe per Wert! (vgl. Seite 329)
    swap(obj);           // wirft keine Exception
    return *this;
}  // Aufruf des Destruktors von obj
```

  Die Funktion `X::swap(X&)` muss natürlich existieren. Sie vertauscht die Attribute von `*this` mit denen von `obj`. Dabei kann vorteilhaft die Funktion `swap()` der Standardbibliothek eingesetzt werden, zum Beispiel

```
void X::swap(X& obj) {
  std::swap(attribut1, obj.attribut1);
  std::swap(attribut2, obj.attribut2);
  // usw.
}
```

  Diese Form ist exception-sicher. Sie ermöglicht dem Compiler, bei einem temporären Argument auf der rechten Seite der Zuweisung, den Kopierkonstruktor zu umgehen.
- Ein nicht nach obigem Muster geschriebener Zuweisungsoperator kann eine Prüfung der Zuweisung des Objekts auf sich selbst enthalten, nicht unnötige oder gefährliche Anweisungen (zum Beispiel `delete`) auszuführen:

```
X& X::operator=(const X& obj) {
    if(this != &obj) {
      //... Anweisungen (Ausführung nur bei Nicht-Identität)
    }
    return *this;
}
```

  In der Praxis wird das wohl kaum vorkommen. Das Weglassen dieser Prüfung ist unschädlich, wenn der Zuweisungsoperator exception-sicher ist.

8. **Referenzen oder Zeiger?**
   Alles, was mit Referenzen getan werden kann, ist im Prinzip auch mit Zeigern möglich. In manchen Fällen sind Referenzen jedoch vorzuziehen.
   Referenzen sind bei der Übergabe in und aus Funktionen sinnvoll, weil sie innerhalb der Funktion syntaktisch wie ein Objektname verwendet werden können. Der Compiler löst die Referenz auf, während beim Zeiger stets vom Programmierer dereferenziert werden muss. Eine Referenz bezieht sich immer auf ein existierendes Objekt, sie kann nie NULL sein.
9. **Wann wird `delete [ ]` benötigt?**
   `delete [ ]` ist genau dann erforderlich, wenn das zu löschende Objekt mit `new [ ]` erzeugt wurde. Der Compiler weiß (leider) nicht, ob ein Objekt mit `new [ ]` erzeugt wurde, und prüft daher auch nicht, ob es mit `delete [ ]` freigegeben wird.

10. **Speicherbeschaffung und -freigabe kapseln**
    Die Operatoren `new` und `delete` sind stets paarweise zu verwenden. Um Speicherfehler zu vermeiden, empfiehlt sich das »Verpacken« dieser Operationen in Konstruktor und Destruktor wie bei der Beispielklasse `MeinString` (Seite 233) oder bei der Verwendung der »Smart Pointer« von Seite 339. Ein weiterer Vorteil ist die korrekte Speicherfreigabe bei Exceptions (siehe Seite 567).
11. **Wird ein virtueller Destruktor benötigt?**
    Das Vorhandensein virtueller Funktionen ist ein Indiz für die Notwendigkeit eines virtuellen Destruktors. Er wird genau dann benötigt, wenn `delete` auf einen Basisklassenzeiger angewendet wird, der auf ein dynamisch erzeugtes Objekt einer abgeleiteten Klasse verweist. Virtuelle Destruktoren sollten immer dann verwendet werden, wenn von der betreffenden Klasse abgeleitet wird oder nicht auszuschließen ist, dass von ihr zukünftig durch Ableitung neue Klassen gebildet werden.
12. **Nur virtuelle Funktionen überschreiben!**
    Nicht-virtuelle Funktionen einer Basisklasse sollten *nicht* in abgeleiteten Klassen überschrieben werden. Der Grund liegt darin, dass das Verhalten eines Programms sich nicht ändern sollte, wenn auf eine Methode über den Objektnamen oder über Basisklassenzeiger bzw. -referenzen zugegriffen wird.
13. **Initialisierung von Objekten**
    Objekte sollten aus Effizienzgründen über Initialisierungslisten anstatt mit Zuweisungen im Codeblock des Konstruktors initialisiert werden. Die Initialisierung von Objektkonstanten ist ohnehin nur über eine Liste möglich.
14. **Konstanz von Objekten**
    Nutzen Sie die Prüfungsmöglichkeiten des Compilers! Alle Modifikationsversuche unveränderlicher Objekte werden schon vom Compiler zurückgewiesen, wenn sie als `const` deklariert sind.
    Ein (konstantes oder veränderliches) Objekt einer Klasse `X`, das durch einen Funktionsaufruf *nicht* verändert werden soll, ist an eine Funktion per Wert (`int func(X Obj)`), per konstanter Referenz (`int func(const X& Obj)`) oder per Zeiger auf ein konstantes Objekt (`int func(const X* ZeigerAufObjekt)`) zu übergeben. Bei größeren Objekten empfiehlt sich eine der beiden letzten Möglichkeiten.
15. **Makros**
    Verwenden Sie nur wirklich notwendige Makros. Meistens gibt es eine alternative Lösung in C++.
16. **inline**
    Funktionen sollten nur dann `inline` deklariert werden, wenn sie sehr kurz sind und/oder die Laufzeit deutlich verbessert wird.

# A.2 C++-Schlüsselwörter

Die Bezeichner in Tabelle A.1 sind reserviert für den Gebrauch als Schlüsselwort und sollen nicht anderweitig benutzt werden. In der Tabelle sind Symbole, die als Ersatz für bestimmte Zeichen gelten können, *nicht* enthalten (Beispiele: `and`, `or`, `not_eq`, ...);

**Tabelle A.1:** C++-Schlüsselwörter

| | | | | | |
|---|---|---|---|---|---|
| alignas | const_cast | extern | noexcept | static_assert | union |
| alignof | constexpr | false | nullptr | static_cast | unsigned |
| asm | continue | float | operator | struct | using |
| auto | decltype | for | private | switch | virtual |
| bool | default | friend | protected | template | void |
| break | delete | goto | public | this | volatile |
| case | do | if | register | thread_local | wchar_t |
| catch | double | inline | reinterpret_cast | throw | while |
| char | dynamic_cast | int | return | true | |
| char16_t | else | long | short | try | |
| char32_t | enum | mutable | signed | typedef | |
| class | explicit | namespace | sizeof | typeid | |
| const | export | new | static | typename | |

Darüber hinaus gibt es die reservierten Bezeichner `final` und `override`.

# A.3 ASCII-Tabelle

ASCII ist die Abkürzung für *American Standard Code for Information Interchange*. Es gibt auch einen ISO-Code (ISO = *International Standards Organization*), der teilweise nationale Symbole erlaubt. ASCII ist jedoch weiter verbreitet. Er ist ein 7-Bit-Code und besteht aus 128 Zeichen, die in nichtdruckbare und druckbare Zeichen unterteilt werden. Die ersteren werden Steuerzeichen (englisch *control characters*) genannt. Die Zeichen sind in den folgenden Tabellen A.2 und A.3 dargestellt.

Der Piepton »bell« der ersten Tabelle könnte natürlich als `\x07` anstatt als `\a` geschrieben werden, dasselbe gilt entsprechend für `\0`, `\t`, `\v` und `\r`. Anstelle der Hex-Darstellung `\x..` ist auch die oktale Darstellung möglich (siehe Seite 44). Die Zeichen mit den Nummern 34, 39 und 92 haben eine besondere Bedeutung in C++, weswegen sie in einem Programm durch einen vorangestellten Backslash (\) gekennzeichnet werden müssen, wenn nur das Zeichen selbst gemeint ist.

**Tabelle A.2:** ASCII-Steuerzeichen

| Nr. | hex | Abkürzung | Name | C++ |
|---|---|---|---|---|
| 0 | 0x00 | NUL | null | `\0` |
| 1 | 0x01 | SOH | start of heading | `\x01` |
| 2 | 0x02 | STX | start of text | `\x02` |
| 3 | 0x03 | ETX | end of text | `\x03` |
| 4 | 0x04 | EOT | end of transmission | `\x04` |
| 5 | 0x05 | ENQ | enquiry | `\x05` |
| 6 | 0x06 | ACK | acknowledge | `\x06` |
| 7 | 0x07 | BEL | alert | `\a` |
| 8 | 0x08 | BS | backspace | `\b` |
| 9 | 0x09 | HT | horizontal tab | `\t` |
| 10 | 0x0A | NL/LF | new-line | `\n` |
| 11 | 0x0B | VT | vertical tab | `\v` |
| 12 | 0x0C | FF | form feed | `\f` |
| 13 | 0x0D | CR | carriage return | `\r` |
| 14 | 0x0E | SO | shift out | `\x0E` |
| 15 | 0x0F | SI | shift in | `\x0F` |
| 16 | 0x10 | DLE | data link escape | `\x10` |
| 17 | 0x11 | DC1 | device control 1 | `\x11` |
| 18 | 0x12 | DC2 | device control 2 | `\x12` |
| 19 | 0x13 | DC3 | device control 3 | `\x13` |
| 20 | 0x14 | DC4 | device control 4 | `\x14` |
| 21 | 0x15 | NAK | negative acknowledge | `\x15` |
| 22 | 0x16 | SYN | synchronous idle | `\x16` |
| 23 | 0x17 | ETB | end of transmission block | `\x17` |
| 24 | 0x18 | CAN | cancel | `\x18` |
| 25 | 0x19 | EM | end of medium | `\x19` |
| 26 | 0x1A | SUB | substitute | `\x1A` |
| 27 | 0x1B | ESC | escape | `\x1B` |
| 28 | 0x1C | FS | file separator | `\x1C` |
| 29 | 0x1D | GS | group separator | `\x1D` |
| 30 | 0x1E | RS | record separator | `\x1E` |
| 31 | 0x1F | US | unit separator | `\x1F` |
| 127 | 0x7F | DEL | delete | `\x7F` |

**Tabelle A.3:** Druckbare ASCII-Zeichen (Spalte Z. = Zeichen)

| Nr. | hex | Z. | C++ | Nr. | hex | Z. | C++ | Nr. | hex | Z. | C++ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 32 | 0x20 | | | 64 | 0x40 | @ | @ | 96 | 0x60 | ' | ' |
| 33 | 0x21 | ! | ! | 65 | 0x41 | A | A | 97 | 0x61 | a | a |
| 34 | 0x22 | " | \" | 66 | 0x42 | B | B | 98 | 0x62 | b | b |
| 35 | 0x23 | # | # | 67 | 0x43 | C | C | 99 | 0x63 | c | c |
| 36 | 0x24 | $ | $ | 68 | 0x44 | D | D | 100 | 0x64 | d | d |
| 37 | 0x25 | % | % | 69 | 0x45 | E | E | 101 | 0x65 | e | e |
| 38 | 0x26 | & | & | 70 | 0x46 | F | F | 102 | 0x66 | f | f |
| 39 | 0x27 | ' | \' | 71 | 0x47 | G | G | 103 | 0x67 | g | g |
| 40 | 0x28 | ( | ( | 72 | 0x48 | H | H | 104 | 0x68 | h | h |
| 41 | 0x29 | ) | ) | 73 | 0x49 | I | I | 105 | 0x69 | i | i |
| 42 | 0x2A | * | * | 74 | 0x4A | J | J | 106 | 0x6A | j | j |
| 43 | 0x2B | + | + | 75 | 0x4B | K | K | 107 | 0x6B | k | k |
| 44 | 0x2C | , | , | 76 | 0x4C | L | L | 108 | 0x6C | l | l |
| 45 | 0x2D | - | - | 77 | 0x4D | M | M | 109 | 0x6D | m | m |
| 46 | 0x2E | . | . | 78 | 0x4E | N | N | 110 | 0x6E | n | n |
| 47 | 0x2F | / | / | 79 | 0x4F | O | O | 111 | 0x6F | o | o |
| 48 | 0x30 | 0 | 0 | 80 | 0x50 | P | P | 112 | 0x70 | p | p |
| 49 | 0x31 | 1 | 1 | 81 | 0x51 | Q | Q | 113 | 0x71 | q | q |
| 50 | 0x32 | 2 | 2 | 82 | 0x52 | R | R | 114 | 0x72 | r | r |
| 51 | 0x33 | 3 | 3 | 83 | 0x53 | S | S | 115 | 0x73 | s | s |
| 52 | 0x34 | 4 | 4 | 84 | 0x54 | T | T | 116 | 0x74 | t | t |
| 53 | 0x35 | 5 | 5 | 85 | 0x55 | U | U | 117 | 0x75 | u | u |
| 54 | 0x36 | 6 | 6 | 86 | 0x56 | V | V | 118 | 0x76 | v | v |
| 55 | 0x37 | 7 | 7 | 87 | 0x57 | W | W | 119 | 0x77 | w | w |
| 56 | 0x38 | 8 | 8 | 88 | 0x58 | X | X | 120 | 0x78 | x | x |
| 57 | 0x39 | 9 | 9 | 89 | 0x59 | Y | Y | 121 | 0x79 | y | y |
| 58 | 0x3A | : | : | 90 | 0x5A | Z | Z | 122 | 0x7A | z | z |
| 59 | 0x3B | ; | ; | 91 | 0x5B | [ | [ | 123 | 0x7B | { | { |
| 60 | 0x3C | < | < | 92 | 0x5C | \ | \\ | 124 | 0x7C | \| | \| |
| 61 | 0x3D | = | = | 93 | 0x5D | ] | ] | 125 | 0x7D | } | } |
| 62 | 0x3E | > | > | 94 | 0x5E | ^ | ^ | 126 | 0x7E | ~ | ~ |
| 63 | 0x3F | ? | ? | 95 | 0x5F | _ | _ | | | | |

# A.4 Rangfolge der Operatoren

Die Rangfolge der Operatoren ist im C++-Standard [ISOC++] nicht direkt spezifiziert. Sie kann aber durch die Syntax der Programmiersprache abgeleitet werden, etwa wie es hier im Buch auf Seite 116 gemacht wird, wo durch die Syntax die Regel »Punktrechnung vor Strichrechnung« gewährleistet wird. In der Tabelle A.4 bedeuten kleine Zahlen große Prioritäten.

**Tabelle A.4:** Präzedenz von Operatoren

| Rang | Operatoren |
|---|---|
| 0 | `::` |
| 1 | `.` `->` `[ ]` `f()` (Funktionsaufruf)<br>`Typ()` (Typumwandlung im funktionalen Stil)<br>`++` `--` (postfix) `typeid()` `dynamic_cast<>()`<br>`static_cast<>()` `reinterpret_cast<>()` `const_cast<>()` |
| 2 | `sizeof` `++` `--` (präfix) `~` `!`<br>`+` `-` (unär) `&` (Adressoperator) `*` (Dereferenzierung)<br>`new` `new[ ]` `delete` `delete[ ]`<br>`(Typ) Ausdruck` (C-Stil-Typumwandlung) |
| 3 | `.*` `->*` |
| 4 | `*` `/` `%` |
| 5 | `+` `-` (binär) |
| 6 | `<<` `>>` |
| 7 | `<` `>` `<=` `>=` |
| 8 | `==` `!=` |
| 9 | `&` (bitweises UND) |
| 10 | `^` (bitweises exklusiv-ODER) |
| 11 | `\|` (bitweises ODER) |
| 12 | `&&` (logisches UND) |
| 13 | `\|\|` (logisches ODER) |
| 14 | `?:` (Bedingungsoperator) |
| 15 | alle Zuweisungsoperatoren `=`, `+=`, `<<=` usw. |
| 16 | `throw` |
| 17 | `,` |

Auf gleicher Prioritätsstufe wird ein Ausdruck von links nach rechts abgearbeitet mit Ausnahme der Ränge 2, 14 und 15, die von rechts abgearbeitet werden. Wegen der leichten Konvertierbarkeit zwischen `char`, `int` und `bool` werden mögliche Fehler nicht durch den Compiler entdeckt. Beispiele für mögliche Missverständnisse (teilweise aus [vdL]):

### Operatorenrangfolge: Mögliche Missverständnisse

**Tabelle A.5:** Operatorenrangfolge: Mögliche Missverständnisse

| Ausdruck oder Anweisung | vermutlich erwartetes Ergebnis | tatsächliches Ergebnis |
|---|---|---|
| `i = 1,2;` | `i` wird 2 | `i` wird 1, die 2 wird verworfen |
| `a[2,3];` | `a[2][3]` | `a[3]` |
| `if(a =! 2)` | `if(a != 2)` | `if(a = (!2))`, d.h. `if(false)` |
| `x = msb<<4 + lsb` | `x = (msb<<4) + lsb` | `x = msb << (4+lsb)` |
| `c = getchar() != EOF` | `(c = getchar()) != EOF` | `c= (getchar() != EOF)` |
| `val&mask != 0` | `(val&mask) != 0` | `val & (mask != 0)` |
| `a < b < c` | `a < b && b < c` | `(a < b) < c` (Vergleich `bool` mit `int`!) |
| `cout << a<<2` | `cout << (a<<2)` | `(cout << a) << 2` |
| `int *fp()` | `int (*fp)()` Deklaration eines Funktionszeigers | Deklaration einer Funktion, die einen Zeiger auf `int` zurückgibt |

# A.5 Compilerbefehle

Hier finden Sie die wichtigsten Befehle für den GNU C++-Compiler, die auch von einigen anderen Compilern verstanden werden. In der Windows-Welt ist die Endung `.exe` für ausführbare Dateien vorgesehen, in der Unix-Welt ist der Name frei wählbar, zum Beispiel könnte die Datei einfach *summe* heißen. Wenn der Name nicht vordefiniert wird, heißt die ausführbare Datei `a.out`.

| | |
|---|---|
| `g++ --help` | die wichtigsten Optionen anzeigen |
| `g++ --version` | Compiler-Version anzeigen |
| `g++ -c summe.cpp` | nur Compilieren (*summe.o* wird erzeugt) |
| `g++ -o summe summe.o` | Linken |
| `g++ summe.cpp` | Compilieren und Linken |

Mehrere Dateien:

| | |
|---|---|
| `g++ a1.cpp main.cpp` | Compilieren und Linken |

oder einzeln

| | |
|---|---|
| `g++ -c a1.cpp` | Compilieren |
| `g++ -c main.cpp` | Compilieren |
| `g++ a1.o main.o` | Linken, Ergebnis *a.out* |
| `g++ -o main.exe a1.o main.o` | Linken, Ergebnis *main.exe* |

Überall kann die Option `-Wall` dazugenommen werden. `W` steht für »Warnung«, `all` für »alle«. Diese Option ist empfehlenswert, weil der Compiler nicht nur Fehler, sondern auch Warnungen ausgibt, die auf syntaktisch richtigen, aber vermutlich falschen Programmcode deuten.

Eigene *include*-Verzeichnisse werden mit der Option `-I` voreingestellt. Einzelheiten sind auf Seite 128 zu finden.

Eine für ein spezielles Gebiet vorübersetzte und gepackte Bibliothek (englisch *library*) hat in der Regel einen Namen, der mit *lib* anfängt und mit *.a* aufhört. So kann eine Bibliothek zur Komprimierung von Bilddaten *libjpeg.a* heißen. Solche Bibliotheken werden mit der Option `-l` eingebunden, wobei *lib* und *.a* weggelassen werden. Beispiel:

```
g++ -o main.e a1.o main.o -ljpeg
```

Die Option `-g` fügt dem Ergebnis Informationen für den Debugger `gdb` zu. Der Debugger ist ein mächtiges Werkzeug zum Aufspüren von Fehlern, wenn alles Nachdenken versagt hat. Weitere Informationen zum Compiler oder zum Debugger erhalten Sie auf Ihrem Unix-System durch Eingabe von `info g++` oder `info gdb` bzw. `man g++` und `man gdb`.



## A.6 Lösungen zu den Übungsaufgaben

Das Verzeichnis *cppbuch* der Beispiele enthält nicht nur die Beispielprogramme, sondern auch die Lösungen, und zwar im Verzeichnis *cppbuch/loesungen*. Die Kapitelnummer bestimmt den Verzeichnisnamen, die laufenden Nummer der Aufgabe den entsprechenden Dateinamen. So ist die Datei *4.cpp* im Unterverzeichnis *k1* (= Kapitel 1) die Lösung zu Aufgabe 1.4. Manchmal gehören zu einer Lösung mehrere Dateien. Diese befinden sich in einem entsprechend gekennzeichneten Unterverzeichnis. Zum Beispiel enthält ein Unterverzeichnis *7* die Dateien zur Lösung von Aufgabe 7.

Die Lösungen sind nur als Vorschlag aufzufassen. Oft gibt es mehrere Lösungen, auch wenn nur eine angegeben ist. Einige wenige Programme zu den Lösungen wurden aus Platzgründen nicht abgedruckt, sind aber im Verzeichnis *cppbuch/loesungen* enthalten.

### Kapitel 1

**1.1** `log(x)` ist für $x \leq 0$ nicht definiert, `sqrt(x)` ist für ein negatives x nicht definiert. Die möglichen Ausgaben *inf* bzw. *nan* stehen für »infinity« (unendlich) bzw. »not a number« (keine gültige Zahl).

**1.2**

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

int main() {
   unsigned int ui = 0;
   unsigned long int uli = 0;
   cout << "max. unsigned int    = "<< ~ui      << endl;
   cout << "max. unsigned long int= "<< ~uli    << endl;
   cout << "max. int             = "<< (~ui>>1) << endl;
   cout << "max. long int        = "<< (~uli>>1) << endl;
}
```

1.3
```
#include<iostream>
using namespace std;

int main() {
   int anfang;
   int ende;
   cout << "Nur ganze Zahlen eingeben:" << endl
        << "Bereichsanfang:";
   cin >> anfang;
   cout << "Bereichsende:";
   cin >> ende;
   if(anfang > ende) {
      cout << "Der Bereichsanfang darf nicht nach dem Bereichsende"
           " liegen!" << endl;
   }
   else {
      cout << "Zahl:";
      int zahl;
      cin >> zahl;
      if(zahl >= anfang && zahl <= ende) {
         cout << zahl << " liegt im Bereich " << anfang
              << ".." << ende << endl;
      }
      else {
         cout << zahl << " liegt nicht im Bereich " << anfang
              << ".." << ende << endl;
      }
   }
}
```

1.4
```
#include<iostream>
using namespace std;

int main() {
   cout << "Maximum dreier Zahlen! Eingabe: ";
   int a, b, c;
   cin >> a >> b >> c;
   cout <<" Maximum = ";
   if(a > b) {
      if(a > c) {
         cout << a;
      }
      else {
         cout << c;
      }
   }
   else {
      if(b > c) {
         cout << b;
      }
      else {
         cout << c;
      }
```

```
   }
   cout << endl;
}
```

**1.5**

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

int main() {
   cout << "Eingabe einer Zahl: ";
   int zahl = 0;
   cin >> zahl;
   int anzahlDerBytes = sizeof zahl;
   int anzahlDerBits = 8 * anzahlDerBytes;
   cout <<" binär :  ";
   for(int k = anzahlDerBits-1; k >= 0 ; --k) {
      if(zahl & (1 << k)) {
         cout << "1";
      }
      else {
         cout << "0";
      }
   }
   cout << endl;
}
```

*Bemerkung:* Die `1` in der `if(...)`-Bedingung ist vom Typ `int`. Sie muss durch mindestens so viele Bits wie `zahl` repräsentiert werden. Wenn `zahl` als `long` deklariert werden soll, ist daher `1L` zu schreiben.

**1.6** a) Unendliche Schleife, falls der Startwert `i > 0` ist, weil `i` nicht verändert wird.

b) Unendliche Schleife, falls `i<0` oder `i` ungerade ist. Auch in allen anderen Fällen ist die Schleife nicht besonders sinnvoll, da das Ergebnis stets `i == 0` ist.

c) Die Schleife terminiert nur, falls zu Beginn `i < 0` (bei beliebigem `n`) ist.

**1.7** a) Es wird die Summe der Zahlen 2...101 gebildet. Abhilfe: Anweisungen im Block vertauschen.

b) Die geschweiften Klammern fehlen. Das Ergebnis ist 101.

c) Korrekte Lösung. Ohne Schleife geht es auch!

d) `sum` wird innerhalb der Schleife stets auf 0 gesetzt.

e) Durch die vorangestellte 0 ist 0100 eine *Oktalzahl* mit dem Dezimalwert 64 (siehe auch Seite 44).

**1.8**

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

int main() {
   int n1, n2;
   bool ungueltig;
   do {
      cout << "Natürliche Zahlen n1 und n2 eingeben (n1 <= n2):";
```

```
    cin >> n1 >> n2;
    ungueltig = n1 < 0 || n2 < 0 || n1 > n2;
    if(ungueltig) {
       cout << "Eingabefehler!" << endl;
    }
  } while(ungueltig);

  // Berechne die Summe beider Zahlen
  int summe = 0;
  cout << "a) Summe mit for-Schleife berechnet: ";
  for(int i = n1; i <= n2; ++i) {
     summe += i;
  }
  cout << summe << endl;
  cout << "b) Summe mit while-Schleife berechnet: ";
  summe = 0;
  int i = n1;
  while(i <= n2) {
     summe += i++;
  }
  cout << summe << endl;
  cout << "c) Summe mit do while-Schleife berechnet: ";
  summe = 0;
  i = n1;
  do {
     summe += i++;
  } while(i <= n2);
  cout << summe << endl;

  cout << "d) Summe ohne Schleife berechnet: ";
  summe = n2*(n2+1)/2 - (n1-1)*n1/2 ;
  cout << summe << endl;
  return 0;
}
```

**1.9**

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

int main() {
   char c;
   bool zuEnde = false;
   while(!zuEnde) {
      cout << "Wählen Sie: a, b, x = Ende : ";
      cin >> c;

      switch(c) {
         case 'a': cout << "Programm a\n"; break;
         case 'b': cout << "Programm b\n"; break;
         case 'x': zuEnde = true;        break;
         default : cout << "Falsche Eingabe! "
                        "Bitte wiederholen!\n";
      }
   }
```

```
    cout << "\n Programmende\n";
}
```

**1.10**

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

int main() {
   string str = "17462309"; // aus Aufgabentext
   long int z = 0;
   for(unsigned int i = 0; i < str.size(); ++i) {
      z *= 10;
      z += (int)str.at(i) - (int)'0';
   }
   cout << "z = " << z;
   int quersumme = 0;
   while(z > 0) {
      quersumme += z % 10;
      z /= 10;
   }
   cout << "  Quersumme = " << quersumme << endl;
   return 0;
}
```

**1.11**

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

int main() {
   cout << "Umwandlung einer natürlichen Dezimalzahl in "
      "eine römische Zahl.\n Dezimalzahl eingeben:";
   int dezimalzahl;
   cin >> dezimalzahl;
   // Position 0123456
   const string ZEICHENVORRAT("IVXLCDM");
   int zehner = 1000, n = 6; // Start mit M=1000 (Pos. 6)
   string ergebnis;

   while (dezimalzahl != 0) { // Ziffern sukzessive abtrennen
      int ziffer = dezimalzahl / zehner;
      if ((ziffer > 3 && zehner == 1000) // Tausender
          ||   ziffer <= 3) {           // oder 0,1,2,3
         for (int i=1; i<=ziffer; i++) {
            ergebnis += ZEICHENVORRAT.at(n);
         }
      }
      else if (ziffer <= 4) {           // 4
         ergebnis += ZEICHENVORRAT.at(n);
         ergebnis += ZEICHENVORRAT.at(n+1);
      }
      else if (ziffer <= 8) {           // 5,6,7,8
         ergebnis += ZEICHENVORRAT.at(n+1);
         for (int i=1; i<=ziffer-5; i++) {
            ergebnis += ZEICHENVORRAT.at(n);
```

```
        }
      }
      else {
         ergebnis += ZEICHENVORRAT.at(n); // 9
         ergebnis += ZEICHENVORRAT.at(n+2);
      }
      n -= 2;
      dezimalzahl %= zehner;
      zehner /= 10;
   }
   cout << "Ergebnis: " << ergebnis << endl;
}
```

**1.12**

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

int main() {
   const int MINIMUM = -99;
   const int MAXIMUM = 100;
   const int INTERVALLZAHL = 10;
   const int INTERVALLBREITE = (MAXIMUM - MINIMUM + 1)/INTERVALLZAHL;
   int eingabe;
   vector<int> intervalle(INTERVALLZAHL);
   cout << "Bitte Zahlen im Bereich " << MINIMUM
        << " bis " << MAXIMUM << " eingeben:\n";
   cin >> eingabe;

   while(eingabe >= MINIMUM && eingabe <= MAXIMUM) {
      intervalle[(eingabe-MINIMUM) /INTERVALLBREITE]++;
      cin >> eingabe;
   }
   for(int i = 0; i < INTERVALLZAHL; i++) {
      cout << "Intervall "
           << i*INTERVALLBREITE + MINIMUM << " .. "
           << (i+1)*INTERVALLBREITE + MINIMUM -1 << ": "
           << intervalle[i] << endl;
   }
}
```

**1.13**

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

int main() {
   cout << "Bitte eine Startzahl > 0 eingeben: ";
   long long zahl;
   cin >> zahl;
   int iterationen = 0;
   long long maxzahl=0;
   while(zahl > 1) {
      ++iterationen;
      if(zahl % 2 == 0) {          // Zahl ist gerade
```

```
        zahl /= 2;
      }
      else {
        zahl = 3 * zahl + 1;
      }
      cout << zahl << endl;
      if(maxzahl < zahl) {
        maxzahl = zahl;
        cout << " neues Maximum. Weiter mit ENTER" << endl;
        string dummy;
        getline(cin, dummy); // weiter mit Tastendruck
      }
    }
    cout << iterationen << " Iterationen. Maximale Zahl ="
         << maxzahl << endl;
}
```

**1.14**

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

struct Person {           // Person-Typ anlegen
   string nachname;
   string vorname;
   int alter;
};

int main() {
   Person diePerson;      // Person-Objekt anlegen
   cout << "Nachnamen eingeben: ";
   cin >> diePerson.nachname;
   cout << "Vornamen eingeben: ";
   cin >> diePerson.vorname;
   cout << "Alter eingeben: ";
   cin >> diePerson.alter;
   cout << "Die Person hat folgende Daten:" << endl;
   cout << "Nachname : " << diePerson.nachname << endl;
   cout << "Vorname  : " << diePerson.vorname << endl;
   cout << "Alter    : " << diePerson.alter << endl;
}
```

## Kapitel 2

**2.1** Aus Platzgründen wird die Lösung nicht abgedruckt. Sie liegt im Verzeichnis *cppbuch/loesungen/k2* der Beispiele vor.

**2.2** Die Lösung ist in der Lösung zu Aufgabe 2.3 enthalten.

**2.3**

```
#include <iostream>
#include <cstdlib> // für exit()
#include<string>
#include <fstream>
```

```
using namespace std;

int main() {
   ifstream quelle;
   cout << "Dateiname :";
   string Quelldateiname;
   cin >> Quelldateiname;

   quelle.open(Quelldateiname.c_str(), ios::binary|ios::in);
   if (!quelle)  {// muss existieren
      cerr << Quelldateiname
           << " kann nicht geöffnet werden!\n";
      exit(-1);
   }
   unsigned char c;       // unsigned! (Bereich 0..255 statt -128..127)
   unsigned int count = 0, low, hi;
   // char ist notwendig, weil get(unsigned char) nicht implementiert ist (GNU C++).
   char cc;
   const int ZEILENLAENGE = 16;
   string buchstaben;
   string hexcodes;
   while (quelle.get(cc)) {
      c = cc;
      low = int(c) & 15;
      hi  = int(c) >> 4;
      // Umsetzung der Werte 0...15 auf ASCII-Zeichen (vgl. Tabelle Seite 887)
      if (low < 10) {
         low += 48;       // '0'...'9'
      }
      else {
         low += 55;       // 'A'...'F'
      }
      if (hi < 10) {
         hi  += 48;
      }
      else {
         hi  += 55;
      }
      hexcodes += char(hi);
      hexcodes += char(low);
      hexcodes += ' ';
      if(c < ' ') { // nicht druckbares Zeichen
         c = '.';
      }
      buchstaben += c;
      ++count;
      count %= ZEILENLAENGE;
      if (count == 0) {
         cout << buchstaben << " " << hexcodes << endl;
         buchstaben = "";
         hexcodes = "";
      }
   }
```

```
   if (count != 0) { // Rest ausgeben
      cout << buchstaben;
      for(size_t i=0; i < (ZEILENLAENGE-count); ++i) {
         cout << ' ';
      }
      cout << "  " << hexcodes << endl;
   }
   cout << endl;
}
```

**2.4**

```
// Datei-Statistik
#include<iostream>
#include<cstdlib> // für exit()

#include<fstream>
#include<string>
using namespace std;

int main() {
    ifstream quelle;
    cout << "Dateiname :";
    string Quelldateiname;
    cin >> Quelldateiname;
    quelle.open(Quelldateiname.c_str());
    if (!quelle) { // muss existieren
        cerr << Quelldateiname
             << " kann nicht geöffnet werden!\n";
        exit(-1);
    }
    char c;
    unsigned long zeichenzahl = 0, wortzahl = 0, zeilenzahl = 0;
    bool wort = false;
    while (quelle.get(c)) {
       if (c == '\n') {
          ++zeilenzahl;
       }
       else {
          ++zeichenzahl;
       }
       // Anpassung auf Umlaute fehlt noch!
       if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')){
          // Wortanfang, oder c ist in einem Wort
          wort = true;
       }
       else {
          if(wort) {
             ++wortzahl; // Wortende überschritten
          }
          wort = false;
       }
    }
    cout << "Anzahl der Zeichen (ohne Zeilenendekennung) = "
         << zeichenzahl << endl;
```

```
    cout << "Anzahl der Worte = " << wortzahl  << endl;
    cout << "Anzahl der Zeilen = " << zeilenzahl << endl;
}
```

Die Klammern um `(c >= 'A'&& c <= 'Z')` usw. sind nicht unbedingt notwendig; sie dienen der besseren Lesbarkeit.

## Kapitel 3

**3.1**

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

int dauerInSekunden(int stunden, int minuten, int sekunden);

int main() {
   int std = 3;
   int m  = 37;
   int sec = 40;
   cout << std << " Stunden und " << m << " Minuten und "
        << sec << " Sekunden sind insgesamt "
        << dauerInSekunden(std, m, sec) << " Sekunden."
        << endl;
}

int dauerInSekunden(int stunden, int minuten, int sekunden) {
   return 3600 * stunden + 60 * minuten + sekunden;
}
```

**3.2**

```
#include<iostream>
#include<cmath> // wegen pow(), s.u.
using namespace std;

double power(double x, int y);

int main() {
   cout << "x^y berechnen. Zahlen x und y eingeben (y ganzzahlig):";
   double x;
   int y;
   cin >> x >> y;
   cout << "x^y = " << power(x, y) << endl;
   cout << "pow(x,y) = " << pow(x, y) << endl; // aus <cmath>
}

// Die Funktion power() entspricht der Funktion pow() der C++-Bibliothek <cmath>.
double power(double x, int y) {
   double ergebnis = 1;
   bool negativ = false;
   if(y < 0) {
      y = -y;
      negativ = true;
   }
   for(int i=0; i < y; ++i) {
```

```
      ergebnis *= x;
   }
   if(negativ) {
      ergebnis = 1.0/ergebnis;
   }
   return ergebnis;
}
```

**3.3**

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

long fakultaet(int n);

int main() {
   cout << "Ganze Zahl >= 0 eingeben: ";
   int n;
   cin >> n;
   cout << n << "! = " << fakultaet(n) << endl;
}

long fakultaet(int n) {
   if(n < 2) {
      return 1;        // Rekursionsabbruch
   }
   return n*fakultaet(n-1);
}
```

**3.4**

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

void bewegen(int n, int a, int b, int c) {
   while (n > 0) {
      bewegen(n - 1, a, c, b);
      cout << "Bringe eine Scheibe von " << a
           << " nach " << b << endl;
      --n;
      int t = a; a = c; c = t;
   }
}

int main() {
   cout << "Türme von Hanoi! Anzahl der Scheiben: ";
   int scheiben;
   cin >> scheiben;
   bewegen(scheiben, 1, 2 ,3);
}
```

**3.5**

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

void str_umkehr(string& s);
```

```
int main() {
   cout << "Reihenfolge der Zeichen umdrehen. Zeichenkette eingeben:";
   string str;
   cin >> str;
   str_umkehr(str);
   cout << str << endl;
}

void str_umkehr(string& s) { // dreht die Reihenfolge der Zeichen um
    int links = 0, rechts = s.length() - 1;
    while(links < rechts) {
        char temp = s[links];
        s[links++] = s[rechts];
        s[rechts--] = temp;
    }
}
```

**3.6** Aus Platzgründen wird die Lösung nicht abgedruckt. Sie ist aber vollständig in den Beispielen enthalten (siehe *cppbuch/loesungen/k3/5.cpp*).

**3.7**

```
#include<iostream>
using namespace std;
bool istAlphanumerisch(const string& text);   // Proptotyp

int main() {
   string einText;
   cout << "Zeichenfolge eingeben:";
   getline(cin, einText);
   if(istAlphanumerisch(einText)) {
      cout << "Die eingegebene Zeichenkette enthält "
         "nur Buchstaben und Ziffern." << endl;
   }
   else {
      cout << "Die eingegebene Zeichenkette enthält NICHT "
         "nur Buchstaben und Ziffern." << endl;
   }
}

bool istAlphanumerisch(const string& text) {
   bool ergebnis = true;
   for(size_t i = 0; i < text.length(); ++i) {
      char zeichen = text.at(i);
      bool istZiffer = zeichen >= '0' && zeichen <= '9';
      bool istBuchstabe = (zeichen >= 'A' && zeichen <= 'Z')
         || (zeichen >= 'a' && zeichen <= 'z');
      // Vorzeile: && bindet stärker, die Klammern sind nur zur besseren Lesbarkeit
      if(!istZiffer && !istBuchstabe) {
         ergebnis = false;
         break; // weitere Prüfungen sind nicht notwendig
      }
   }
   return ergebnis;
}
```

**3.8** Die Lösung ist einfach, wenn wir bedenken, dass es sich um eine bloße *Textersetzung* handelt. `QUAD(x+1)` würde *ohne* die Klammern `x+1*x+1` und damit ein falsches arithmetisches Ergebnis liefern. *Mit* Klammern gibt es in diesem Fall keine Probleme: `((x+1)*(x+1))` (siehe jedoch Seite 130). Die äußeren Klammern sind wichtig, damit `QUAD(x)` in einem zusammengesetzten Ausdruck verwendet werden kann.

**3.9** • *taschenrechner.h*

```
#ifndef TASCHENRECHNER_H
#define TASCHENRECHNER_H
long ausdruck(char& c);
long summand(char& c);
long faktor(char& c);
long zahl(char& c);
#endif
```

Diese Datei wird mit #include in *main.cpp* und *taschenrechner.cpp* eingebunden. Wegen der sehr großen Ähnlichkeit des Restes der Lösung mit der auf den Seiten 118 ff. abgedruckten wird hier aus Platzgründen auf eine Wiedergabe verzichtet. In den Beispielen (Verzeichnis *cppbuch/loesungen/k3/8*) ist das Programm vollständig vorhanden.

**3.10** • *gettype.t*

```
#ifndef GETTYPE_T
#define GETTYPE_T
#include<string>
using std::string;

// Template
template<typename T>
string getType(T t) { return "unbekannter Typ";}

// Template-Spezialisierungen
template<> string getType(int t) { return "int";}
template<> string getType(unsigned int t) { return "unsigned int";}
template<> string getType(double t) { return "double";}
template<> string getType(char t) { return "char";}
template<> string getType(bool t) { return "bool";}
#endif
```

• *main.cpp*

```
#include<iostream>
#include"gettype.t"
using namespace std;

int main() {
  int i;
  cout << getType(i) << endl;
  unsigned int ui;
  cout << getType(ui) << endl;
```

```
   float f; // nicht in getType() berücksichtigt!
   cout << getType(f) << endl;
   double d;
   cout << getType(d) << endl;
   char c;
   cout << getType(c) << endl;
   bool b;
   cout << getType(b) << endl;
}
```

**3.11** • *betrag.t*

```
#ifndef BETRAG_T
#define BETRAG_T
#include<iostream>
#include<cstdlib> // für exit()

// Template
template<typename T>
T betrag(T t) {
   return (t < 0) ? -t : t;
}

// Template-Spezialisierung
template<> char betrag(char c) {
   std::cerr << "Betrag von 'char' ist undefiniert" << std::endl;
   exit(1);
   return c; // damit der Compiler zufrieden ist (wg. exit() nicht erreichbar)
}
// Template-Spezialisierung
template<> bool betrag(bool b) {
   std::cerr << "Betrag von 'bool' ist undefiniert" << std::endl;
   exit(1);
   return b;
}
#endif
```

• *main.cpp*

```
#include<iostream>
#include"betrag.t"
using namespace std;
int main() {
   int i = -1;
   cout << "Der Betrag von " << i << " ist " << betrag(i) << endl;
   double d = -2.345;
   cout << "Der Betrag von " << d << " ist " << betrag(d) << endl;
   // Fehlermeldung provozieren
   bool b = true;
   cout << "Der Betrag von " << b << " ist " << betrag(b) << endl;
}
```

**3.12** Der erste Fehler steckt in der Anweisung `temp = feld[0];`, weil diese Anweisung die Existenz von mindestens einem Vektorelement voraussetzt. Die Funktion würde

bei einem *leeren* Vektor versagen und möglicherweise »crashen«. Der zweite Fehler ist nicht ganz so leicht zu finden. Die Funktion sortiert einwandfrei, wenn alle Elemente verschieden sind, nicht aber, wenn es gleiche Elemente gibt und die auch noch die größten sind. Zum Beispiel wird die Folge 1200, 1200, 38, 1, 0, 3, 99, 1010, 4 nicht korrekt sortiert. Der Algorithmus verwendet die Überlegung: Nur *eine* Vertauschung ändert schon `temp`, weswegen es als Indikator genommen werden kann. Der Fehler: Dies gilt nicht, wenn nach der letzten Vertauschung `temp` genau den Wert hat, den auch `feld[0]` hat (größtes Element). Die Behauptung im Quellcode »// keine Vertauschung mehr« und auch die Argumentation im Aufgabentext sind also falsch.

**3.13** Um Mehrfachberechnungen der Potenzen von $x$ zu vermeiden, wird das Polynom durch geschickte Klammerung umformuliert:
$((((((\ldots k_3)x + k_2)x + k_1)x + k_0$ (sogenanntes Horner-Schema).

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

double polynom(const vector<double>& koeff, double x) {
   int n = koeff.size()-1;
   double ergebnis = koeff[n];
   for(int i = n-1; i >= 0 ; --i) {
      ergebnis *= x;
      ergebnis += koeff[i];
   }
   return ergebnis;
}

int main() {
   vector<double> koeffizienten(3);
   koeffizienten[0] = 1.1;
   koeffizienten[1] = 2.22;
   koeffizienten[2] = 13.0;
   cout << polynom(koeffizienten, 2.04) << endl;
   cout << polynom(koeffizienten, 3.033) << endl;
}
```

**3.14**

```
/* Dieses Programm listet sich selbst */
#include <string>
#include <iostream>
using namespace std;
char AS = 34;       // Anführungsstriche
char BS = 92;       // Backslash
char NZ = 10;       // neue Zeile
void c(const string& t) {
  cout << t << AS;
  unsigned int i = 0;
  while (i < t.length()) {
     if (t[i] == NZ) cout << BS << 'n'<<AS<< t[i] << AS;
     else cout << t[i];
     ++i;
```

```
  }
  cout<<AS<<')'<<';'<<'c'<<'('<<'s'<<')'<<';'<<'}'<<NZ;
}
int main(){ string s("/* Dieses Programm listet sich selbst */\n"
"#include <string>\n"
"#include <iostream>\n"
"using namespace std;\n"
"char AS = 34;          // Anführungsstriche\n"
"char BS = 92;          // Backslash\n"
"char NZ = 10;          // neue Zeile\n"
"void c(const string& t) {\n"
"  cout << t << AS;\n"
"  unsigned int i = 0;\n"
"  while (i < t.length()) {\n"
"      if (t[i] == NZ) cout << BS << 'n'<<AS<< t[i] << AS;\n"
"      else cout << t[i];\n"
"      ++i;\n"
"  }\n"
"  cout<<AS<<')'<<';'<<'c'<<'('<<'s'<<')'<<';'<<'}'<<NZ;\n"
"}\n"
"int main(){ string s("");c(s);}
```

**3.15** Eine `unsigned`-Zahl ist immer größer oder gleich 0, deswegen kann der erste Teil der Bedingung nie wahr werden. Eine `unsigned`-Zahl kann niemals größer als `UINT_MAX` sein, weil `UINT_MAX` per Definition die größte `unsigned`-Zahl ist. Damit kann auch der zweite Teil der Bedingung nie wahr werden. Die Funktion ist sinnlos.

## Kapitel 4

**4.1**

```
Rational add(long a, const Rational& b) {
   Rational r(a);
   r.add(b);
   return r;
}

Rational add(const Rational& a, long b) {
  return add(b, a); // Aufruf von add(long, const Rational&)
}
```

**4.2**

```
void ausgabeEinerRationalzahl(const Rational& r) {
    std::cout << r.Zaehler() << "/" << r.Nenner();
    std::cout << std::endl;
}
```

**4.3** Lösungsbeispiel

- *IntMenge.h*

```
// Klasse zur Implementierung eines Datentyps für Mengen mit int-Elementen
#ifndef IntMenge_h
#define IntMenge_h
#include<cstddef> // size_t
```

```
#include<vector>

class IntMenge {
 public:
   IntMenge();
   void hinzufuegen(int el);
   void entfernen(int el);
   bool istMitglied(int el) const;
   size_t size() const;
   void anzeigen() const;
   void loeschen();  // alle Elemente löschen
   int getMax() const; // größtes Element
   int getMin() const; // kleinstes Element
 private:
   size_t anzahl;
   std::vector<int> vec;
   // Die Hilfsfunktion finden() gibt die Position des
   // Elements zurück. -1 bedeutet nicht vorhanden
   int finden(int el) const;
};
#endif
```

Die private Hilfsfunktion `finden(int el)` gibt die Position des Elements `el` zurück. Sie wird intern zur Vermeidung von Code-Duplizierung verwendet. Wenn es sie nicht gäbe, müssten `entfernen()` und `istMitglied()` mit einer Schleife versehen werden.

- *IntMenge.cpp*

```
#include"IntMenge.h"
#include<iostream>
#include<cassert>

IntMenge::IntMenge()
   : anzahl(0) {
}
```

Die folgende Methode `hinzufuegen()` nutzt aus, dass ein `vector` dynamisch mit `push_back()` vergrößerbar ist. Die Variable `anzahl` gibt die tatsächliche Anzahl der gespeicherten Elemente an. Sie kann kleiner als die Größe des Vektors sein, nämlich dann, wenn Elemente gelöscht worden sind.

```
void IntMenge::hinzufuegen(int el) {
   if(!istMitglied(el)) { // ansonsten ignorieren
      if(anzahl < vec.size()) {
         vec[anzahl] = el;
      }
      else { // Platz reicht nicht
         vec.push_back(el);
      }
      ++anzahl;
   }
}
```

Ein Element wirklich zu löschen, hieße den Vektor zu verkleinern – eine zeitaufwendige Operation. Da die Reihenfolge der Elemente in einer Menge nicht sortiert sein muss, bietet sich stattdessen an, das letzte Element an die Stelle des zu löschenden zu kopieren. Wenn dann noch `anzahl` um eins heruntergezählt wird, ist das vorherige letzte Element nicht mehr erreichbar, denn alle Schleifen in den folgenden Methoden haben `anzahl` als Grenze. Der freigewordene Platz steht für `hinzufuegen()` zur Verfügung. Nach dieser Logik ist auch das Löschen aller Elemente denkbar schnell und einfach: `anzahl` wird auf 0 gesetzt (siehe Methode `loeschen()`).

```
void IntMenge::entfernen(int el) {
   int wo = finden(el);
   if(wo > -1) {
      vec[wo] = vec[--anzahl]; // letztes Element umkopieren
   }
}

bool IntMenge::istMitglied(int el) const {
   return finden(el) > -1;
}

size_t IntMenge::size() const {
   return anzahl;
}

void IntMenge::anzeigen() const {
   for(size_t i=0; i < anzahl; ++i) {
      std::cout << vec[i] << " ";
   }
   std::cout << std::endl;
}

void IntMenge::loeschen() {
   anzahl = 0;
}

int IntMenge::finden(int el) const {
   for(size_t i=0; i < anzahl; ++i) {
      if(vec[i] == el)
         return i;
   }
   return -1; // nicht gefunden
}

int IntMenge::getMin() const {
   assert(anzahl > 0);
   int erg = vec[0];
   for(size_t i=1; i < anzahl; ++i) {
      if(vec[i] < erg)
         erg = vec[i];
   }
   return erg;
}
```

```
int IntMenge::getMax() const {
   assert(anzahl > 0);
   int erg = vec[0];
   for(size_t i=1; i < anzahl; ++i) {
      if(vec[i] > erg)
         erg = vec[i];
   }
   return erg;
}
```

Man kann sich noch einige Optimierungen vorstellen. Zum Beispiel könnte der erste Aufruf von `getMin()` oder `getMax()` sowohl Minimum als auch Maximum ermitteln, und die Werte könnten in entsprechenden Attributen gespeichert werden (Cache). Eine zweite Abfrage würde dann einen gespeicherten Wert zurückgeben und wäre damit sehr schnell. Ein erneutes Durchlaufen der Schleife wäre nur beim Hinzufügen oder Entfernen fällig und auch nur, wenn Minimum oder Maximum betroffen wären. Auch kann man sich überlegen, dass die Schleifen überhaupt zu aufwendig sind – dann bräuchte man allerdings eine andere Datenstruktur. Die Klasse `set` der C++-Bibliothek verwendet deshalb eine Variante des binären Suchbaums.

**4.4** Die mehrseitige Lösung wird aus Platzgründen nicht abgedruckt. Sie ist aber vollständig in den Beispielen (Verzeichnis *cppbuch/loesungen/k4/4*) enthalten.

**4.5** Man kann im `public`-Bereich eine Referenz auf `const` einfügen, die auf ein privates Attribut verweist. Da man einer Referenz nichts zuweisen kann, muss sie im Konstruktor initialisiert werden.

```
#include<iostream>

class MeineKlasse {
    public:
       MeineKlasse()
       : readonlyZahl(privateZahl), // Initialisierung der Referenz
         privateZahl(0) {
       }

       void aendern(int wert) {
          privateZahl = wert;
       }

       // public-Referenz auf Konstante, Initialisierung im Konstruktor
       const int& readonlyZahl;
    private:
       int privateZahl;
};

using namespace std;

int main() {
    MeineKlasse objekt;
    // objekt.privateZahl = 999; // Fehler! Zugriff nicht möglich!
    // objekt.readonlyZahl = 999; // Fehler! Änderung nicht möglich!
```

```
    objekt.aendern(999);            // erlaubte Änderung
    // erlaubter direkter lesender Zugriff:
    cout << "objekt.readonlyZahl=" << objekt.readonlyZahl << endl; // 999
}
```

**4.6** • *taschenrechner.h*

```
#ifndef TASCHENRECHNER_H
#define TASCHENRECHNER_H
#include<string>

class Taschenrechner {
 public:
   Taschenrechner(const std::string&);
   const std::string& getAnfrage();
   long getErgebnis();
 private:
   long ausdruck(char& c);
   long summand(char& c);
   long faktor(char& c);
   long zahl(char& c);
   void get(char& c);
   std::string anfrage;
   size_t position;
   long ergebnis;
};
#endif
```

• *taschenrechner.cpp*

```
#include"taschenrechner.h"
#include<cctype>
#include<iostream>

Taschenrechner::Taschenrechner(const std::string& str)
   : anfrage(str), position(0), ergebnis(0L) {
   char c;
   get(c); // 1. Zeichen lesen
   ergebnis = ausdruck(c);
}

const std::string& Taschenrechner::getAnfrage() {
  return anfrage;
}

long Taschenrechner::getErgebnis() {
   return ergebnis;
}

void Taschenrechner::get(char& c) {
   do {
      if(position >= anfrage.length()) {
         c = '#'; // ungültiges Zeichen, d.h. Abbruch
      }
```

```
      else {
         c = anfrage[position++];
      }
   } while(c == ' '); // Leerzeichen ignorieren
}

long Taschenrechner::ausdruck(char& c) { // Übergabe per Referenz!
// Der weggelassene Rest ist wie auf Seite 119, nur dass get(c);
// statt cin.get(c); verwendet wird.
  // ...
  return a;
}
```

Aus Platzgründen und weil die Struktur nach vorstehendem Muster klar ist, wurden die restlichen Funktionen weggelassen. In den Beispielen (Verzeichnis *cppbuch/loesungen/k4/6*) ist das Programm vollständig vorhanden.

## Kapitel 5

**5.1** Ja. Aus der Gleichheit von `(kosten+i)` und `(i+kosten)` und aus der Kenntnis, dass der Compiler *stets* die Umwandlung in die Zeigerdarstellung von `[ ]` vornimmt, folgt, dass man genausogut `i[kosten]` statt `kosten[i]` formulieren kann. Es ist jedoch absolut unüblich und erschwert die Lesbarkeit des Programms.

**5.2** `sizeof(int)*dim1*dim2` = 24, Bytenummer = `(i*dim2+j)*sizeof(int)`. Daraus ergibt sich, dass `dim1` nur zur Berechnung des Speicherplatzes gebraucht wird, aber nicht zur Adressberechnung, zu der jedoch alle weiteren Dimensionen benötigt werden.

**5.3** Es muss m = p, r = n und s = q gelten, damit die Matrizenmultiplikation definiert ist. Daher benötigt man nur noch drei Konstanten. Für die Funktion `tabellenausgabe2D()` siehe Seite 212.

```
int main() {
   // Initialisierung (Beispiel)
   const int N = 2, M = 3, Q = 4;
   int a[N][M] = {{1,2,3}, {4,5,6}};
   int b[M][Q] = {{1,2,3,0},{4,1,1,5},{1,7,1,4}};
   int c[N][Q];
   for(int i = 0; i < N; ++i) {              // Multiplikation
      for(int j = 0; j < Q; ++j) {
         c[i][j] = 0;
         for(int k = 0; k < M; ++k) {
            c[i][j] += a[i][k] * b[k][j];
         }
      }
   }
   // Ergebnis ausgeben oder weiterrechnen
   tabellenausgabe2D(a, N);
   cout << "multipliziert mit " << endl;
   tabellenausgabe2D(b, M);
   cout << "ergibt" << endl;
   tabellenausgabe2D(c, N);
```

```
    // ...
}
```

**5.4** Da eine echte dreidimensionale Ausgabe in der Ebene nicht möglich ist, werden n (DIM2 x DIM3)-Matrizen ausgegeben.

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

template<typename Feldtyp>
void Tabellenausgabe3D(Feldtyp T, size_t n) {
   const size_t DIM2 = sizeof T[0] /sizeof T[0][0];
   const size_t DIM3 = sizeof T[0][0] /sizeof T[0][0][0];
   for(size_t i = 0; i < n; ++i) {
      for(size_t j = 0; j < DIM2; ++j) {
         for(size_t k = 0; k < DIM3; ++k)
            cout << T[i][j][k] << ' ';
         cout << endl;
      }
      cout << endl;
   }
   cout << endl;
}

int main() {
   const int N = 2, M = 3, Q = 4;
   int mat3D[N][M][Q]; // 3D-Matrix
   // Mit Werten füllen
   int m = 0;
   for(int i = 0; i < N; ++i) {
      for(int j = 0; j < M; ++j) {
         for(int k = 0; k < Q; ++k)
            mat3D[i][j][k] = ++m;
      }
   }
   Tabellenausgabe3D(mat3D, N);
}
```

**5.5** Die Funktion entspricht der Funktion `strcpy()` der C++-Standardbibliothek.

```
void strcopy(char *ziel, const char *quelle) {
// kopiert den Inhalt von quelle in den String ziel
// (und überschreibt den vorherigen Inhalt dabei).
    while((*ziel++ = *quelle++));
}
```

**5.6**

```
char* strduplikat(const char *s) {
// liefert einen Zeiger auf den neu erzeugten String.
// ACHTUNG: Der Aufrufer ist für das delete verantwortlich!
   char* neu = new char[strlen(s)+1];
   strcpy(neu, s); // wie strcopy() von oben
   return neu;
}
```

**5.7**

```
// Vergleichsfunktion für C-String-Array-Elemente
int scmp(const void *a, const void *b) {
   // Umwandlung in einen const-Zeiger auf einen C-String, dh. auf const char *
   const char *pa = *static_cast<const char* const*>(a);
   const char *pb = *static_cast<const char* const*>(b);
   return strcmp(pa,pb);
}
```

Quicksort wird ähnlich wie im Textbeispiel aufgerufen. Da wir ein Feld von Zeigern vor uns haben, wird als Elementgröße die Größe eines Zeigers auf `char` übergeben: `qsort(sfeld, size, sizeof(char*), scmp);`.

Alternative Lösung mit der Standardfunktion `std::sort()`:

```
#include<iostream>
#include<cstring>
#include<string>
#include<algorithm> // enthält sort()
using namespace std;

// Vergleichsfunktion für C-String-Array-Elemente
bool scmp(const char *a, const char *b) {
   return strcmp(a, b) < 0;
}

int main() {
   const char* sfeld[] = {"eins", "zwei", "drei", "vier", "fünf",
                          "sechs", "sieben", "acht", "neun", "zehn"};
   size_t anzahlElemente = sizeof(sfeld)/sizeof(sfeld[0]);
   std::sort(sfeld, sfeld + anzahlElemente, scmp);
   // ALPHABETISCHE Ausgabe des sortierten Feldes:
   for(size_t i = 0; i < anzahlElemente; ++i)
      cout << ' ' << sfeld[i];
   cout << endl;

   // Entsprechend für C++-Strings
   string strings[] = {"eins", "zwei", "drei", "vier", "fünf",
                       "sechs", "sieben", "acht", "neun", "zehn"};
   size_t anzahl = sizeof(sfeld)/sizeof(sfeld[0]);
   // bei C++-Strings ist keine Vergleichsfunktion notwendig
   std::sort(strings, strings + anzahl);
   // ALPHABETISCHE Ausgabe des sortierten Feldes:
   for(size_t i = 0; i < anzahl; ++i)
      cout << ' ' << strings[i];
   cout << endl;
}
```

**5.8**

```
void leerzeichenEntfernen(char* s) {
   char* q = s;
   do  {
      if(*s != ' ') {
         *q++ = *s;
      }
```

```
   } while(*s++);
}
```

**5.9**

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

int main( int argc, char* argv[]) {
   cout << "Dateien ausgeben" << endl;
   if(argc == 1) {
      cout << "Keine Dateinamen in der Kommandozeile gefunden.\n"
           << "Gebrauch: " << argv[0] // Programmname
           << " datei1 datei2 usw." << endl;
      return 0;
   }
   int nr = 0;
   while(argv[++nr] != 0) {
      ifstream quelle;
      quelle.open(argv[nr], ios::binary|ios::in);
      cout << "Datei " << argv[nr];
      if(!quelle) {                  // Fehlerabfrage
         cout << " nicht gefunden." << endl;
         continue;                   // weiter bei while
      }
      cout << ":" << endl;
      char ch;
      while(quelle.get(ch)) {
         cout << ch;                 // zeichenweise ausgeben
      }
      quelle.close();
   }
}
```

**5.10** Ausgabe von Namen in einer Datei

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

bool istBuchstabe(char c) {  // vgl. isalpha(), Seite 875
   return c >= 'A' && c <= 'Z'
       || c >= 'a' && c <= 'z'
       || c == '_';
}

bool istAlphanumerisch(char c) { // vgl. isalnum(), Seite 875
   return c >= '0' && c <= '9'
      || istBuchstabe(c);
}

int main( int argc, char* argv[]) {
   if(argc == 1) {
      cout << "Kein Dateiname in der Kommandozeile gefunden."
```

```
         " Gebrauch: " << argv[0]   // Programmname
           << " dateiname" << endl;
      return 0;
   }
   ifstream quelle(argv[1]);
   if(!quelle) {                  // Fehlerabfrage
      cout << "Datei " << argv[1] << " nicht gefunden." << endl;
      return 0;
   }
   char ch;
   bool namengefunden = false;
   while(quelle.get(ch)) {
      if(istBuchstabe(ch)) {
         cout << ch;
         namengefunden = true;
      }
      else if(namengefunden && istAlphanumerisch(ch)) {
         cout << ch;
      }
      else if(namengefunden) {
         namengefunden = false;
         cout << endl;
      }
   }
   quelle.close();
}
```

## Kapitel 6

**6.1**

```
void MeinString::insert(size_t pos, const MeinString& m) {
   // m vor pos einfügen
   if(pos > len) {
      pos = len;
   }
   reserve(len + m.len);
   // Teil hinter pos verschieben
   size_t neuesende = len + m.len;
   for(size_t anz = 0; anz <= len-pos; ++anz) {
      start[neuesende] = start[neuesende-m.len];
      --neuesende;
   }
   // m einfügen
   const char* temp = m.start;
   while(*temp) {
      start[pos++] = *temp++;
   }
   len = len + m.len;              // Verwaltungsinformation aktualisieren
}
```

**6.2** • *format.h*

```
#ifndef FORMAT_H
```

```
#define FORMAT_H
#include<string>
using std::string;

class Format {
 public:
   Format(int weite, int nachk);
   string toString(double d) const;
 private:
   int weite;
   int nachkommastellen;
};
#endif
```

• *format.cpp*

```
#include"format.h"
#include<iostream>
using namespace std;

Format::Format(int w, int nk)
   : weite(w), nachkommastellen(nk) {
   if(nk < 0) nk = 0;
   if(nk > 16) nk = 16;
   if(w < nk) w = nk+1;
}

string Format::toString(double d) const {
   string ergebnis;
   bool negativ = false;
   if(d < 0.0) {
      negativ = true;
      d = -d;
   }
   // Rundung
   double rund = 0.5;
   for(int i=0; i < nachkommastellen; ++i)
      rund /= 10.0;
   d += rund;
   // Mit der folgenden Normierung (d.h. Zahl beginnt mit 0,...) wird erreicht, dass
   // die Anzahl der Stellen vor dem Komma bekannt ist (Stellenwert).
   int stellenwert = 0;
   // Zahl normieren, falls >=1
   while(d >= 1.0) {
      ++stellenwert;
      d /= 10.0;
   }
   if(stellenwert == 0) {
      ergebnis += '0';  // wenigstens eine 0 vor dem Komma
   }

    // Die Zahl wird sukzessive mit 10 multipliziert, die jeweils erste Ziffer
    // zunächst ermittelt (zif), dann abgetrennt und an den Ergebnis-String
```

```
    // gehängt usw.
   do {
      if(stellenwert == 0) {
         ergebnis += ',';                  // Komma
      }
      d *= 10.0;
      int zif = (int)d;
      d -= zif;
      ergebnis += (char)zif + (int)'0';
      --stellenwert;
   } while(nachkommastellen + stellenwert > 0);
   if(negativ) {
      ergebnis = '-' + ergebnis;
   }
   int diff = weite - ergebnis.length();
   for(int i=0; i < diff; ++i) {
      ergebnis = " " + ergebnis;
   }
   return ergebnis;
}
```

### 6.3 • *teilnehmer.h*

Bei der Speicherung in einem `vector<Teilnehmer*>` müssen alle verbundenen Teilnehmer im selben Gültigkeitsbereich sein! Der Grund: Wenn die Lebensdauer unterschiedlich ist, können ungültige Referenzen entstehen. Beispiel:

```
Teilnehmer otto("Otto");
{
   Teilnehmer andrea("Andrea");
   otto.lerntKennen(andrea);  // alles bestens
}
otto.druckeBekannte();       // ups! Andrea ist futsch!
```

Aus diesem Grund ist es günstiger, nur die Namen zu speichern.

```
#ifndef TEILNEHMER_H
#define TEILNEHMER_H
#include<string>
#include<vector>
using std::string;
using std::vector;
```

```
class Teilnehmer {
 public:
   Teilnehmer(const string& name);
   void lerntKennen(Teilnehmer& tn);
   bool kennt(const Teilnehmer& tn) const;
   void druckeBekannte() const;
   const string& gibNamen() const;
 private:
   string name;
   vector<string> dieBekannten;
};
```

```
#endif
```

• *teilnehmer.cpp*

```
#include"teilnehmer.h"
#include<iostream>
using std::cout;
using std::endl;

Teilnehmer::Teilnehmer(const string& n)
   : name(n) {
}

void Teilnehmer::lerntKennen(Teilnehmer& tn) {
   if(&tn != this // 'sich selbst kennenlernen' ignorieren
      && !kennt(tn) ) { // wenn noch unbekannt, eintragen
      dieBekannten.push_back(tn.gibNamen());
      tn.lerntKennen(*this); // wechselseitig kennenlernen
   }
}

bool Teilnehmer::kennt(const Teilnehmer& tn) const {
   bool erg = false;
   for(size_t i = 0; i < dieBekannten.size(); ++i) {
      if(tn.gibNamen() == dieBekannten.at(i)) {
         erg = true;
         break;
      }
   }
   return erg;
}

void Teilnehmer::druckeBekannte() const {
   for(size_t i = 0; i < dieBekannten.size(); ++i) {
      cout << " " << dieBekannten.at(i);
   }
   cout << endl;
}

const string& Teilnehmer::gibNamen() const {
   return name;
}
```

## Kapitel 7

**7.1** Nein. Die Funktion kann nicht mehr von `GraphObj` geerbt werden, ohne dass `Strecke` abstrakt wird. Für die Klasse `Strecke` muss eine überladene Elementfunktion `flaeche()` mit dem Rückgabewert 0 geschrieben werden.

**7.2** • *person.h*

```
#ifndef PERSON_H
#define PERSON_H
```

```
#include<string>
using std::string;

class Person {
 public:
   Person(const string& n, const string& v)
      : nachname(n), vorname(v) {
   }
   const string& getNachname() const { return nachname;}
   const string& getVorname() const { return vorname;}
   virtual string toString() const = 0;
   virtual ~Person(){}
 private:
   string nachname;
   string vorname;
};
// Die Standardimplementierung einer rein virtuellen Methode
// muss nach [ISOC++] außerhalb der Klassendefinition stehen:
inline string Person::toString() const {
     return vorname + " " + nachname;
}
#endif
```

- *student.h*

```
#ifndef STUDENT_H
#define STUDENT_H
#include"person.h"
#include<string>
using std::string;

class StudentIn : public Person {
 public:
   StudentIn(const string& name, const string& vorname,
             const string& matnr)
      : Person(name, vorname), matrikelnummer(matnr) {
   }
   const string& getMatrikelnummer() const {
      return matrikelnummer;
   }
   virtual string toString() const {
      return "Student/in " + Person::toString()
         + ", Mat.Nr.: " + matrikelnummer;
   }
   virtual ~StudentIn(){}
 private:
   string matrikelnummer;
};
#endif
```

- *prof.h*

```
#ifndef PROF_H
#define PROF_H
```

```
#include"person.h"
#include<string>
using std::string;

class ProfessorIn : public Person {
 public:
   ProfessorIn(const string& nachname, const string& vorname,
               const string& lgb)
      : Person(nachname, vorname), lehrgebiet(lgb) {
   }
   const string& getLehrgebiet() const {
      return lehrgebiet;
   }
   virtual string toString() const {
      return "Prof. " + Person::toString()
         + ", Lehrgebiet: " + lehrgebiet;
   }
   virtual ~ProfessorIn(){}
 private:
   string lehrgebiet;
};
#endif
```

**7.3** Da im obigen Programm Zeiger auf `Person` verwendet werden, erfordert ein Zugriff auf Methoden, die nicht in `Person` deklariert sind, eine Typumwandlung. Beispiel:

```
cout << "Die Matrikelnummer von "
    << diePersonen[0]->getNachname() << " ist "
    << ((StudentIn*)diePersonen[0])->getMatrikelnummer() // !
    << endl;
```

Die Typumwandlung in den Typ `StudentIn*` funktioniert natürlich nur, wenn man genau weiß, dass der Zeiger an der Stelle `[0]` auf ein Objekt des dynamischen Typs `StudentIn` verweist. Was aber, wenn man es nicht genau weiß? Dazu geben die Abschnitte 7.9 und 7.10 Auskunft.

**7.4**

```
cout << "Die Matrikelnummern mit dynamic_cast: " << endl;
for(size_t i = 0; i < diePersonen.size(); ++i) {
   cout << diePersonen[i]->getVorname() << ": ";
   StudentIn* ps = dynamic_cast<StudentIn*>(diePersonen[i]);
   if(ps) {
      cout << ps->getMatrikelnummer() << endl;
   }
   else {
       cout << " hat keine Matrikelnummer." << endl;
   }
}
```

**7.5**

```
cout << endl << "Die Matrikelnummern mit typeid: " << endl;
for(size_t i = 0; i < diePersonen.size(); ++i) {
   cout << diePersonen[i]->getVorname();
   if(typeid(StudentIn) == typeid(*diePersonen[i])) {
```

```
      cout << ": "
           << ((StudentIn*)(diePersonen[i]))->getMatrikelnummer()
           << endl;
   }
   else {
      cout << " (interner Typ: "
           << typeid(*diePersonen[i]).name()
           << ") hat keine Matrikelnummer." << endl;
   }
}
```

## Kapitel 9

**9.1** Die Referenz auf den `Rational`-Parameter in der Deklaration darf nicht `const` sein, weil das Objekt verändert wird:

```
// Deklaration als globale Funktion
std::istream& operator>>(std::istream&, Rational&);

// Implementation
std::istream& operator>>(std::istream& eingabe, Rational& r){
     // cerr wird gewählt, damit die Abfragen auch dann
     // auf dem Bildschirm erscheinen, wenn die Standard-
     // ausgabe in eine Datei zur Dokumentation geleitet wird.
     int z, n;
     std::cerr << "Zähler :";
     eingabe >> z;
     std::cerr << "Nenner :";
     eingabe >> n;
     assert(n != 0);  // nicht sehr benutzungsfreundlich ...
     r.set(z, n);
     r.kuerzen();
     return eingabe;
}
```

Anmerkung: Hier wurde die Methode `eingabe()` mit dem Operator nachgebildet. Der bessere Programmierstil ist, die Funktionen der Ein- und Ausgabe zu trennen, sodass die Aufforderung zur Zahleneingabe nicht Bestandteil des Eingabeoperators ist.

**9.2** Der Operator `+=` verändert das Objekt selbst, denn `a += b;` ist nur eine Abkürzung für `a = a+b;`. Daher kann er, vordergründig betrachtet, als Elementfunktion mit nur einem Argument und Rückgabetyp `void` deklariert werden:

```
void operator+=(Rational);.
```

Die Implementierung könnte wie folgt aussehen:

```
void Rational::operator+=(Rational b) { // nicht optimal
     zaehler = zaehler*b.nenner + b.zaehler*nenner;
     nenner = nenner*b.nenner;
     kuerzen();
}
```

Um Verkettungen wie `c = a += b;`, die zu `c = a.operator+=(b)` aufgelöst werden sowie die Verwendung innerhalb des binären `operator+()` zu erlauben, muss ein Objekt des passenden Datentyps zurückgegeben werden, also ein Objekt der Klasse `Rational` (statt `void` wie vorher). Um die Konstruktion von temporären Objekten durch den Kopierkonstruktor bei der Ergebnisrückgabe zu vermeiden, wird die Referenz auf das Zielobjekt zurückgegeben. Die Referenz auf `const` in der Parameterliste erspart die Kopie beim Eintritt in die Funktion.

```
Rational& Rational::operator+=(const Rational& b) {
    zaehler = zaehler*b.nenner + b.zaehler*nenner;
    nenner = nenner*b.nenner;
    kuerzen();
    return *this;
}
```

**9.3** Deklaration in *ratioop.h* als Elementfunktion:

```
Rational& operator-=(const Rational&);
Rational& operator*=(const Rational&);
Rational& operator/=(const Rational&);
```

Ebenfalls in *ratioop.h* werden die binäre Operatoren als globale Funktionen deklariert. Dabei wird die Empfehlung von Seite 168 beachtet, den ersten Parameter per Wert zu übergeben, weil in der Funktion eine Kopie gebraucht wird. Das `const` bei dem Rückgabetyp verhindert unsinnige Anweisungen wie `(a + b) = c;`.

```
// globale Operatoren
const Rational operator+(Rational, const Rational&);
const Rational operator-(Rational, const Rational&);
const Rational operator*(Rational, const Rational&);
const Rational operator/(Rational, const Rational&);
```

Definition in *ratioop.cpp* als Elementfunktion:

```
Rational& Rational::operator-=(const Rational& b) {
    zaehler = zaehler*b.nenner - b.zaehler*nenner;
    nenner = nenner*b.nenner;
    kuerzen();
    return *this;
}

Rational& Rational::operator*=(const Rational& b) {
    zaehler *= b.zaehler;
    nenner *= b.nenner;
    kuerzen();
    return *this;
}

Rational& Rational::operator/=(const Rational& b) {
    zaehler *= b.nenner;
    nenner *= b.zaehler;
    kuerzen();
    return *this;
}
```

Definition der globalen Funktionen in *ratioop.cpp*:

```
const Rational operator+(Rational a, const Rational& b) {
   return a += b;
}

const Rational operator-(Rational a, const Rational& b) {
   return a -= b;
}

const Rational operator*(Rational a, const Rational& b) {
   return a *= b;
}

const Rational operator/(Rational a, const Rational& b) {
   return a /= b;
}
```

**9.4** Deklaration in *ratioop.h* als globale Funktion:

```
bool operator==(const Rational&, const Rational&);
```

Definition in *ratioop.cpp* als globale Funktion:

```
bool operator==(const Rational& a, const Rational& b) {
   return  a.getZaehler() == b.getZaehler()
        && a.getNenner() == b.getNenner();
}
```

Es wird hier angenommen, dass beide Zahlen in der gekürzten Darstellung vorliegen, weil dies durch die Elementfunktionen erzwungen wird. Andernfalls müssten beide Argumente vor dem Vergleich gekürzt werden.

**9.5** `operator=()` darf *nichts* tun. Schließlich darf die `SerienNr` als Konstante eines Objekts nicht verändert werden. Der Sinn des Operators besteht nur darin, Zuweisungsoperationen im Programm zu erlauben, ohne dass der Compiler meckert. Dies ist wichtig, wenn von der Klasse `NummeriertesObjekt` geerbt wird, weil bei der Zuweisung eines Objekts der abgeleiteten Klasse die Zuweisungsoperatoren der Elemente der Klasse inklusive der anonymen Subobjekte aufgerufen werden.

```
NummeriertesObjekt& operator=(const NummeriertesObjekt&) {
   return *this;
}
```

**9.6** Deklaration in *meinstring.h*

```
// als Elementfunktion
MeinString& operator=(const MeinString&); // Zuweisung
MeinString& operator=(const char *); // Zuweisung

// Indexoperator:
const char& operator[](std::size_t position) const;
// Indexoperator. Die Referenz erlaubt Ändern des Zeichens.
char& operator[](std::size_t position);
```

```
// global:
std::ostream& operator<<(std::ostream&, const MeinString&);
```

Implementierung in *meinstring.cpp*:

```
#include"meinstring.h"
#include<stdexcept>
#include<cstring>

namespace {
   void bereichPruefen(bool bedingung) {
      if(!bedingung) {
         throw std::out_of_range("MeinString: Bereichsüberschreitung");
      }
   }
}

MeinString& MeinString::operator=(const MeinString& m) {
   reserve_only(m.len);
   strcpy(start, m.start);
   len = m.len;
   return *this;
}

MeinString& MeinString::operator=(const char *s) {
   size_t temp = strlen(s);
   reserve_only(temp);
   strcpy(start, s);
   len = temp;
   return *this;
}

char& MeinString::operator[](size_t pos) { // Zeichen per Referenz holen
    bereichPruefen(pos >= 0 && pos<= len); // Nullbyte lesen ist erlaubt
    return start[pos];
}

const char& MeinString::operator[](size_t pos) const { // Zeichen holen
    bereichPruefen(pos >= 0 && pos<= len); // Nullbyte lesen ist erlaubt
    return start[pos];
}
```

In Analogie zum C++-Standard-Entwurf ist das Lesen des Nullbytes erlaubt, anders als bei der Funktion `at()`. Weil für nichtkonstante `MeinString`-Objekte die nichtkonstante Variante von `operator[]()` genommen wird, ergibt sich aus der Tatsache, dass auch eine Referenz auf das Nullbyte zurückgegeben werden kann, ein Schönheitsfehler: `operator[]()` erlaubt das Beschreiben des Nullbytes. `at()` kann deshalb nicht ohne Weiteres durch `char& operator[](int)` ersetzt werden, wenn nicht schreibend auf das Nullbyte zugegriffen werden darf. `operator[]()` entsprechend auch für diese Fälle abzusichern, ist mit wenig Aufwand nur möglich, wenn jede andere Methode das Nullbyte auf Veränderung prüft, was etwas Laufzeit kostet.

```
// Ausgabeoperator (globale Funktion)
std::ostream& operator<<(std::ostream& os, const MeinString& m) {
    os << m.c_str();
    return os;
}
```

**9.7** Deklaration in *meinstring.h*

```
// als Elementfunktion
MeinString& operator+=(const MeinString&); // Verketten
// global
MeinString operator+(MeinString, const MeinString&);
```

Implementierung in *meinstring.cpp*:

```
MeinString& MeinString::operator+=(const MeinString& m) { // Verketten
   char *p = new char[len + m.len + 1]; // neuen Platz beschaffen
   strcpy(p, start);        // Teil 1 kopieren
   strcpy(p + len, m.start); // Teil 2 kopieren
   delete [] start;          // alten Platz freigeben
   len   += m.len;           // Verwaltungsinformation aktualisieren
   start = p;
   return *this;
}

// Verketten
MeinString operator+(MeinString a, const MeinString& b) {
  return a += b;
}
```

Eine ausführliche Diskussion des Plus-Operators und seiner Optimierungsmöglichkeiten gibt es in Abschnitt 22.1.

**9.8** Ein Rückgabetyp `Datum&` erspart den impliziten Aufruf des Kopierkonstruktors.

**9.9** *Nein!* Die lokale Variable `temp` ist nach Verlassen der Operatorfunktion nicht mehr existent. Wenn weitere Erläuterungen nötig sein sollten: Schlagen Sie sie auf Seite 112 nach. In der vorhergehenden Aufgabe wird ein schon *vor* dem Eintritt in die Operatorfunktion existierendes Objekt zurückgegeben.

**9.10** Die folgenden Operatoren sind in *datum.h* zu deklarieren. Die Implementierung gehört nach *datum.cpp*. `#include<iostream>` nicht vergessen!

```
std::ostream& operator<<(std::ostream& os, const Datum& d) {
  os << d.tag() << '.' << d.monat() << '.' << d.jahr();
  return os;
}
```

**9.11**

```
bool operator==(const Datum& a, const Datum& b) {
  return   a.tag()  == b.tag()
        && a.monat() == b.monat()
        && a.jahr() == b.jahr();
}
```

```
bool operator!=(const Datum& a, const Datum& b) {
   return !(a == b);
}
bool operator<(const Datum& a, const Datum& b) {
   return   a.jahr() < b.jahr()
         || a.jahr() == b.jahr() && a.monat() < b.monat()
         || a.jahr() == b.jahr()
            && a.monat() == b.monat() && a.tag() < b.tag();
}
```

**9.12**

```
int datumDifferenz(const Datum& a, const Datum& b) {
   if(a == b) {       // kurzer Prozess bei Gleichheit
      return 0;
   }
   bool richtigeReihenfolge = a < b;
   Datum frueher = a;
   Datum spaeter = b;
   if(!richtigeReihenfolge) {// ggf. vertauschen
      frueher = b;
      spaeter = a;
   }
   int Differenz = 0;
   while(frueher != spaeter) { // nicht optimiert (tageweises Hochzählen)
      ++Differenz;
      ++frueher;
   }
   return richtigeReihenfolge ? Differenz : -Differenz;
}
```

**9.13** Das Ergebnis ist der 19.1.2038 (2147483647 Sekunden seit dem 1.1.1970), falls `time_t` einem 32-Bit-`int` entspricht. Dies ist auf vielen Unix-Systemen der Fall.

**9.14** • *ungueltigesdatumexception.h*:

```
#ifndef UNGUELTIGESDATUMEXCEPTION_H
#define UNGUELTIGESDATUMEXCEPTION_H
#include<stdexcept>
#include<string>

class UngueltigesDatumException : public std::runtime_error {
 public:
   UngueltigesDatumException(int t, int m, int j)
      : std::runtime_error(toString(t, m, j)) {
   }
 private:
   static std::string toString(int tag, int monat, int jahr) {
      std::string t = std::to_string(tag);
      std::string m = std::to_string(monat);
      std::string j = std::to_string(jahr);
      return t + "." + m + "." + j + " ist kein gültiges Datum.";
   }
};
#endif
```

Die Deklaration der Methode `set()` in *datum.h*:

```
void set(int t, int m, int j);
```

Die Methode `set()` in *datum.cpp* lautet:

```
void Datum::set(int t, int m, int j) {
   if(!istGueltigesDatum(t, m, j)) {
      throw UngueltigesDatumException(t, m, j);
   }
   tag_ = t;
   monat_ = m;
   jahr_ = j;
}
```

**9.15**

```
std::string Datum::toString() const {
   std::string temp("tt.mm.jjjj");
                // implizite Umwandlung in char
   temp[0] = tag_/10 +'0';
   temp[1] = tag_%10 +'0';
   temp[3] = monat_/10 +'0';
   temp[4] = monat_%10 +'0';
   int pos = 9;                  // letzte Jahresziffer
   int j = jahr_;
   while(j > 0) {
      temp[pos] = j % 10 + '0';  // letzte Ziffer
      j = j/10;                  // letzte Ziffer abtrennen
      --pos;
   }
   return temp;
}
```

**9.16**

```
template<typename T>
void Matrix<T>::swap(Matrix<T>& rhs) { // Verwendung in op*= unten
   super::swap(rhs);
   std::swap(yDim, rhs.yDim);
}

template<typename T>
Matrix<T>& Matrix<T>::operator*=(const Matrix<T>& b) {
   if(spalten() != b.zeilen())
       throw "Falsche Dimension in Matrix*= !";
   Matrix<T> erg(zeilen(), b.spalten());
   for(size_t i = 0; i < zeilen(); ++i) {
      for(size_t j=0; j < b.spalten(); ++j) {
         erg[i][j]= T(0);
         for(size_t k=0; k < spalten(); ++k) {
            erg[i][j] += super::operator[](i)[k] * b[k][j];
         }
      }
   }
   swap(erg);          // *this mit erg vertauschen
   return *this;
}
```

**9.17** Innerhalb des Operators wird der vorhandene Kurzform-Operator für die Multiplikation aufgerufen.

```
template<typename T>
Matrix<T> operator*(Matrix<T> a, const Matrix<T>& b) {
   return a *= b;    // Matrix<T>::operator*=(const Matrix<T>& b)
}
```

Manche mögen wegen des Aufwandes den Aufruf des Kopierkonstruktors bei der Rückgabe von `a` bemängeln. Andererseits ist bei genauer Betrachtung der Aufwand gegenüber dem Gesamtaufwand der Multiplikation für sehr große Matrizen tatsächlich vernachlässigbar: Falls wir der Einfachheit halber große quadratische Matrizen mit $n$ Zeilen und $n$ Spalten betrachten, ist der Aufwand für den Kopierkonstruktor $\propto n^2$, der Aufwand zur Multiplikation jedoch $\propto n^3$. Der tatsächliche Aufwand wird jedoch geringer sein, weil der Compiler den Aufruf des Kopierkonstruktors bei der Rückgabe temporärer Objekte wegoptimieren kann. Alternativ kann man den Konstruktoraufruf durch die in Abschnitt 22.1 ff. gezeigten Techniken selbst wegoptimieren – oder besser: Man setzt gleich eine der fertigen Bibliotheken ein.

**9.18** Die hier vorgestellte Lösung benutzt keine Schleife, die Frage kann also mit »Ja« beantwortet werden. Im Lösungsvorschlag werden die Matrixelemente, die ja vom Typ `mathVektor<T>` sind, mit `v`, einem `mathVektor<T>` initialisiert. `v` wiederum wird beim Aufruf `v.init(Wert);` durch `Vektor<T>::init (const T&)` initialisiert.

```
template<typename T>
void Matrix<T>::init(const T& Wert) {
   mathVektor<T> v(Spalten()); // Hilfsvektor v definieren und initialisieren
   v.init(Wert);
   // Die Matrix ist ein Vektor (von Vektoren), dessen Elemente nun initialisiert werden.
   Vektor<mathVektor<T> >::init(v);
}
```

Eine konzeptionell einfachere und vermutlich verständlichere Lösung wäre eine geschachtelte Schleife über alle Elemente. Ein Laufzeitnachteil ergibt sich nicht, weil die Schleife in der vorgestellten Lösung ebenfalls vorhanden ist, wenn auch versteckt. Überdies erspart sie die Erzeugung des temporären Vektors `v`.

## Kapitel 11

**11.1**

```
Liste(const Liste& liste) // Kopierkonstruktor
: anfang(0), anzahl(0) {
  if(liste.size() > 0) {
    iterator I = liste.begin();
    push_front(*I++); // erstes Element anlegen
    ListElement *letztes = anfang;
    while(I != liste.end()) {
      // Elemente am Ende einfügen, damit die
      // Reihenfolge erhalten bleibt
      letztes->naechstes = new ListElement(*I++, 0);
      letztes = letztes->naechstes;
      ++anzahl;
```

```
    }
  }
}

Liste& operator=(Liste temp) { // Zuweisungsoperator
  std::swap(temp.anfang, anfang);
  std::swap(temp.anzahl, anzahl);
  return *this;
}

~Liste() {    // Destruktor
  clear();
}

iterator erase(iterator p) {
  if(empty()) {
     return iterator(); // leere Liste
  }
  ListElement* zuLoeschen = p.aktuellesElement;
  // Vorgänger suchen
  ListElement* vorgaenger = anfang;
  if(zuLoeschen != anfang) {
    while( vorgaenger->naechstes != zuLoeschen) {
      vorgaenger = vorgaenger->naechstes;
    }
    // Zeiger verbiegen
    vorgaenger->naechstes = zuLoeschen->naechstes;
  }
  else { // am Anfang löschen
    anfang = zuLoeschen->naechstes; // Zeiger verbiegen
  }
  delete zuLoeschen;
  --anzahl;
  return ++p;  // Nachfolger zurückgeben
}

void pop_front() {
  erase(begin());
}

bool empty() const {
  return anfang == 0;
}

size_t size() const {
  return anzahl;
}
void clear() {
  while(!empty()) {
    pop_front();
  }
}
```

## Kapitel 13

**13.1**

```
#ifndef ABLAGE_H
#define ABLAGE_H
#include <boost/thread.hpp>

namespace {
   boost::mutex ausgabeMutex;
}

class Ablage {
public:
   Ablage(int platz)
      : kapazitaet(platz), inhalt(new int[platz]),
        anzahl(0), lesePos(-1), schreibPos(0) {
   }
   ~Ablage() {
      delete [] inhalt;
   }
   int get() {
      boost::unique_lock<boost::mutex> lock(objektMutex);
      while(anzahl == 0) { // leer
         cond.wait(lock);
      }
      --anzahl;
      cond.notify_all();
      lesePos = (lesePos + 1) % kapazitaet;
      return inhalt[lesePos];
   }
   void put(int wert) {
      boost::unique_lock<boost::mutex> lock(objektMutex);
      while(anzahl == kapazitaet) { // voll
         cond.wait(lock);
      }
      inhalt[schreibPos] = wert;
      ++anzahl;
      schreibPos = (schreibPos + 1) % kapazitaet;
      cond.notify_all();
   }
private:
   int kapazitaet;
   int* const inhalt;
   int anzahl;
   // Aufbau als Ringpuffer (FIFO)
   int lesePos;     // letzte gelesene Position
   int schreibPos;  // nächste zu schreibende Position
   boost::mutex objektMutex;
   boost::condition_variable cond;
   // wegen Zeigerattribut inhalt:
   Ablage(const Ablage&);          // Kopie verbieten
   Ablage& operator=(const Ablage&);  // Zuweisung verbieten
};
#endif
```

## Kapitel 24

**24.1** Aus Platzgründen wird die Lösung nicht abgedruckt. Sie ist vollständig in den Beispielen enthalten (siehe Verzeichnis *cppbuch/loesungen/k24/1*).

**24.2** • *heap.t*

```
#ifndef HEAP_T
#define HEAP_T
#include<algorithm>
#include<vector>
#include<utility>
using std::vector;

template<class T, class Compare = std::less<T> >
class Heap {
 public:
   Heap(const Compare& cmp = Compare())
      : anz(0), comp(cmp), v(vector<T>(1)), last(v.begin()) {
   }

   void push(const T& t) {
      if(anz == v.size()) {
         v.resize(anz+100);
         last = v.begin() + anz; // neu bestimmen
      }
      *last = t;
      push_heap(v.begin(), ++last, comp);
      ++anz;
   }

   void pop() {
      pop_heap(v.begin(), last--, comp);
      --anz;
   }

   const T& top() const { return *v.begin(); }

   bool empty() const {  return anz == 0; }

   size_t size() const {  return anz; }

   vector<T> toSortedVector() const {
      vector<T> temp(anz);
      for(size_t i = 0; i < anz; ++i) {
         temp[i] = v[i];
      }
      sort_heap(temp.begin(), temp.end(), comp);
      return temp;
   }
 private:
   size_t anz;
   Compare comp;
   vector<T> v;
```

```
  typename vector<T>::iterator last;
};
#endif
```

• Anwendungsbeispiel *main.cpp*

```
#include<iostream>
#include"heap.t"
using namespace std;

int main() {
  Heap<pair<int, string> > promis;
  promis.push(make_pair(7, "Jack Nicholson"));
  promis.push(make_pair(10, "Bill Clinton"));
  promis.push(make_pair(7, "Thomas Gottschalk"));
  promis.push(make_pair(8, "Brad Pitt"));
  promis.push(make_pair(8, "Peter Jackson"));
  promis.push(pair<int, string>(10, "Tina Turner")); // mal ohne make_pair

  cout << "Sortiert:" << endl;
  vector<pair<int, string> > vs = promis.toSortedVector();
  for(size_t i=0; i < vs.size(); ++i) {
    cout << vs[i].second << ", Priorität "
         << vs[i].first << endl;
  }
  cout << "Leeren:" << endl;
  while(!promis.empty()) {
    cout << promis.top().second << ", Rang "
         << promis.top().first
         << " size=" << promis.size()
         << endl;
    promis.pop();
  }
}
```

**24.3**

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

int main() {
  cout << "Quadratische Gleichung x*x+p*x+q = 0\n";
  cout << "Koeffizienten p, q eingeben:";
  double p, q;
  cin >> p >> q;
  double Diskriminante = p*p/4.0 - q;

  cout << "Lösung :\n";
  if (Diskriminante >= 0.0) {
    double x1 = -p/2.0 + sqrt(Diskriminante);
    double x2 = -p/2.0 - sqrt(Diskriminante);
    cout << "x1= " << x1 << " x2= " << x2 <<endl;
  }
```

```
   else {
      complex<double> ergebnis(-p/2, sqrt(-Diskriminante));
      cout << "x1 = " << ergebnis << endl;
      cout << "x2 = " << conj(ergebnis) << endl;
   }
}
```

## Kapitel 28

**28.1**

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

void bewegen(int n, int a, int b, int c) {
   stack<int> s;
   int t;              // zum Vertauschen der Werte
   // ersten Aufruf transformieren
   while (n > 0) {
      // aktuelle Daten sichern
      s.push(n); s.push(a); s.push(b); s.push(c);
      // Aufruf mit neuen Daten simulieren
      --n; t = b; b = c; c = t;
   }
   // Haupt-Schleife
   while (!s.empty()) {
      c = s.top(); s.pop(); // Daten wiederherstellen
      b = s.top(); s.pop();
      a = s.top(); s.pop();
      n = s.top(); s.pop();
      cout << "Bringe eine Scheibe von " << a
           << " nach " << b << endl;
      --n; t = a; a = c; c = t;
      while (n > 0) {
         // aktuelle Daten sichern
         s.push(n); s.push(a); s.push(b); s.push(c);
         // Aufruf mit neuen Daten simulieren
         --n; t = b; b = c; c = t;
      }
   }
}

int main() {
   cout << "Türme von Hanoi! Anzahl der Scheiben: ";
   int scheiben;
   cin >> scheiben;
   bewegen(scheiben, 1, 2 ,3);
}
```

**28.2**

```
#include<iostream>
#include<utility>
#include<queue>
```

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

int main() {
   priority_queue<pair<int, string> > promis;
   promis.push(make_pair(7, "Jack Nicholson"));
   promis.push(make_pair(10, "Bill Clinton"));
   promis.push(make_pair(7, "Thomas Gottschalk"));
   promis.push(make_pair(8, "Brad Pitt"));
   promis.push(make_pair(8, "Peter Jackson"));
   promis.push(pair<int, string>(10, "Tina Turner"));
   while(!promis.empty()) {
      cout << promis.top().second << ", Priorität "
           << promis.top().first << endl;
      promis.pop();
   }
}
```

**28.3**

```
priority_queue<pair<int, string>,
               deque<pair<int, string> >,
               greater<pair<int, string> > > promis;
```

**28.4**

```
#include<iostream>
#include<utility>
#include<map>
#include<string>
using namespace std;

int main() {
   multimap<int, string, greater<int> > promis;
   // multimap<int, string> promis; // umgekehrte Sortierung
   promis.insert(make_pair(7, "Jack Nicholson"));
   // ... usw. wie in Lösung 28.2
   for(multimap<int, string>::iterator iter = promis.begin();
       iter != promis.end(); ++iter) {
      cout << (*iter).second << ", Priorität "
           << (*iter).first << endl;
   }
}
```

## Kapitel 30

**30.1** `count()` würde nicht funktionieren, weil es ein `pair`-Objekt als Parameter verlangt, es aber in der Aufgabe nur um den Rang, also nur einen Teil der Paar-Kombination geht. Mit `count_if()`, das ein Prädikat verlangt (vgl. Seite 661), ist das Problem zu lösen, weil das Prädikat beliebig gestaltet werden kann. Das Prädikat ist ein Funktionsobjekt und vergleicht nur, ob der Rang der gewünschte ist – der Name wird ignoriert:

```
// gleicherrang.h
#ifndef GLEICHERRANG_H
#define GLEICHERRANG_H
#include<utility>
#include<string>

class GleicherRang {
 public:
   GleicherRang(int r) : rang(r) { }
   bool operator()(const std::pair<int, std::string>& p) const {
      return p.first == rang;
   }
 private:
   int rang;
};
#endif
```

Abgesehen von `#include"gleicherrang.h"` wird das Programm der Lösung 28.4 nur noch um folgendes Stück erweitert:

```
int gesucht = 8;
cout << "Es gibt "
     << count_if(promis.begin(), promis.end(),
                 GleicherRang(gesucht))
     << " Einträge mit Rang " << gesucht << endl;
```

**30.2** `equal_range()` arbeitet nur auf sortierten Containern und braucht daher die Information, welches von zwei Elementen das größere ist. In diesem speziellen Fall wird dabei nur der Rang verglichen. Aus denselben Gründen wie in der vorhergehenden Lösung benötigt `equal_range()` ein Funktionsobjekt, das den Vergleich erledigt:

- *rangvergleich.h*

```
#ifndef RANGVERGLEICH_H
#define RANGVERGLEICH_H
#include<utility>
#include<string>

class Rangvergleich {
public:
   bool operator()(const std::pair<int, std::string>& p1,
                   const std::pair<int, std::string>& p2) const {
      return p1.first > p2.first;
   }
};
#endif
```

Abgesehen vom Inkludieren der Header-Datei wird das Programm der Lösung 28.4 nur noch um folgendes Stück erweitert:

```
// nur der Rang interessiert, siehe rangvergleich.h
pair<int, string> gesuchtesPaar(8, "Dummy");
cout << "Es gibt folgende Einträge mit Rang "
     << gesuchtesPaar.first << ":" << endl;
```

```
pair<multimap<int, string, greater<int> >::iterator,
     multimap<int, string, greater<int> >::iterator>
 bereich = equal_range(promis.begin(), promis.end(),
                       gesuchtesPaar, Rangvergleich());

for(multimap<int, string>::iterator iter = bereich.first;
       iter != bereich.second; ++iter) {
   cout << (*iter).second << endl;
}
```

# A.7 Installation der DVD-Software für Windows

## A.7.1 Installation des Compilers und der Entwicklungsumgebung

Die einfachste Möglichkeit ist die Installation von der DVD, auf der die verwendete Software in komprimierter Form vorliegt. Loggen Sie sich zur Installation als Administrator ein. Anschließend klicken Sie die Datei *installcomp.exe* von der DVD an. Danach melden Sie sich ab und wieder an, damit die Pfadeinstellungen wirksam werden. Bei der Installation werden die folgenden Verzeichnisse angelegt bzw. überschrieben:

- *C:\MinGW*: Dieses Verzeichnis enthält den GNU C++-Compiler 4.5.2 und zugehörige Programmme sowie das Datenbankprogramm SQLite. Auch sind einige Dienstprogramme dabei. Platzbedarf etwa 320 MB.
- *C:\CodeBlocks*: Verzeichnis mit der Entwicklungsumgebung. Platzbedarf etwa 44 MB. Es wird auch eine Verknüpfung auf dem Desktop angelegt.
- *C:\cppbuchincludes*: Hier sind einige für das Compilieren der Beispiele notwendige Dateien abgelegt.

### De-Installation

Die Software wird vom Rechner entfernt, indem die genannten Verzeichnisse gelöscht werden. Die Desktop-Verknüpfung muss manuell gelöscht werden, ebenso die Einträge

```
C:\MinGW\msys\1.0\bin;C:\MinGW\bin;C:\MinGW\lib;C:\Codeblocks
```

in der PATH-Umgebungsvariablen (nicht zwingend).

## A.7.2 Installation der Boost-Bibliothek

Die Installation wird erst ab Kapitel 12 gebraucht, ist also für den Einstieg in die C++-Programmierung nicht notwendig. Die oben beschriebene Installation des Compilers muss abgeschlossen sein. Loggen Sie sich zur als Administrator ein. Anschließend klicken Sie die Datei *installboost.exe* von der DVD an. Danach melden Sie sich ab und wieder an, damit die Pfadeinstellungen wirksam werden. Bei der Installation wird das Verzeichnis

*C:\Boost* angelegt bzw. überschrieben. Platzbedarf etwa 1,2 GB. Während der Installation kann der Platzbedarf wegen der temporären Dateien kurzfristig noch größer werden. Boost kann auch auf einem anderen Laufwerk wie *D:* oder *E:* installiert werden. In diesem Fall muss die Datei *C:\cppbuchincludes\make\include.mak* entsprechend angepasst werden. Wegen der Größe der Bibliothek dauert die Installation einige Minuten.

#### De-Installation

Die Software wird vom Rechner entfernt, indem das Installationsverzeichnis *C:\Boost* gelöscht wird. Der Eintrag `C:\Boost\stage\lib` in der PATH-Umgebungsvariablen muss manuell gelöscht werden (nicht zwingend).

### A.7.3 Installation von Qt

Die Installation wird erst ab Kapitel 14 gebraucht, ist also für den Einstieg in die C++-Programmierung nicht notwendig.

1. Führen Sie die Datei *qt-win-opensource-4.7.2-mingw.exe* (Verzeichnis *win/qt*) aus und folgen Sie den Anweisungen. Die Fragen sollten Sie mit
   ja (Fehlermeldung wegen MinGW ignorieren)
   o (open source) und
   y (Lizenz akzeptieren) beantworten.
2. Ergänzen Sie die PATH–Umgebungsvariable um das Verzeichnis `C:\Qt\4.7.2\bin`. Der Eintrag wird nach Abmelden und Wiederanmelden wirksam.

Als Alternative bietet sich die vollständige Entwicklungsumgebung Qt SDK an, die Sie von *http://qt.nokia.com/* herunterladen können.

#### De-Installation

Die Software wird mit den Windows-Betriebsmitteln vom Rechner entfernt, das heißt, Programmdeinstallation über die Systemsteuerung.

### A.7.4 Codeblocks einrichten

Klicken Sie das Code::Blocks-Symbol auf dem Desktop an. Zuerst wird der Compiler abgefragt. Einfach GNU GCC anklicken und mit OK bestätigen. Die aufpoppenden Tipps- und Skript-Fenster schließen. Für manche Programme werden die Boost-Library und die Dateien im Verzeichnis *C:\cppbuchinclude* benötigt. Deswegen wird Code::Blocks dafür eingerichtet. Ich gehe davon aus, dass Sie Boost durch Entpacken der Datei *boost.tgz* installiert haben. In der Menüleiste »Settings« klicken und »Compiler und debugger« wählen. Unter dem oberen Reiter »Compiler settings« gibt es ein wenig darunter den Reiter »Compiler Flags«. Dort anklicken:

- Produce debugging symbols [-g]
- Enable all compiler warnings [-Wall]
- Have g++ follow the coming C++0x ISO C++ language standard [-std=c++0x]

Dann den Reiter »Search directories« anklicken und darunter den Reiter »Compiler« wählen. Unter der Fläche »Add« (bzw. »Hinzufügen«) anklicken und das Verzeichnis

```
C:/cppbuchincludes/include
```

eintragen – oder mit dem Suchbutton rechts vom Eingabefeld ermitteln. `/home/user` ist ein Platzhalter, bitte für Ihr System anpassen. Als Nächstes auf dieselbe Art

```
C:/Boost
```

eintragen. Im nächsten Schritt werden dem Linker (Reiter »Linker settings«) die Bibliotheken mitgeteilt. Dazu im Feld »Link Libraries« mit dem »Add«- oder »Hinzufügen«-Button die folgenden Dateien eintragen:

```
C:/Boost/stage/lib/libboost_regex-mgw45-mt-1_45.dll.a
C:/Boost/stage/lib/libboost_filesystem-mgw45-mt-1_45.dll.a
C:/Boost/stage/lib/libboost_system-mgw45-mt-1_45.dll.a
C:/Boost/stage/lib/libboost_thread-mgw45-mt-1_45.dll.a
C:/Boost/stage/lib/libboost_unit_test_framework-mgw45-mt-1_45.dll.a
```

Es gibt viel mehr Boost-Libraries, aber nur die angegebenen werden von einigen der Beispiele benötigt. Jetzt unten mit OK bestätigen. Damit sind Sie für die weiteren Beispiele gerüstet, wenn es sich nur um einzelne Dateien handelt.

### Das erste Projekt

Das Starten einer ersten einfachen Programmdatei wird auf Seite 38 beschrieben. Es gibt aber auch Programme, die aus mehreren Dateien bestehen. Für diese Dateien muss ein sogenanntes Projekt angelegt werden. Dazu klicken Sie im »Start here«-Fenster von Code::Blocks auf »Create a new Project«. Im erscheinenden Fenster gibt es eine große Auswahl verschiedener Projekttypen. Für die Beispiele dieses Buchs genügen »Console application« und »Qt4 project«. Bitte wählen Sie »Console application« und dann »Next« und »C++« im erscheinenden Fenster. Als erstes Beispiel wird das Projekt im Verzeichnis *cppbuch/k4/ratio* gewählt. Um das Programm zu erzeugen, könnten Sie direkt in das Verzeichnis gehen und `make` eingeben, aber hier geht es um die Anlage des Projekts in Code::Blocks. Geben Sie als Titel »ratio« an.

In der nächsten Zeile suchen Sie das Verzeichnis *cppbuch/k4* und tragen es ein. Mit »Next« und »Finish« bestätigen. Normalerweise wird automatisch eine Datei *main.cpp* angelegt, aber hier soll die vorhandene genutzt werden. Deswegen die Warnung mit »Nein« beantworten. Links im Fenster können Sie den Bereich »Sources« expandieren und sehen dann *main.cpp*. Ein Doppelklick holt die Datei in den Editor. Das Projekt ist aber noch nicht vollständig; deswegen wird mit Rechtsklick auf den Projekt-Namen »ratio« und »add files...« die Datei *rational.cpp* ausgewählt. Bestätigen Sie mit OK, und Sie sehen auch diese Datei links im Bereich. Mit der Taste F9 können Sie alle Dateien übersetzen und ausführen. Wegen einer beabsichtigten Division durch 0 gibt es einen Abbruch, wie Sie sehen. Wenn Sie in *main.cpp* die letzten Zeilen löschen, beendet sich das Programm regulär.

Letzlich können Sie aber auch mit einem beliebigen ASCII-Editor (Wordpad) die Dateien bearbeiten und mit *make* die Übersetzung anstoßen. Fehlermeldungen werden dann auf der Konsole statt in der IDE angezeigt.

**Weiterführende Informationen**

Weitere Informationen zur Bedienung von Code::Blocks bitte ich, dem Manual (Verzeichnis *win\codeblocks* der DVD) und der Internetseite *http://www.codeblocks.de* zu entnehmen. Dort sind besonders die Rubriken Forum und Wiki interessant.

### A.7.5 Integration von Qt in ein Code::Blocks-Projekt

Die Integration von Qt in ein Code::Blocks-Projekt wird an einem schon vorhandenen Beispiel gezeigt, damit Sie nicht so viel Tipparbeit haben. Die Hinweise sind leicht auf ein neu anzulegendes Projekts übertragbar. Nach dem Start von Code::Blocks »Create a new project« anwählen, im erscheinenden Fenster weiter unten »QT4 project« anklicken. Dann geben Sie den Projekt-Namen »label« (keinen anderen, weil existierende Dateien verwendet werden) und das Verzeichnis *...cppbuch\k14* an, in dem das Projekt gespeichert werden soll. Die Punkte sind durch den Rest des vollständigen Pfadnamens zu ersetzen. Mit »Next« kommen Sie zum nächsten Fenster, in dem Sie angeben, wo Qt installiert ist. Wenn Qt installiert und im Pfad ist, können Sie das Feld `$(#qt4)` so belassen.

Falls Sie Qt nicht von der DVD, sondern manuell installiert haben sollten, kann es sein, dass CodeBlocks sich beschwert, weil es *QtCore4.lib* nicht findet. Dann machen Sie einfach eine Kopie von *C:\Qt\4.7.2\lib\QtCore4.dll*, benennen sie in *QtCore4.lib* um, und bringen sie nach *C:\Qt\4.7.2\lib*.

Mit »Next« beenden Sie die Eingaben. Die nächste Frage beantworten Sie bitte mit *Nein*, damit die vorhandene Datei *main.cpp* nicht überschrieben wird! Nehmen Sie nun die folgenden Einstellungen vor:

- Unter »Project« → »Properties« den Reiter »Project settings« wählen und das Kästchen neben dem Text »This is a custom Makefile« aktivieren. Damit wird zur Compilation das von *qmake* erzeugte Makefile ausgeführt.
- Danach den Reiter »Build target« anklicken und mit »Rename« das Target »Debug« in »debug« umbenennen. Das von *qmake* erzeugte Makefile enthält »debug« als Target.
- Im selben Fenster bei »Output filename« bitte `debug\label.exe` eintragen (also *nicht:* `bin\Debug\label.exe`!). Mit OK bestätigen.
- Unter »Project« → »Build options« ganz links »debug« anklicken und bei »Pre/post build steps« in das obere Feld den Text

  ```
  qmake -project
  qmake
  ```

  eintragen. Die erste Anweisung erzeugt eine Steuerungsdatei für das Projekt, die zweite ein Makefile. Mit OK bestätigen.

Wenn Sie links »Sources« expandieren und auf *main.cpp* klicken, wird die Datei im Editor angezeigt. Übersetzung und Ausführung des Programms können nun wie üblich über die Menüleiste (Build) oder die Taste F9 gestartet werden. Diese Einstellungen müssen für *jedes* Qt-Projekt vorgenommen werden!

## A.7.6 Bei Verzicht auf die automatische Installation

Die folgenden Anweisungen gelten nur für den Fall, dass Sie fertigen Installationsdateien nicht benutzen wollen oder wissen möchten, wie die einzelnen Installationsschritte aussehen. Damit die Programme erreichbar sind, müssen sie im Pfad sein – dies sollten Sie als Nächstes erledigen.

### Pfad einstellen: Windows 7

1. Als Administrator einloggen. Dann »Start« → »Systemsteuerung« → »System und Sicherheit« → »System« und dort den links »Erweiterte Systemeinstellungen« anklicken. Auf der erscheinenden Registerkarte unten »Umgebungsvariablen« anklicken. Dann im Fenster unten bei den Systemvariablen die Variable »Path« wählen und dann »Bearbeiten« anklicken. Den Pfad um die benötigten Verzeichnisse ergänzen. Er sollte am *Anfang* (!) enthalten:

   ```
   C:\MinGW\msys\1.0\bin;C:\MinGW\bin;C:\MinGW\lib;C:\Codeblocks
   ```

   Nach Installation von Boost und Qt kommen hinzu

   ```
   ;C:\Boost\stage\lib;C:\Qt\4.7.2\bin
   ```

   Bearbeiten Sie die Pfadangaben sehr sorgfältig und löschen Sie nicht schon vorhandene Einträge wie etwa `;%SystemRoot%\system32` usw.
   Alternativ kann auch eine in Prozentzeichen eingeschlossene Umgebungsvariable angegeben werden, sofern sie definiert ist, zum Beispiel `%MINGW_HOME%\bin`.
   Danach mit zweimal »OK« beenden.
2. Damit die Änderung wirksam wird, jetzt abmelden und erneut als Administrator anmelden.

### Pfad einstellen: Windows XP

Klicken Sie »Start« → »Systemsteuerung« → »System« und dort den Reiter »Erweitert« an. Auf der erscheinenden Registerkarte unten »Umgebungsvariablen« anklicken. Im Fenster unten bei den Systemvariablen die Variable »Path« und dann »Bearbeiten« anklicken. Dann weiter wie oben bei Windows 7 beschrieben.

### Pfad einstellen: Windows Vista

Als Administrator einloggen. Dann »Start« → »Systemsteuerung« → »System« → »Erweiterte Systemeinstellungen« und dort den Reiter »Erweitert« anklicken. Dann weiter wie oben bei Windows 7 beschrieben.

### Installation des Compilers

Entpacken Sie die Datei *win/mingw/mingw.tgz* von der DVD mit einem geeigneten Programm nach *C:*. Passen Sie den Pfad an wie oben beschrieben.

### Installation der IDE Code::Blocks

Klicken Sie die Datei *codeblocks-10.05-setup.exe* im Verzeichnis *win/codeblocks/* der DVD an und folgen Sie den Anweisungen.

### Installation der Boost-Library

Gehen Sie in das Verzeichnis *win\boost* der DVD. Folgen Sie den Anweisungen in der Datei *INSTALL.txt*.

### Beispieldateien entpacken

Um mit den Beispieldateien zu arbeiten, kopieren Sie die Datei *cppbuch.tgz* aus dem Verzeichnis *win* der DVD in das Verzeichnis *Eigene Dateien* (Windows XP) oder *Dokumente* (Windows 7 oder Windows Vista). Öffnen Sie ein Shell-Fenster und gehen Sie mit `cd`-Befehlen in dieses Verzeichnis. Die Datei wird mit

```
tar xvfz cppbuch.tgz
```

entpackt. Wenn Sie danach zum Beispiel in das Verzeichnis *cppbuch/k1* gehen und das Kommando `make` eintippen, werden alle Beispiele in diesem Verzeichnis übersetzt. Die entstehenden Programme, erkennbar an der Endung *.exe*, werden durch Eingabe des Namens in das Shell-Fenster aufgerufen.

## A.8 Installation der DVD-Software für Linux

### A.8.1 Installation des Compilers

Wenn Sie `g++ --version` in ein Shell-Fenster eintippen und Sie die Meldung »command not found« bekommen, ist der Compiler nicht installiert. Andernfalls sollte er sich mit einer Versionsnummer 4.4.x oder höher melden. Installieren Sie sich den C++-Compiler gegebenenfalls von der Betriebssystem-DVD oder mit dem Software-Update-Tool des Betriebssystems, falls er nicht vorhanden ist. Für einige wenige Beispiele wird jedoch mindestens die Version 4.5 des GNU C++-Compilers benötigt. Falls bereits die Version 4.5 oder neuer auf Ihrem System vorhanden ist, können Sie den Rest des Abschnitts überspringen und direkt zum Abschnitt *Installation von Code::Blocks für Linux* (A.8.3) gehen!

Um einen Konflikt mit dem C++-Compiler des Betriebssystems zu vermeiden, wird er separat installiert.

**Hinweis**

Wenn Ihnen die folgenden Schritte (noch) zu kompliziert erscheinen, lesen Sie einfach unten beim Abschnitt *Installation von Code::Blocks für Linux* (A.8.3) weiter. Sie verzichten damit nur auf den Test einiger Beispiele, die von neueren C++-Techniken Gebrauch machen. Sie können die Installation jedoch zu einem späteren Zeitpunkt nachholen.

Vergewissern Sie sich bitte, ob auf Ihrem System die Pakete GMP Version 4.2 oder höher (*http://gmplib.org/*) und MPFR Version 2.3 oder höher(*http://www.mpfr.org/*) installiert sind. Die Pakete *gmp-devel* und *mpfr-devel* müssen auch vorliegen. Wenn nicht, instal-

lieren Sie diese Pakete mit Ihrem System-Werkzeug zur Softwareinstallation. Falls Ihnen diese Pakete nicht vorliegen, kopieren Sie sich die entsprechenden Dateien aus dem Verzeichnis *linux/gcc* der DVD in das Verzeichnis */usr/local*. Die Pakete werden zunächst entpackt:

Endung *.tar.gz*:

```
tar xvfz XXX.tar.gz
```

XXX steht für den ersten Teil des Dateinamens. Endung *.tar.bz2*:

```
bzip2 -d tar xvfz XXX.tar.bz2
tar xvf XXX.tar
```

Anschließend gehen Sie mit `cd XXX` in das betreffende Verzeichnis geben ein:

```
./configure
make install
ldconfig
```

Anschließend prüfen Sie das Paket MPC (*http://www.multiprecision.org/*) auf dieselbe Weise. Die Versionsnummer muss 0.8 oder größer sein.

Zur Installation des Compilers loggen Sie sich als root ein und kopieren die mit *gcc-* beginnenden Dateien aus dem Verzeichnis *linux/gcc* der DVD in das Verzeichnis */usr/local*. Die Steuerungsdatei zur Installation (Makefile) wird mit den folgenden Befehlen erzeugt (xxx ist ein Platzhalter):

```
bzip2 -d gcc-core-4.5-xxx.tar.bz2
bzip2 -d gcc-g++-4.5-xxx.tar.bz2
tar xvf gcc-core-4.5-xxx.tar
tar xvf gcc-g++-4.5-xxx.tar
cd gcc-4.5-xxx
./configure --prefix=/usr/local/gcc45
```

Der Parameter `prefix` sorgt dafür, dass der Compiler im Verzeichnis */usr/local/gcc45* installiert wird. Wird `prefix` usw. weggelassen, wird der Compiler der neue C++-Systemcompiler, was vielleicht nicht erwünscht ist. Anschließend können Übersetzung und Installation mit

```
make install
```

gestartet werden. Der Prozess kann recht lange dauern, abhängig von der Maschine. Wenn Sie einen Dual- oder Quad-Core-Rechner haben, können Sie den Vorgang erheblich beschleunigen, indem Sie die Option `-j4` für einen Quad-Core-Rechner bzw. `-j2` für einen Dual-Core-Rechner angeben, also etwa

```
make -j4 install
```

Loggen Sie sich aus und melden Sie sich als normaler Benutzer wieder an.

### A.8.2 Installation von Boost

Auf meinem System brach die Installation ab, weil Python nicht gefunden wurde. Nach Installation der Pakete python und python-devel lief alles wie geplant. Loggen Sie sich als root ein und kopieren die Datei *boost_1_45_0.tar.bz2* aus dem Verzeichnis *linux/boost* der DVD in das Verzeichnis */usr/local*. Dort entpacken Sie die Datei mit `tar --bzip2 -xf boost_1_45_0.tar.bz2`

Gehen Sie in das entstandenen Verzeichnis mit `cd boost_1_45_0` und rufen Sie dort

```
/.bootstrap.sh -help
```

auf, um die Optionen zu sehen. In der Regel können Sie die Voreinstellungen beibehalten. Rufen Sie dann

`./bootstrap.sh` auf und anschließend

```
./bjam
```

um die Übersetzung zu starten. Der Vorgang dauert recht lange, ein idealer Moment für eine Kaffeepause. Die erzeugten Libraries werden im Unterverzeichnis *stage/lib* abgelegt. Der nächste Schritt

```
./bjam install
```

bringt unter anderem die Libraries nach */usr/local/lib* und kopiert die Headerdateien nach */usr/include/boost*. Das ist alles.

### A.8.3 Installation von Code::Blocks

Zur Installation unter Linux führen Sie bitte die folgenden Schritte durch:

1. Loggen Sie sich als Administrator (root) ein.
2. Prüfen Sie, ob *wxWidgets* installiert ist. Wenn nicht, installieren Sie es mit dem System-Tool zur Installation. Suchen Sie nach »wx« (unter SuSE 11.x heißen die Pakete wxGTK und wxGTK-devel, weil sie auf der GTK-Bibliothek basieren). Sie finden eine wxWidgets-Version auf der DVD im Verzeichnis *linux/codeblocks*. Andere Portierungen gibt es bei *http://www.wxwidgets.org/downloads/*. Falls notwendig, können Sie wxWidgets mit dem üblichen Verfahren installieren, nachdem Sie die Datei von der DVD nach */usr/local* kopiert haben:

   ```
   bzip2 -d wxGTK-XXX.tar.bz2
   tar xvf wxGTK-XXX.tar
   cd wxGTK-XXX
   ./configure
   make install
   ldconfig
   ```

   *libtool*, *automake* und *zip* müssen ebenfalls installiert sein.
3. Anschließend kopieren Sie die Datei *codeblocks.tgz* zum Beispiel nach */usr/local*. Dann geben Sie ein:

   ```
   cd /usr/local
   tar xvfz codeblocks.tgz
   cd codeblocks
   ./bootstrap
   make
   make install
   ldconfig
   ```

   Die durch `make` angestoßene Übersetzung kann einige Zeit dauern. Nach Abschluss der Installation kann Code::Blocks gestartet werden, indem als Kommando `codeblocks` eingegeben wird.

*Alternative:* Wenn Sie *subversion* installiert haben, können Sie sich die jeweils neueste Version von Code::Blocks herunterladen. Gehen Sie dazu nach */usr/local* und geben dort ein:

```
svn checkout svn://svn.berlios.de/codeblocks/trunk
```

Das entstehende Verzeichnis *trunk* benennen Sie in *codeblocks* um, damit Sie später wissen, was das Verzeichnis enthält. Gehen Sie mit `cd codeblocks` in das Verzeichnis und rufen die folgenden Kommandos auf:

```
./bootstrap
./configure
make
make install
ldconfig
```

### A.8.4 Code::Blocks einrichten

Starten Sie Code::Blocks durch Eingabe von `codeblocks` in einer Konsole. Zuerst wird der Compiler abgefragt. Einfach GNU GCC anklicken und mit OK bestätigen. Die aufpoppenden Tipps- und Skript-Fenster schließen. Für manche Programme werden die Boost-Library und die Dateien im Verzeichnis *cppbuch/include* benötigt. Deswegen wird Code::Blocks dafür eingerichtet. Ich gehe davon aus, dass Sie Boost wie oben beschrieben installiert haben. In der Menüleiste »Settings« klicken und »Compiler und debugger« wählen. Unter dem oberen Reiter »Compiler settings« gibt es ein wenig darunter den Reiter »Compiler Flags«. Dort anklicken:

- Produce debugging symbols [-g]
- Enable all compiler warnings [-Wall]
- Have g++ follow the coming C++0x ISO C++ language standard [-std=c++0x]

Dann den Reiter »Search directories« anklicken und darunter den Reiter »Compiler« wählen. Unter der Fläche »Add« (beziehungsweise »Hinzufügen«) anklicken und das Verzeichnis `/home/user/cppbuch/include` eintragen – oder mit dem Suchbutton rechts vom Eingabefeld ermitteln. `/home/user` ist ein Platzhalter, bitte für Ihr System anpassen. Als Nächstes auf dieselbe Art

```
/usr/local/include
```

eintragen. Im nächsten Schritt werden dem Linker (Reiter »Linker settings«) die Bibliotheken mitgeteilt. Dazu im Feld »Link Libraries« mit dem »Add«- oder »Hinzufügen«-Button die folgenden Dateien eintragen:

```
/usr/local/lib/libboost_regex.so
/usr/local/lib/libboost_filesystem.so
/usr/local/lib/libboost_system.so
/usr/local/lib/libboost_thread.so
/usr/local/lib/libboost_unit_test_framework.so
```

Es gibt viel mehr Boost-Libraries, aber nur die angegebenen werden von einigen der Beispiele benötigt. Jetzt unten mit OK bestätigen. Damit sind Sie für die weiteren Beispiele gerüstet, wenn es sich nur um einzelne Dateien handelt. Wie Sie mit Code::Blocks ein erstes Programm starten können, lesen Sie auf Seite 939.

## A.8.5 Beispieldateien entpacken

Um mit den Beispieldateien zu arbeiten, kopieren Sie die Datei *cppbuch.tgz* aus dem Verzeichnis *linux* der DVD in Ihr Home-Verzeichnis. Öffnen Sie ein Shell-Fenster und gehen Sie mit `cd`-Befehlen in dieses Verzeichnis. Die Datei wird mit

```
tar xvfz cppbuch.tgz
```

entpackt. Wenn Sie danach zum Beispiel in das Verzeichnis *cppbuch/k1* gehen und das Kommando `make` eintippen, werden alle Beispiele in diesem Verzeichnis übersetzt. Die entstehenden Programme, erkennbar an der Endung *.exe*, werden durch Eingabe des Namens in das Shell-Fenster aufgerufen.

## A.8.6 Installation von Qt4

Die Installation hängt von der verwendeten Linux-Distribution ab. Am einfachsten ist die Verwendung eines Linux-Systems, das Qt4 bereits enthält, einschließlich *libqt4-devel*. Bei Suse-Linux geschieht die Installation mit dem Programm Yast2. Nach Neustart des Users ist der Pfad wirksam, und ein Programm kann mit

```
qmake -project
qmake
make
```

übersetzt werden. Wie Sie Qt in ein Code::Blocks-Projekt integrieren, lesen Sie unten in Abschnitt A.8.7.

**Alternative**

Als Alternative bietet sich die vollständige Entwicklungsumgebung Qt SDK an, die Sie von *http://qt.nokia.com/* herunterladen können.

### Lokale Installation von Qt4

Wenn Sie wollen, können Sie eine eigene Installation von Qt vornehmen, wenn Sie zum Beispiel eine neuere Qt-Version ausprobieren wollen, ohne mit der vorinstallierten Qt-Version Ihres Linux-Systems in Konflikt zu geraten. Dazu kopieren Sie die Datei *qt-everywhere-opensource-src-4.7.2.tar.gz* von der DVD in ein beliebiges Verzeichnis – oder Sie laden sich eine aktuellere Version von der Qt-Internetseite (*http://www.qt.nokia.com/*) herunter. Wenn Qt zum Beispiel in */home/user/programme/qt* installiert werden soll, geben Sie Folgendes ein:

```
tar xvfz qt-everywhere-opensource-src-4.7.2.tar.gz
cd qt-everywhere-opensource-src-4.7.2/
./configure -prefix /home/user/programme/qt
```

Geben Sie auf die Fragen `o` für Open Source ein, und `yes`, um die Lizenzbedingungen zu akzeptieren. Dann folgen die Kommandos

```
gmake
gmake install
```

Die Übersetzung dauert *lange*! Wenn Sie einen Dual- oder Quad-Core-Rechner haben, empfiehlt es sich, `gmake -j2` bzw. `gmake -j4` einzugeben. Dann nutzt gmake alle Prozessoren und spart Zeit. Um die Qt-Version möglichst einfach ansprechen zu können, ist es sinnvoll, den Pfad entsprechend festzulegen. Im Fall einer Bash-Shell tragen Sie in die Datei *.bashrc* des Home-Verzeichnisses ein:

```
PATH=.:/home/user/programme/qt/bin:${PATH}
export PATH
```

Nach Aus- und wieder Einloggen ist der Pfad wirksam.

### A.8.7 Integration von Qt in ein Code::Blocks-Projekt

Die Anleitung ähnelt dem Abschnitt A.7.5, aber im Detail zeigen sich doch einige Unterschiede in den Implementierungen von Qt für Windows und Linux, weswegen die 1:1-Anwendung des angegebenen Abschnitts nicht möglich ist.

Die Integration von Qt in ein Code::Blocks-Projekt wird an einem schon vorhandenen Beispiel gezeigt, damit Sie nicht so viel Tipparbeit haben. Die Hinweise sind leicht auf ein neu anzulegendes Projekt übertragbar. Nach dem Start von Code::Blocks Datei → New → Project anwählen, im erscheinenden Fenster weiter unten »QT4 project« anklicken und oben rechts mit »Go« bestätigen.

Dann geben Sie den Projekt-Namen »label« (keinen anderen, weil existierende Dateien verwendet werden!) und das Verzeichnis *...cppbuch\k14* an, in dem das Projekt gespeichert werden soll. Die Punkte sind durch den Rest des vollständigen Pfadnamens zu ersetzen. Mit »Weiter« kommen Sie zum nächsten Fenster, in dem Sie angeben, wo Qt installiert ist. Wenn Qt auf Ihrem System vorhanden ist, ist normalerweise */usr* einzutragen. Wenn das nicht funktioniert, können Sie mit `qmake -query "QT_INSTALL_PREFIX"` den Ort erfragen. Bei einer lokalen Installation nach obiger Anleitung könnte der Ort zum Beispiel */home/user/programme/qt* sein.

Mit »Weiter« und »Fertigstellen« beenden Sie die Eingaben. Die nächste Frage beantworten Sie bitte mit *Nein*, damit die vorhandene Datei *main.cpp* nicht überschrieben wird! Nehmen Sie nun die folgenden Einstellungen vor:

- Unter »Project« → »Properties« den Reiter »Project settings« wählen und das Kästchen neben dem Text »This is a custom Makefile« aktivieren. Damit wird zur Compilation das von *qmake* erzeugte Makefile ausgeführt.
- Danach den Reiter »Build target« anklicken und mit »Rename« das Target »Debug« in »first« umbenennen. Das von *qmake* erzeugte Makefile enthält »first« als Target.
- Im selben Fenster bei »Output filename« nur `label` eintragen (*nicht:* `bin/Debug/label`!). Mit OK bestätigen.
- Unter »Project« → »Build options« ganz links »first« anklicken und bei »Pre/post build steps« in das obere Feld den Text

  ```
  qmake -project
  qmake
  ```

  eintragen. Die erste Anweisung erzeugt eine Steuerungsdatei für das Projekt, die zweite ein Makefile.

- Rechts den Reiter »Make commands« anklicken und in die Zeile »Clean project/target« rechts `$make –f $makefile clean` eintragen (d.h. `$target` löschen). Mit OK bestätigen.

Wenn Sie links »Sources« expandieren und auf *main.cpp* klicken, wird die Datei im Editor angezeigt. Übersetzung und Ausführung des Programms können nun wie üblich über die Menüleiste (Build) oder die Taste F9 gestartet werden. Diese Einstellungen müssen für *jedes* Qt-Projekt vorgenommen werden!

# Glossar



Dieses Glossar enthält Kurzdefinitionen der wichtigsten Begriffe der objektorientierten Programmierung und verwandter Bereiche. Das Glossar ist zum Nachschlagen und zur Wiederauffrischung der Begriffe gedacht, nicht zur Einführung. Entsprechende Textstellen können über das Stichwortverzeichnis gefunden werden.

## Abstrakter Datentyp

Ein Abstrakter Datentyp fasst *Daten* und *die Funktionen*, mit denen die Daten bearbeitet werden dürfen, zusammen. Der Sinn liegt darin, den richtigen Gebrauch der Daten sicherzustellen. Mit »Funktion« ist hier *nicht* die konkrete Implementierung gemeint, das heißt, *wie* die Funktion im Einzelnen auf die Daten wirkt. Zur Benutzung eines Abstrakten Datentyps reicht die Spezifikation der Zugriffsoperation aus. Ferner sind logisch zusammengehörige Dinge an einem Ort konzentriert. Ein Abstrakter Datentyp ist ein → *Typ* zusammen mit → *Datenkapselung*. Eine → *Klasse* in C++ ist ein Abstrakter Datentyp.

## Abstrakte Klasse

Eine abstrakte Klasse ist eine → *Klasse*, von der es keine → *Instanzen* gibt. Abstrakte Klassen definieren → *Schnittstellen*, die durch abgeleitete Klassen implementiert werden müssen.

## Aggregation

Die Aggregation ist ein Spezialfall der → *Assoziation*, der Enthaltensein (Teil-Ganzes-Beziehung) beschreibt. Wenn im Ganzen nur Verweise auf die Teile existieren, sind diese nicht existentiell abhängig vom Ganzen. Andernfalls spricht man auch von Komposition. In diesem Fall werden die Teile zusammen mit dem Ganzen zerstört. Zur Umsetzung in C++ siehe Seite 585.

### Attribut

Attribute beschreiben die Eigenschaften eines Objekts. Der aktuelle Zustand eines Objekts wird durch die Werte der Attribute beschrieben. Zum Beispiel kann *Farbe* ein Attribut sein; ein möglicher Attributwert wäre *rot*.

### Ausnahme

Eine Ausnahme (englisch *exception*) ist die Verletzung der → *Vorbedingung* einer Operation (→ *Methode*) einer Klasse. C++ bietet die Möglichkeit, Ausnahmen zu erkennen und zu behandeln (*exception handling*).

### Assoziation

Die Assoziation ist eine gerichtete Beziehung zwischen Klassen. Sie kann in eine Richtung verweisen (A kennt B, aber nicht umgekehrt) oder bidirektional sein (A kennt B und B kennt A).

### Behälterklasse

Datenstruktur zur Speicherung von Objekten. Beispiele: Array, Liste, Vektor

### Bindung

→ *dynamische Bindung*, → *statische Bindung*

### Botschaft

In der rein objektorientierten Programmierung wird davon ausgegangen, dass ein laufendes Programm(-system) aus einer Menge von Objekten besteht, die miteinander über Botschaften (englisch *messages*) kommunizieren. Ein genauerer Begriff als Botschaft ist *Aufforderung*, weil das empfangende Objekt etwas tun soll. Ein Objekt, das eine Aufforderung erhält, führt eine dazu passende Operation (→ *Methode*) aus, die in der → *Klasse* beschrieben ist.

### Client

Im informationstechnischen Sprachgebrauch heißen Dinge (Objekte, Rechner, ...), die eine Dienstleistung erbringen, *Server*. Die Dienstleistung wird erbracht für einen *Client* (deutsch: Klient, Kunde), der selbst ein Rechner oder Objekt sein kann.

### Container

→ *Behälterklasse*

### Daten

Der Zustand eines Objekts wird durch seine Daten beschrieben. Die Daten sind die Werte der → *Attribute* eines Objekts.

### Datenkapselung

Datenkapselung ist das »Verstecken« der Daten eines Objekts vor direkten Zugriffen. Zugriffe sind nur über die öffentliche → *Schnittstelle* der Datenkapsel (→ *Abstrakter Datentyp*) möglich. Datenbezogene Fehler sind damit leicht lokalisierbar. In C++ wird Datenkapselung mit der Zugriffsspezifikation `private` realisiert.

### Definition

Eine *Definition* liegt vor, wenn *mehr als nur der Name eingeführt wird*, zum Beispiel wenn Speicherplatz angelegt werden muss für Daten oder Code oder die innere Struktur eines Datentyps beschrieben wird, aus der sich der benötigte Speicherplatz ergibt. Weil *auch* ein Name eingeführt wird, ist eine Definition immer auch eine Deklaration. Die Umkehrung gilt nicht.

### Deklaration

Eine *Deklaration* teilt dem Compiler mit, dass eine Funktion (oder eine Variable) mit diesem Aussehen irgendwo definiert ist. Damit kennt er den Namen bereits, wenn er auf einen Aufruf der Funktion stößt, und ist in der Lage, eine Syntaxprüfung vorzunehmen. Eine Deklaration führt einen Namen in ein Programm ein und gibt dem Namen eine Bedeutung. Eine Deklaration kann gleichzeitig eine → *Definition* sein.

### Delegation

Ein Objekt wird durch den Aufruf einer Methode aufgefordert, eine Dienstleistung zu erbringen. Diese Aufforderung kann an ein weiteres Objekt zur Bearbeitung weitergeleitet (= delegiert) werden.

### Distribution

Die Zusammenstellung von Programm(en) samt Dokumentation und anderen Dateien (Bilder, Audiodateien) in einer zur Verteilung geeigneten, in der Regel gepackten Form, wird Distribution genannt.

### Dynamische Bindung

Wenn sich erst während des Programmlaufs ergibt, welche Methode für ein Objekt aufgerufen werden soll, kann das Binden noch nicht zur Compilier- oder zur Link-Zeit erfolgen, sondern eben erst später (englisch *late binding*). In C++ wird im Allgemeinen während der Übersetzung sichergestellt, dass nur zulässige Aufrufe einer Methode möglich sind (Ausnahme: siehe Beispiele zum `dynamic_cast`). In anderen Sprachen, zum Beispiel *Smalltalk*, wird die Überprüfung ausschließlich erst zur Laufzeit vorgenommen, wodurch auf Kosten einiger Aspekte der Programmsicherheit eine größere Flexibilität ermöglicht wird.

### Frühe Bindung

→ *statisches Binden*

### GNU

Das 1984 begonnene GNU-Projekt (*http://www.gnu.org/*) hatte die Entwicklung eines freien Unix-ähnlichen Betriebssystems zum Ziel. Auch wenn dieses Ziel nicht erreicht wurde, ist dabei jede Menge freier Software entstanden, siehe auch → *Open Source*.

### Identität

Ein Objekt besitzt eine *Identität*, die es unterscheidbar macht von einem beliebigen anderen Objekt, selbst wenn beide gleiche Daten enthalten. Die Identität zu einem bestimmten Zeitpunkt wird durch eine eindeutige Position im Speicher gewährleistet; zwei Objekte können niemals dieselbe Adresse haben, es sei denn, ein Objekt ist im anderen enthalten. C++ hat keine Sprachmittel für die Identität. Die Adresse als Identitätsmerkmal gilt nur für ein → *vollständiges Objekt* und dann auch bei Mehrfachvererbung. Falls dies nicht ausreichend sein sollte, kann die Identität durch ein eigens für diesen Zweck vorgesehenes Element des Objekts definiert werden, zum Beispiel durch eine Seriennummer.

### Initialisierung

Wenn ein Objekt *während der Erzeugung* mit Anfangsdaten versehen wird, heißt der Vorgang *Initialisierung*. Die Initialisierung ist die Aufgabe eines Konstruktors. Sie ist von der → *Zuweisung* zu unterscheiden.

### Instanz

Eine Instanz einer Klasse ist eine andere Bezeichnung für ein → *Objekt*. Die Erzeugung eines Objekts wird auch Instanziierung genannt.

### Interface

→ *Schnittstelle*

### Kapselung

→ *Datenkapselung*

### Klasse

Eine Klasse definiert die Merkmale (Daten) und das Verhalten (Operationen, Methoden) einer Menge von Objekten. Eine Klasse ist ein Datentyp, genauer: ein → *Abstrakter Datentyp*. In C++ gilt die Umkehrung (ein Datentyp ist eine Klasse) *nicht*, weil die Grunddatentypen (zum Beispiel `int`) und darauf aufbauende zusammengesetzte Typen (zum Beispiel C-Array) nicht als Klasse implementiert sind. Eine Klasse definiert die Struktur aller nach ihrem Muster erzeugten Objekte, entweder direkt oder indirekt durch → *Vererbung*.

### Klassifikation

Klassifikation ist ein Verfahren, um Gemeinsamkeiten von Dingen herauszufinden und auszudrücken. Von Unterschieden wird abstrahiert. In C++ wird ein Satz gleicher Merkmale und Verhaltensweisen durch die → *Klasse* beschrieben.

### Komplexität

→ *Zeitkomplexität*

### Lexikografischer Vergleich

Ein lexikografischer Vergleich ist ein Vergleich, wie er bei der Sortierung von Begriffen in einem Lexikon verwendet wird. Danach entscheiden die jeweils ersten beiden Elemente zweier zu vergleichender Folgen, welche Folge als kleiner aufgefasst wird. Falls jedoch die jeweils ersten Elemente *gleich* sind, werden die jeweils zweiten Elemente zum Vergleich herangezogen usw. Zum Beispiel würde bei den Zeichenfolgen »Objekt« und »Oberklasse« erst der dritte Buchstabe über die Sortierung entscheiden.

### Linken

Beim *statischen* Linken werden die Bibliotheksmodule zu dem ausführbaren Programm gebunden. Die so erzeugte ausführbare Datei ist dementsprechend größer. *Vorteil:* Sie kann auf einen anderen Computer derselben Bauart und mit demselben Betriebssystemtyp kopiert werden und funktioniert dort wie auf dem Originalsystem. *Nachteil:* Wenn N Programme dieselbe statische Bibliothek benötigen, wird N mal der zugehörige Speicherplatz gebraucht, wenn die Programme gleichzeitig laufen.

Dynamisches Linken heißt, dass Bibliotheksmodule *nicht* in der ausführbaren Datei enthalten sind, sondern erst bei Ausführung des Programms dazugebunden werden. *Vorteil:* Wenn beliebig viele Programme dieselbe dynamische Bibliothek benötigen, wird der zugehörige Speicherplatz nur einmal gebraucht. *Nachteil:* Die ausführbare Datei funktioniert auf einem anderen Computer derselben Bauart und mit demselben Betriebssystemtyp nur, wenn die dynamische Bibliothek dort installiert ist. Wie dynamische und statische Bibliotheken erzeugt werden, lesen Sie in Abschnitt 23.4.

### L-Wert

Ein L-Wert (Links-Wert, (englisch *lvalue*)) ist ein Ausdruck, der im Kontext seines Auftretens als (symbolische) *Adresse* eines Objekts oder einer Funktion aufgefasst werden kann. Ein L-Wert kann auf der linken Seite einer Zuweisung auftreten, daher der Name (siehe Seite 62). Ein L-Wert kann auch auf der rechten Seite einer Zuweisung auftreten, ein R-Wert jedoch *nur* auf der rechten Seite. In der Regel muss ein Objekt, das verändert werden soll, als L-Wert vorliegen. Alle Ausdrücke, die keine L-Werte sind, sind R-Werte (englisch *rvalue*)[1]. Ein R-Wert repräsentiert den Wert eines Objekts, nicht seine Adresse. Dazu gehört die rechte Seite einer Zuweisung, aber auch eine temporäre Kopie, zum Beispiel vom Kopierkonstruktor bei der Rückgabe eines Funktionswerts erzeugt. Von einem L-Wert kann die Adresse ermittelt werden, von einem R-Wert nicht. Einige Beispiele:

- Der Indexoperator (siehe Seite 326) liefert einen L-Wert zurück:

```
a[i] = 5;
```

- Wenn ein L-Wert in einem Kontext auftritt, wo ein R-Wert erwartet wird, wird er in einen R-Wert umgewandelt:

[1] Nach [ISOC++], Abschnitt 3.10, ist die Unterscheidung noch differenzierter. Da gibt es noch die Wertkategorien *glvalue*, *prvalue* und *xvalue*.

```
x = a[i];
```

- Eine binärer Operator liefert einen R-Wert zurück, in diesem Fall ein temporäres Ergebnis, das `c` zugewiesen wird:

```
c = operator+(a, b);
```

### Mehrfachverbung

Eine Klasse kann in C++ von mehr als einer Klasse erben (→ *Vererbung*).

### Methode

Methode ist eine andere Bezeichnung für eine Operation, die auf den Daten eines → *Objekts* ausgeführt werden kann. In C++ heißen Methoden auch Elementfunktionen (englisch *member functions*), um auszudrücken, dass eine Methode ein Element einer Klasse ist. Sie unterscheidet sich von einer normalen Funktion auch dadurch, dass sie Zugriff auf die internen Daten des → *Objekts* hat.

### Nachbedingung

Die Nachbedingung ist die Spezifikation einer → *Methode*, eines Programmschritts oder einer Funktion. Sie beschreibt ausgehend vom Zustand des Programms *vor* Ausführung (→ *Vorbedingung*), welchen Zustand ein Programm *nach* der Ausführung hat.

### MIME (Multipurpose Internet Mail Extension)

erlaubt es, Multimedia-Inhalte binären Formats an E-Mails zu binden. Der Inhaltstyp, oft MIME-Typ genannt, bestimmt die Art der Verarbeitung. Es können bei der IANA [MIME] registrierte Typen oder selbst definierte sein, die der vorgeschriebenen Syntax genügen. Der MIME-Typ wird nicht nur bei E-Mails, sondern auch in anderen Zusammenhängen verwendet (z.B. Browser).

### Nachricht

→ *Botschaft*

### Oberklasse

Es kann verschiedene Klassen geben, die gemeinsame Anteile enthalten. Diese Anteile können »herausgezogen« werden und bilden eine Oberklasse. Die Klassen werden dann als Spezialisierung der Oberklasse aufgefasst, weil sie nur noch die Unterschiede beschreiben. Zum Beispiel haben eine Tanne und eine Eiche die gemeinsame Eigenschaft, ein Baum zu sein mit all seinen Merkmalen. In einer Beschreibung (Klasse) für eine Tanne genügt es, auf die Oberklasse »Baum« zu verweisen (→ *Vererbung*) und nur die Besonderheit »Nadeln« anzugeben. Eine Oberklasse ist eine durch → *Klassifikation* gewonnene Abstraktion in der Form einer *ist-ein*-Beziehung. Ein Tanne »ist ein« Baum – eine Eiche auch. Eine Oberklasse kann selbst wieder von einer weiteren Oberklasse erben. Manchmal wird eine Oberklasse oder die oberste Oberklasse »Basisklasse« ((englisch *base class*)) genannt.

### Objekt

Ein Objekt ist die konkrete Ausprägung des durch eine → *Klasse* definierten Datentyps. Es hat einen inneren Zustand, der durch Attribute in Form von anderen Objekten oder Elementen der in der Programmiersprache vorgegebenen Datentypen dargestellt wird. Der Zustand kann sich durch Aktivitäten des Objekts ändern, also durch Ausführen von Operationen auf Objektdaten. Jedes Objekt hat eine Identität, sodass auch gleiche Objekte unterscheidbar sind. Im Ablauf eines Programms werden Objekte erzeugt (und wieder gelöscht), die aufgrund des Empfangs einer → *Botschaft* hin aktiv werden. Die Menge aller möglichen Botschaften für ein Objekt heißt → *Schnittstelle*.

### Open Source

Bei Open Source-Software sind die Quellen, wie der Name sagt, frei zugänglich. Open Source-Software ist meistens kostenlos. Sie kann modifiziert, kopiert und weitergegeben werden. Allerdings heißt Open Source nicht, dass alles erlaubt ist. Was erlaubt ist, ist nicht einheitlich geregelt, sodass es für verschiedene Open Source-Software unterschiedliche Lizenzen gibt. Die Bekannteste ist unter *http://www.gnu.org/licenses/* erhältliche »GNU General Public License«.

### Operation

→ *Methode*, → *Botschaft*

### POD (plain old data type)

Dieser Begriff kennzeichnete Datentypen, die es auch in der Sprache C geben kann (daher »plain old«), die also ohne C++-Eigenschaften auskommen. Beispiel:

```
struct Punkt_POD {
    int x;
    int y;
};
```

```
struct Punkt_keinPOD {
   int x;
   int y;
   Punkt_keinPOD(int x, int y);
};
```

Die ausführliche Definition finden Sie in [ISOC++], Kapitel 9.

### Polymorphismus

Die Fähigkeit von Programmelementen, sich *zur Laufzeit* auf → *Objekte* verschiedener → *Klassen* beziehen zu können, heißt Polymorphismus. Anders formuliert: Erst zur Laufzeit eines Programms wird die zu dem jeweiligen Objekt passende Realisierung einer Operation ermittelt. In C++ müssen diese Klassen in einer Vererbungsbeziehung stehen (→ *Vererbung*). Mit Polymorphismus eng verknüpft ist der Begriff → *dynamische Bindung*.

### Resource Acquisition Is Initialization (RAII)

Ein Objekt wird durch den Konstruktor initialisiert. RAII ist das Prinzip, Ressourcen durch die Initialisierung (das heißt durch den Konstruktor) zu belegen. Gleichzeitig ist damit die Freigabe der Ressource durch den Destruktor verbunden. Die Vorteile: Die Freigabe geschieht erst am Ende der Lebensdauer des Objekts. Sie erfolgt automatisch auch im Fehlerfall (Exception), ohne dass sie gesondert programmiert werden muss (wegen der

Freigabe durch den Destruktor). Dieses Prinzip wird oft vorteilhaft eingesetzt. Beispiele: automatisches Schließen von Dateien, exception-sichere Beschaffung von Ressourcen mit `shared_ptr` (Seite 567), Synchronisation mit Mutex-Variablen (Seite 426 ff.), Unit-Test-Fixtures (Seite 534), `sentry`-Objekte zur Absicherung von Ein-/Ausgabeoperationen (Seiten 835, 836 und andere).

### R-Wert

→ *L-Wert*

### Schnittstelle

Als (öffentliche) Schnittstelle (englisch *(public) interface*) bezeichnet man die Menge von Aufforderungen, auf die ein → *Objekt* reagieren kann. In C++ werden Schnittstellen durch die → *Deklarationen* der `public`- → *Methoden* beschrieben.

### Server

→ *Client*

### Signatur

Die Signatur besteht aus der Kombination des Funktionsnamens mit der Reihenfolge und den Typen der Parameterliste. Anhand der Signatur kann der Compiler überladene Funktionen erkennen. Manche betrachten auch den Rückgabetyp als Teil der Signatur.

### Spätes Binden

→ *Dynamische Bindung*

### Statische Bindung

Ein Funktionsaufruf muss an eine Folge von auszuführenden Anweisungen gebunden werden. Der Aufruf einer Funktion (oder Methode, Operation) heißt statisch gebunden, wenn bereits der Compiler oder der Binder (Linker) die Funktion einbindet, also *vor dem Programmstart*. Die Typverträglichkeit und Zulässigkeit von Funktionsaufrufen kann damit sehr früh geprüft werden. Siehe auch → *Dynamische Bindung*.

### Subtyp

Ein Subtyp ist ein abgeleiteter → *Typ*, wobei ein Objekt eines Subtyps jederzeit an die Stelle eines Objekts der → *Oberklasse* treten kann. Ein abgeleiteter Typ, der nicht vollständig das Verhalten der Oberklasse zeigt, ist kein echter Subtyp.

### Typ

Ein Typ ist die Menge aller Objekte in einem System, die auf dieselbe Art auf eine Menge von → *Botschaften* reagieren. Diese Objekte haben dieselbe öffentliche → *Schnittstelle*. In C++ wird ein Typ durch eine → *Klasse* beschrieben.

### UML

Die Unified Modeling Language (UML) ist eine weit verbreitete grafische Beschreibungssprache für Klassen, Objekte und noch mehr. Sie wird vornehmlich in der Phase des Softwareentwurfs eingesetzt. Grundlagen und Anwendung sind zum Beispiel in [Oe] beschrieben.

### Unterklasse

Eine Klasse, zu der eine → *Oberklasse* existiert, heißt Unterklasse bezüglich dieser Oberklasse. Wenn ein Objekt der Unterklasse stets an die Stelle eines Oberklassenobjekts treten kann, ist die Unterklasse ein → *Subtyp* der Oberklasse. Eine Unterklasse heißt auch »abgeleitete Klasse« ((englisch *derived class*)).

### Vererbung

Vererbung wird definiert durch eine Beziehung zu einer → *Oberklasse*, um deren Merkmale und Verhaltensweisen zu übernehmen. Eine Klasse »erbt« von Oberklassen, indem die direkten Oberklassen in der Klassendefinition angegeben werden. Gleichzeitig wird damit von allen Oberklassen der Oberklasse geerbt, sofern sie existieren. Der Aufruf einer Operation für ein Objekt lässt nicht erkennen, ob sie der Klasse des Objekts oder einer Oberklasse zuzuordnen ist, also geerbt wurde.

### Vertrag

Eine → *Methode* gewährleistet die Einhaltung ihrer Spezifikation, wenn der Aufrufer die → *Vorbedingung* einhält (zum Beispiel Aufruf der Methode mit korrekten Parametern). Die Methode erfüllt damit einen Vertrag.

### Vollständiges Objekt

Unter einem »vollständigen Objekt« wird ein Objekt verstanden, das nicht als Subobjekt dient, also nicht in einem anderen Objekt durch Vererbung enthalten ist.

### Vorbedingung

Die Vorbedingung beschreibt den Zustand eines Programms, der notwendig ist, um den nächsten Programmschritt korrekt durchführen zu können. Der Programmschritt kann der Aufruf einer → *Methode* sein.

### Zeitkomplexität

Der Begriff Komplexität aus der Informatik gibt an, wie die von einem Algorithmus benötigte Zeit und der benötigte Speicherplatz von der Anzahl der im Container gespeicherten Elemente abhängen. Meistens interessiert nur die benötigte Zeit (Zeitkomplexität). Von den Eigenschaften der Maschine, Betriebssystem und der Programmiersprache wird dabei abstrahiert. Letztere gehen mit einem konstanten Faktor ein. Die Zeitkomplexität von Algorithmen wird üblicherweise in der O-Notation angegeben. Dabei meint $O(n)$, dass der Zeitaufwand proportional zur Anzahl $n$ der Elemente (eines Containers) ist. $O(\log n)$ bedeutet, dass der Zeitaufwand proportional zum Logarithmus von $n$ ist, und $O(1)$, dass der Zeitaufwand konstant bzw. unabhängig von $n$ ist.

### Zustand

Der Zustand eines Objekts ist definiert durch die Menge der Werte seiner → *Attribute*.

### Zuweisung

Eine Zuweisung weist ein Objekt dem anderen zu und ändert damit dessen Wert. Im Unterschied zur → *Initialisierung* muss das zu ändernde Objekt vor der Zuweisung bereits existieren.

### Zusicherung

Eine Zusicherung (englisch *assertion*) ist eine logische Bedingung, die erfüllt sein muss. Zusicherungen dienen der Verifikation von Programmen, das heißt dem Nachweis, dass ein Programm seiner Spezifikation entspricht. → *Vorbedingungen* und → *Nachbedingungen* sind Beispiele für Zusicherungen.

# Literatur-verzeichnis


[Abr] Dave Abrahams: *Want Speed? Pass by Value*. *http://cpp-next.com/archive/2009/08/want-speed-pass-by-value/*

[ALSU] Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullmann: *Compiler*. Pearson 2008

[Alex] Andrei Alexandrescu: *Modern C++ Design*. Addison-Wesley 2001

[asio] Christopher Kohlhoff: *Boost.Asio*. *http://www.boost.org/doc/libs/1_45_0/doc/html/boost_asio.html* oder die lokale Dokumentation Ihres Boost-Systems

[Bal] Heide Balzert: *Lehrbuch der Objektmodellierung*. Spektrum Akademischer Verlag 2004

[Beck] Kent Beck: *Extreme Programming Explained: Embrace Change*. Addison-Wesley 2004

[BeckP] Pete Becker: *The C++ Standard Library Extensions*. Addison-Wesley 2007

[BlSu] Jasmin Blanchette, Mark Summerfield: *C++ GUI-Programming with Qt 4*. Prentice Hall 2008

[boost] Boost-Homepage: *http://www.boost.org/*

[Br] Ulrich Breymann: *Assignment and Polymorphism*, Journal of Object-Oriented Programming 13, No. 11, März 2001, S. 20-24

[CB] Code::Blocks-Homepage *http://www.codeblocks.org/*

[CE] Krzysztof Czarnecki, Ulrich Eisenecker: *Generative Programming*. Addison-Wesley 2000

[CERT] CERT C++ Secure Coding Standard, *https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=637*

[CLR] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: *Introduction to Algorithms*. MIT Press 2009 (auch in deutscher Sprache erhältlich)

[Date] C. J. Date: *An Introduction to Database Systems*. Addison-Wesley 2003

[Ecma] Standard ECMA-262, *http://www.ecma-international.org/publications/standards/Ecma-262.htm*

[ES] Margaret A. Ellis, Bjarne Stroustrup: *The Annotated C++ Reference Manual*. Addison-Wesley 1990

[Fiel] R. Fielding et al.: *Hypertext Transfer Protocol – HTTP/1.1*, *http://www.ietf.org/rfc/rfc2616.txt*, 1999

[Fri] Jeffrey E.F. Friedl: *Mastering Regular Expressions*. 3. Auflage, O'Reilly 2006

[Gamma] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns*. Addison-Wesley 1995

[GCC] Homepage der GNU Compiler Collection: *http://gcc.gnu.org*

[GNU] GNU Autoconf und Verwandte, *http://www.gnu.org/software/autoconf/#family*

[GrJ] Douglas Gregor, Jaakko Järvi: *Variadic Templates for C++0x*, in Journal of Object Technology, vol. 7, no. 2, February 2008, pp. 31-51, *http://www.jot.fm/issues/issue_2008_02/article2/*

[Her] Helmut Herold: *Linux/Unix-Systemprogrammierung*. Addison-Wesley 2004

[HeNy] Mats Henricson, Erik Nyquist: *Programming in C++ – Rules and Recommendations* (auf der DVD vorhanden)

[Hof06] Douglas R. Hofstadter: *Gödel, Escher, Bach. Ein Endloses Geflochtenes Band*. Klett-Cotta 2006

[ISOC] ISO/IEC 9899-201x: *C Standard*, Committee Draft, 16. November 2010. *http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1539.pdf*

[ISOC++] ISO/IEC JTC 1/SC22/WG21: *Programming Language C++*, Final Draft International Standard, 11. April 2011. *http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2011/n3290.pdf*

[JSF] JSF AV C++ Coding Standards, Lockheed Martin 2005, *http://www.research.att.com/~bs/JSF-AV-rules.pdf* (auf der DVD vorhanden)

[KL] Klaus Kreft, Angelika Langer: *Standard C++ IOStreams and Locales*. Addison Wesley 2008

[Lis] Barbara Liskov: *Data Abstraction and Hierarchy*, Addendum to Proc. of the ACM Conference on Object-Orientated Programming Systems, Languages, and Applications (OOPSLA '87). SIGPLAN Notices 23, 5 (May 1988)

[make] GNU make, *http://www.gnu.org/software/make/manual/make.html*

[Mar] Rudolf Marty: *Methodik der Programmierung in Pascal*, Springer 1994

[Mey] Bertrand Meyer: *Touch of Class*. Springer 2009

[Mill] Peter Miller: *Recursive Make Considered Harmful*, *http://miller.emu.id.au/pmiller/books/rmch/*

[MIME] IANA – Internet Assigned Numbers Authority: *MIME Media Types*, *http://www.iana.org/assignments/media-types/index.html*

[Neun] Alexander Neundorf: *Why the KDE project switched to CMake – and how*, *http://lwn.net/Articles/188693/*. Siehe auch *http://www.linuxmagazin.de/heft_abo/ausgaben/2007/02/mal_ausspannen*

[NTP] D. Mills: *RFC 4330: Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI . http://www.ietf.org/rfc/rfc4330.txt*

[Oe] Bernd Oestereich: *Analyse und Design mit UML 2.3*. Oldenbourg 2009

[OON] The Object-Oriented Numerics Page, *http://oonumerics.org/oon/*

[Port] IANA – Internet Assigned Numbers Authority: *PORT NUMBERS*, *http://www.iana.org/assignments/port-numbers*

[Roz] Gennadiy Rozental: *Boost Test Library*. *http://www.boost.org/doc/libs/1_45_0/libs/test/doc/html/index.html* oder die lokale Dokumentation Ihres Boost-Systems

[ScM] Scott Meyers: *Effective C++*, 3. Auflage. Addison-Wesley 2005

[ScMb] Scott Meyers: *Mehr Effectiv C++ programmieren*. Addison-Wesley 1997

[SFP] Ben Collins-Sussmann, Brian W. Fitzpatrick, C. Michael Pilato: *Version Control with Subversion*, *http://svnbook.red-bean.com/* (auf der DVD vorhanden)

[SL] Andreas Spillner, Tilo Linz: *Basiswissen Softwaretest*, Dpunkt 2010.

[SQLite] SQLite Homepage: *http://www.sqlite.org/*

[SSRB] Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann: *Pattern-Oriented Software Architecture – Volume 2, Patterns for Concurrent and Networked Objects*. Wiley 2000

[Str] Bjarne Stroustrup: *The C++-Programming Language*, Addison-Wesley 2000 (auch in deutscher Übersetzung erhältlich)

[Str94] Bjarne Stroustrup: *The Design and Evolution of C++*. Addison-Wesley 1994

[svn] Subversion Homepage: *http://subversion.tigris.org/*

[thread] Anthony Williams: *Thread*. *http://www.boost.org/doc/libs/1_45_0/doc/html/thread.html* oder die lokale Dokumentation Ihres Boost-Systems

[TR1] Draft Technical Report on C++ Library Extensions, *http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1836.pdf*

[Unic] Homepage der Unicode-Organisation, *http://www.unicode.org/*

[Unr] Erwin Unruh, Primzahlenprogramm, *http://www.erwin-unruh.de/primorig.html*

[URI] T. Berners-Lee, R. Fielding, L. Masinter: *Uniform Resource Identifier (URI): Generic Syntax*, *http://www.ietf.org/rfc/rfc3986.txt*

[VaJo] David Vandevoorde, Nicolai M. Josittis: *C++ Templates – The Complete Guide*. Addison-Wesley 2003

[vdL] Peter van der Linden: *Expert C Programming*. SunSoft Press/Prentice Hall 1994

[Ve95] Todd Veldhuizen: *Using C++ template metaprograms. C++ Report* 7, No. 4, May 1995, Seiten 36–43.
Siehe auch »Blitz«-Projekt im Internet: *http://oonumerics.org/blitz/*

[Ve95a] Todd Veldhuizen: *Expression Templates. C++ Report* 7, No. 5, May 1995, Seiten 26–31.

**Hinweis**

Internet-Verweise unterliegen häufig Änderungen. Es kann daher sein, dass die angegebenen Links nach Druck des Buchs nicht mehr alle auffindbar sind. Der letzte Zugriff auf alle Links fand am 4. Juni 2011 statt.

# Register



## Symbole

# A

## B

## C

## D

## E

# F



# G

## H



## I

## J



## K

## L



## M

## N

## O

## P

## Q



## R

## S

# T

## U



## V

## W



## X



## Y

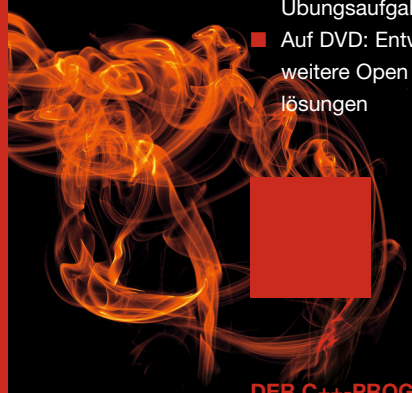


## Z

KOMPETENZ · HANSER · GEWINNT

**ALLES ÜBER C++ – UND NOCH VIEL MEHR //**

- Topaktuell: Entspricht dem neuen ISO-C++-Standard
- Ein Praxisbuch für alle Ansprüche - mehr brauchen Einsteiger und Profis nicht
- Stellt Grundlagen und fortgeschrittene Themen der C++-Programmierung vor und zeigt, welche Unterstützung professionelle Softwareentwickler in der Teamarbeit brauchen
- Enthält über 150 praktische Lösungen für typische Aufgabenstellungen und 85 Übungsaufgaben - natürlich mit Musterlösungen
- Auf DVD: Entwicklungsumgebung und GNU-Compiler für Windows und Linux, weitere Open Source-Software, u.a. Boost und Qt, alle Beispiele und Musterlösungen



**DER C++-PROGRAMMIERER //** Egal ob Sie C++ lernen wollen oder Ihre Kenntnisse in der Softwareentwicklung mit C++ vertiefen, in diesem Buch finden Sie, was Sie brauchen.

C++-Neulinge erhalten eine motivierende Einführung in die Sprache C++. Die vielen Beispiele sind leicht nachzuvollziehen. Klassen und Objekte, Templates, STL und Exceptions sind bald keine Fremdwörter mehr für Sie. Als Profi finden Sie in diesem Buch kurze Einführungen zu Themen wie Thread-Programmierung, Netzwerk-Programmierung mit Sockets und grafische Benutzungsoberflächen. Durch den Einsatz der Boost- und Qt-Libraries wird größtmögliche Portabilität erreicht. Weil Softwareentwicklung nicht nur Programmierung ist, finden Sie hier auch Themen für die professionelle Arbeit im Team, u.a. die Automatisierung der Dokumentation von Programmen, die Versionskontrolle und Werkzeuge zur Projektverwaltung und projektinternen Kommunikation.

Das integrierte »C++-Rezeptbuch« mit mehr als 150 praktischen Lösungen, das sehr umfangreiche Register und das detaillierte Inhaltsverzeichnis machen das Buch zu einem unverzichtbaren Nachschlagewerk für alle, die sich im Studium oder professionell mit der Softwareentwicklung in C++ beschäftigen.

**// »Brillieren kann das Buch vor allem dadurch, dass es dem Leser alle im professionellen Umfeld notwendigen Hilfsmittel zurechtlegt. [...] Die Sprache ist verständlich und kurzweilig, die zahlreichen Codebeispiele sind gut gewählt und erklärt« // iX**

Dr. Ulrich **BREYMANN** ist Professor für Informatik an der Hochschule Bremen. Er engagierte sich im DIN-Arbeitskreis zur Standardisierung von C++ und ist ein renommierter Autor zum Thema C++.

HANSER

www.hanser.de/computer

€ 49,90 [D] | € 51,30 [A]
ISBN 978-3-446-42691-7

9 783446 427549

C++-Programmierer, Einsteiger bis Profis