

# 1

## Es geht los!

Dieses Kapitel behandelt die folgenden Themen:

- Entstehung und Entwicklung der Programmiersprache C++
- Objektorientierte Programmierung - erste Grundlagen
- Wie schreibe ich ein Programm und bringe es zum Laufen?
- Einfache Datentypen und Operationen
- Ablauf innerhalb eines Programms steuern
- Erste Definition eigener Datentypen
- Standarddatentypen `vector` und `string`
- Einfache Ein- und Ausgabe

### 1.1 Historisches

C++ wurde etwa ab 1980 von Bjarne Stroustrup als die Programmiersprache »C with classes« (englisch *C mit Klassen*), die Objektorientierung stark unterstützt, auf der Basis der Programmiersprache C entwickelt. Später wurde die neue Sprache in C++ umbenannt. ++ ist ein Operator der Programmiersprache C, der den Wert einer Größe um 1 erhöht. Insofern spiegelt der Name die Eigenschaft »Nachfolger von C«. 1998 wurde C++ erstmals von der ISO (International Organization for Standardization) und der IEC (International Electrotechnical Commission) standardisiert. Diesem Standard haben sich nationale Standardisierungsgremien wie ANSI (USA) und DIN (Deutschland) angeschlossen. Die Anforderungen an C++ sind gewachsen, auch zeigte sich, dass manches fehlte und anderes überflüssig oder fehlerhaft war. Das C++-Standardkomitee hat kontinuierlich an der Verbesserung von C++ gearbeitet, sodass in Abständen neue Versionen

des Standards herausgegeben wurden. Die Kurznamen sind entsprechend den Jahreszahlen C++03, C++11, C++14 und C++17. C++17 wurde im Juli 2017 von der zuständigen ISO/IEC-Arbeitsgruppe JTC1/SC22/WG21 verabschiedet und bei der ISO zur Veröffentlichung eingereicht.

## 1.2 Objektorientierte Programmierung

Nach üblicher Auffassung heißt Programmieren, einem Rechner mitzuteilen, *was* er tun soll und *wie* es zu tun ist. Ein Programm ist ein in einer Programmiersprache formulierter Algorithmus oder anders ausgedrückt eine Folge von Anweisungen, die der Reihe nach auszuführen sind, ähnlich einem Kochrezept, geschrieben in einer besonderen Sprache, die der Rechner »versteht«. Der Schwerpunkt dieser Betrachtungsweise liegt auf den einzelnen Schritten oder Anweisungen an den Rechner, die zur Lösung einer Aufgabe abzuarbeiten sind.

Was fehlt hier beziehungsweise wird bei dieser Sicht eher stiefmütterlich behandelt? Der Rechner muss »wissen«, *womit* er etwas tun soll. Zum Beispiel soll er

- eine bestimmte Summe Geld von einem Konto auf ein anderes transferieren;
- eine Ampelanlage steuern;
- ein Rechteck auf dem Bildschirm zeichnen.

Häufig, wie in den ersten beiden Fällen, werden Objekte der realen Welt (Konten, Ampelanlage ...) *simuliert*, das heißt im Rechner abgebildet. Die abgebildeten Objekte haben eine *Identität*. Das *Was* und das *Womit* gehören stets zusammen. Beide sind also Eigenschaften eines Objekts und sollen daher nicht getrennt werden. Ein Konto kann schließlich nicht auf Gelb geschaltet werden, und eine Überweisung an eine Ampel ist nicht vorstellbar.

Ein *objektorientiertes Programm* kann man sich als Abbildung von Objekten der realen Welt in Software vorstellen. Die Abbildungen werden selbst wieder Objekte genannt. Klassen sind Beschreibungen von Objekten. Die *objektorientierte Programmierung* berücksichtigt besonders die Kapselung von Daten und den darauf ausführbaren Funktionen sowie die Wiederverwendbarkeit von Software und die Übertragung von Eigenschaften von Klassen auf andere Klassen, Vererbung genannt. Auf die einzelnen Begriffe wird noch eingegangen. Das Motiv hinter der objektorientierten Programmierung ist die rationelle und vor allem ingenieurmäßige Softwareentwicklung.

*Wiederverwendung* heißt, Zeit und Geld zu sparen, indem bekannte Klassen wiederverwendet werden. Das Leitprinzip ist hier, das Rad nicht mehrfach neu zu erfinden! Unter anderem durch den Vererbungsmechanismus kann man Eigenschaften von bekannten Objekten ausnutzen. Zum Beispiel sei *Konto* eine bekannte Objektbeschreibung mit den »Eigenschaften« Inhaber, Kontonummer, Betrag, Dispo-Zinssatz und so weiter. In einem Programm für eine Bank kann nun eine Klasse *Waehrungskonto* entworfen werden, für die alle Eigenschaften von *Konto* übernommen (= geerbt) werden könnten. Zusätzlich wäre nur noch die Eigenschaft »Währung« hinzuzufügen.

Wie Computer können Objekte Anweisungen ausführen. Wir müssen ihnen nur »erzählen«, was sie tun sollen, indem wir ihnen eine *Aufforderung* oder *Anweisung* senden, die in einem Programm formuliert wird. Anstelle der Begriffe »Aufforderung« oder »Anweisung« wird manchmal *Botschaft* (englisch *message*) verwendet, was jedoch den Aufforderungscharakter nicht zur Geltung bringt. Eine gängige Notation (= Schreibweise) für solche Aufforderungen ist *Objektname.Anweisung*(gegebenenfalls *Daten*). Beispiele:

```
dieAmpel.blinken(gelb);
dieAmpel.ausschalten(); // keine Daten notwendig!
dieAmpel.einschalten(gruen);
dasRechteck.zeichnen(position, hoehe, breite); // Daten in cm
dasRechteck.verschieben(5.0); // Daten in cm
```

Die Beispiele geben schon einen Hinweis, dass die Objektorientierung uns ein Hilfsmittel zur Modellierung der realen Welt in die Hand gibt.

## Klassen

Es wird zwischen der *Beschreibung* von Objekten und den *Objekten selbst* unterschieden. Die Beschreibung besteht aus Attributen und Operationen. Attribute bestehen aus einem Namen und Angaben zum Datenformat der Attributwerte. Eine Kontobeschreibung könnte so aussehen:

*Attribute:*

Inhaber: Folge von Buchstaben  
 Kontonummer: Zahl  
 Betrag: Zahl  
 Dispo-Zinssatz in %: Zahl

*Operationen:*

überweisen(Ziel-Kontonummer, Betrag)  
 abheben(Betrag)  
 einzahlen(Betrag)

Eine Aufforderung ist nichts anderes als der Aufruf einer Operation, die auch Methode genannt wird. Ein *tatsächliches* Konto k1 enthält *konkrete* Daten, also Attributwerte, deren Format mit dem der Beschreibung übereinstimmen muss. Die Tabelle zeigt k1 und ein weiteres Konto k2. Beide Konten haben *dieselben* Attribute, aber *verschiedene* Werte für die Attribute.

**Tabelle 1.1:** Attribute und Werte zweier Konten

Attribut	Wert für Konto k1	Wert für Konto k2
Inhaber	Roberts, Julia	Depp, Johnny
Kontonummer	12573001	54688490
Betrag	-200,30 €	1222,88 €
Dispo-Zinssatz	13,75 %	13,75 %

Julia will Johnny 1000 € überweisen. Bei einer Überweisung wird die IBAN (internationale Bankkontonummer) angegeben. Dem Objekt k1 wird also der Auftrag mit den benötigten Daten mitgeteilt: *k1.überweisen("DE25123456000054688490", 1000.00)*.

»DE25123456000054688490« ist die (hier fiktive) IBAN. Die letzten 10 Stellen der IBAN enthalten die Kontonummer, die von der empfangenden Bank extrahiert wird, um den Empfänger zu ermitteln. Anderes Beispiel: Johnny will 22 € abheben. Die Aufforderung wird an k2 gesendet: *k2.abheben(22)*.

Es scheint natürlich etwas merkwürdig, wenn einem Konto ein Auftrag gegeben wird. In der objektorientierten Programmierung werden Objekte als Handelnde aufgefasst, die auf Anforderung selbstständig einen Auftrag ausführen, entfernt vergleichbar einem Sachbearbeiter in einer Firma, der seine eigenen Daten verwaltet und mit anderen Sachbearbeitern kommuniziert, um eine Aufgabe zu lösen.

- Die *Beschreibung* eines tatsächlichen Objekts gibt seine innere *Datenstruktur* und die möglichen *Operationen* oder *Methoden* an, die auf die inneren Daten anwendbar sind.
- Zu *einer* Beschreibung kann es kein, ein oder beliebig viele Objekte geben.

Die Beschreibung eines Objekts in der objektorientierten Programmierung heißt *Klasse*. Die tatsächlichen Objekte heißen auch *Instanzen* einer Klasse.

Auf die inneren Daten eines Objekts nur mithilfe der vordefinierten Methoden zuzugreifen, dient der Sicherheit der Daten und ist ein allgemein anerkanntes Prinzip. Das Prinzip wird *Datenabstraktion* oder Geheimnisprinzip genannt. Der Softwareentwickler, der die Methoden konstruiert hat, weiß ja, wie die Daten konsistent (das heißt widerspruchsfrei) bleiben und welche Aktivitäten mit Datenänderungen verbunden sein müssen. Zum Beispiel muss eine Erhöhung des Kontostands mit einer Gutschrift oder einer Einzahlung verbunden sein. Außerdem wird jeder Buchungsvorgang protokolliert. Es darf nicht möglich sein, dass jemand anderes die Methoden umgeht und direkt und ohne Protokoll seinen eigenen Kontostand erhöht. Wenn Sie die Unterlagen eines Kollegen haben möchten, greifen Sie auch nicht einfach in seinen Schreibtisch, sondern Sie bitten ihn darum, dass er sie Ihnen gibt.

Die hier verwendete Definition einer Klasse als Beschreibung der Eigenschaften einer Menge von Objekten wird im Folgenden beibehalten. Gelegentlich findet man in der Literatur andere Definitionen, auf die hier nicht weiter eingegangen wird. Weitere Informationen zur Objektorientierung sind in Kapitel 3 und in der einschlägigen Literatur zu finden.

## 1.3 Werkzeuge zum Programmieren

Um Programme schreiben und ausführen zu können, brauchen Sie nicht viel: einen Computer mit einem Editor und einem C++-Compiler.

### Der Editor

Ein Editor ist ein Programm, mit dem man Texte schreiben kann. Dabei darf er keine versteckten Sonderzeichen enthalten, weswegen LibreOffice oder Word nicht geeignet sind. Ein für Windows sehr gut geeigneter Texteditor ist *Notepad++*<sup>1</sup>. *Kwrite* läuft unter Linux

<sup>1</sup> <https://notepad-plus-plus.org/>

und Mac OS stellt *TextEdit* zur Verfügung, besser geeignet ist jedoch der Editor von *Xcode*, das Sie dazu jedoch installieren müssen. Um für den Anfang die Einarbeitung in eine Integrierte Entwicklungsumgebung zu sparen, können Sie stattdessen einen einfachen Texteditor benutzen und das Programm in der Konsole (=MS-DOS-Eingabeaufforderung oder PowerShell-Fenster, Linux/MacOS-Terminal)<sup>2</sup> mit den weiter unten gezeigten Kommandos compilieren.

## Der Compiler

Compiler sind die Programme, die Ihren Programmtext in eine für den Computer verarbeitbare Form übersetzen. Von Menschen geschriebener und für Menschen lesbarer Programmtext kann nicht vom Computer »verstanden« werden. Das vom Compiler erzeugte Ergebnis der Übersetzung kann der Computer aber ausführen. Das Erlernen einer Programmiersprache ohne eigenes praktisches Ausprobieren ist kaum sinnvoll. Nutzen Sie daher die Dienste des Compilers möglichst bald anhand der Beispiele – wie, zeigt Ihnen der Abschnitt direkt nach der Vorstellung des ersten Programms. Falls Sie nicht schon einen C++-Compiler oder ein C++-Entwicklungssystem haben, um die Beispiele korrekt zu übersetzen, bietet sich als Alternative zu einem Editor die Benutzung der in Abschnitt 1.5 beschriebenen Entwicklungsumgebungen an. Ein viel verwendeter Compiler ist der GNU C++-Compiler [GCC]. Entwicklungsumgebung und Compiler sind kostenlos erhältlich. Ein Installationsprogramm für einen unter Windows lauffähigen Compiler finden Sie auf der Website zum Buch [CPP]. Hinweise zur Installation finden Sie in Abschnitt A.7 (Seite 957). Bei den meisten Linux-Systemen ist der GNU C++-Compiler enthalten oder kann nachträglich installiert werden. Auf einem Mac mit OS X sollten Sie die Entwicklungsumgebung Xcode mit dem Clang-Compiler<sup>3</sup> installieren (siehe Hinweise dazu ab Seite 961).

## 1.4 Das erste Programm

Sie lernen hier die Entwicklung eines ganz einfachen Programms kennen. Dabei wird Ihnen zunächst das Programm vorgestellt und wenige Seiten weiter erfahren Sie, wie Sie es eingeben und zum Laufen bringen können. Der erste Schritt besteht in der Formulierung der Aufgabe. Sie lautet: »Lies zwei Zahlen a und b von der Tastatur ein. Berechne die Summe beider Zahlen und zeige das Ergebnis auf dem Bildschirm an.« Die Aufgabe ist so einfach, wie sie sich anhört! Im zweiten Schritt wird die Aufgabe in die Teilaufgaben »Eingabe«, »Berechnung« und »Ausgabe« zerlegt:

**Listing 1.1:** Das erste Programm!

```
int main() { // Noch tut dieses Programm nichts!  
    // Lies zwei Zahlen ein
```

<sup>2</sup> Statt des Wortungetüms Eingabeaufforderungsfenster oder des Begriffs Terminal werde ich von nun an in der Regel das Wort *Konsole* verwenden.

<sup>3</sup> [http://clang.llvm.org/get\\_started.html](http://clang.llvm.org/get_started.html)

```

    /* Berechne die Summe beider
       Zahlen */
    // Zeige das Ergebnis auf dem Bildschirm an
}

```

Hier sehen Sie schon ein einfaches C++-Programm. Es bedeuten:

```

int          ganze Zahl zur Rückgabe
main        Schlüsselwort für Hauptprogramm
( )         Innerhalb dieser Klammern können dem Hauptprogramm
            Informationen mitgegeben werden.
{ }         Block
/* ... */   Kommentar, der über mehrere Zeilen gehen kann
// ...     Kommentar bis Zeilenende

```

Ein durch { und } begrenzter *Block* enthält die Anweisungen an den Rechner. Der *Compiler* übersetzt den Programmtext in eine rechnerverständliche Form. Im obigen Programm sind lediglich *Kommentare* enthalten und noch keine Anweisungen an den Computer, so dass unser Programm (noch) nichts tut.

Kommentare werden einschließlich der Kennungen vom Compiler vollständig ignoriert. Ein Kommentar, der mit /\* beginnt, ist mit der ersten \*/-Zeichenkombination beendet, auch wenn er sich über mehrere Zeilen erstreckt. Ein mit // beginnender Kommentar endet am Ende der Zeile. Auch wenn Kommentare vom Compiler ignoriert werden, sind sie doch sinnvoll für den menschlichen Leser eines Programms, um ihm die Anweisungen zu erläutern, zum Beispiel für den Programmierer, der Ihr Nachfolger wird, weil Sie befördert worden sind oder die Firma verlassen haben. Kommentare sind auch wichtig für den Autor eines Programms, der nach einem halben Jahr nicht mehr weiß, warum er gerade diese oder jene komplizierte Anweisung geschrieben hat. Sie sehen:

### Ein Programm ist »nur« ein Text!

- Der Text hat eine Struktur entsprechend den C++-Sprachregeln: Es gibt Wörter wie hier das Schlüsselwort `main` (in C++ werden alle Schlüsselwörter kleingeschrieben). Es gibt weiterhin Zeilen, Satzzeichen und Kommentare.
- Die Bedeutung des Textes wird durch die Zeilenstruktur nicht beeinflusst. Mit \ und folgendem `(ENTER)` ist eine Worttrennung am Zeilenende möglich. Das Zeichen \ wird »Backslash« genannt. Mit dem Symbol `(ENTER)` ist hier und im Folgenden die Betätigung der großen Taste `(↵)` rechts auf der Tastatur gemeint.
- Groß- und Kleinschreibung werden unterschieden! `main()` ist nicht dasselbe wie `Main()`.

Weil die Zeilenstruktur für den Rechner keine Rolle spielt, kann der Programmtext nach Gesichtspunkten der Lesbarkeit gestaltet werden. Im dritten Schritt müssen »nur« noch die Inhalte der Kommentare als C++-Anweisungen formuliert werden. Dabei bleiben die Kommentare zur Dokumentation stehen, wie im Beispielprogramm unten zu sehen ist.

**Listing 1.2:** Summe zweier Zahlen berechnen

```

// cppbuch/k1/summe.cpp
// Hinweis: Alle Programmbeispiele sind von der Internet-Seite zum Buch herunterladbar
// (http://www.cppbuch.de/).

```

```

// Die erste Zeile in den Programmbeispielen gibt den zugehörigen Dateinamen an.
#include<iostream>
using namespace std;

int main() {
    int summe;
    int summand1;
    int summand2;
    // Lies zwei Zahlen ein
    cout << "_Zwei_ganze_Zahlen_eingeben:";
    cin >> summand1 >> summand2;
    /* Berechne die Summe beider Zahlen */
    summe = summand1 + summand2;

    // Zeige das Ergebnis auf dem Bildschirm an
    cout << "Summe=" << summe << '\n';
    return 0;
}

```

Es sind einige neue Worte dazugekommen, die hier kurz erklärt werden. Machen Sie sich keine Sorgen, wenn Sie nicht alles auf Anhieb verstehen! Alles wird im Verlauf des Buchs wieder aufgegriffen und vertieft. Wie das Programm zum Laufen gebracht wird, erfahren Sie nur wenige Seiten danach.

`#include<iostream>` Einbindung der Ein-/Ausgabefunktionen. Diese Zeile muss in jedem Programm stehen, das Eingaben von der Tastatur erwartet oder Ausgaben auf den Bildschirm bringt. Sie können sich vorstellen, dass der Compiler beim Übersetzen des Programms an dieser Stelle erst alle zur Ein- und Ausgabe notwendigen Informationen liest. Details folgen in Abschnitt 2.3.5.

`using namespace std;` Der Namensraum `std` wird benutzt. Schreiben Sie es einfach in jedes Programm an diese Stelle und haben Sie Geduld: Eine genauere Erklärung folgt später (Seiten 64 und 144).

`int main()` `main()` ist das Hauptprogramm (es gibt auch Unterprogramme). Der zu `main()` gehörende Programmcode wird durch die geschweiften Klammern `{` und `}` eingeschlossen. Ein mit `{` und `}` begrenzter Bereich heißt *Block*. Mit `int` ist gemeint, dass das Programm `main()` nach Beendigung eine Zahl vom Typ `int` (= ganze Zahl) an das Betriebssystem zurückgibt. Dazu dient die unten beschriebene `return`-Anweisung. Normalerweise – das heißt bei ordnungsgemäßem Programmablauf – wird die Zahl 0 zurückgegeben. Andere Zahlen können über das Betriebssystem einem folgenden Programm einen Fehler signalisieren.

`int summe;` `int summand1;` `int summand2;` *Deklaration* von Objekten: Mitteilung an den Compiler, der entsprechend Speicherplatz bereitstellt und ab jetzt die Namen `summe`, `summand1` und `summand2` innerhalb des Blocks `{ }` kennt. Es gibt verschiedene Zahlentypen in C++. Mit `int` sind ganze Zahlen gemeint: `summe`, `summand1`, `summand2` sind ganze Zahlen.

;	Ein Semikolon beendet jede Deklaration und jede Anweisung (aber keine Verbundanweisung, siehe weiter unten).
cout	Ausgabe: cout (Abkürzung für <i>character out</i> oder <i>console out</i> ) ist die Standardausgabe. Der Doppelpfeil deutet an, dass alles, was rechts davon steht, zur Ausgabe cout gesendet wird, zum Beispiel cout << summand1;. Wenn mehrere Dinge ausgegeben werden sollen, sind sie durch << zu trennen.
cin	Eingabe: Der Doppelpfeil zeigt hier in Richtung des Objekts, das ja von der Tastatur einen neuen Wert aufnehmen soll. Die Information fließt von der Eingabe cin zum Objekt summand1 beziehungsweise zum Objekt summand2.
=	Zuweisung: Der Variablen auf der linken Seite des Gleichheitszeichens wird das Ergebnis des Ausdrucks auf der rechten Seite zugewiesen.
"Text"	beliebige Zeichenkette, die die Anführungszeichen selbst nicht enthalten darf, weil sie als Anfangs- beziehungsweise Endemarkierung einer Zeichenfolge dienen. Wenn die Zeichenfolge die Anführungszeichen enthalten soll, sind diese als \" zu schreiben: cout << \"C++\" ist der Nachfolger von \"C\"!; erzeugt die Bildschirmausgabe "C++" ist der Nachfolger von "C"!. 
'\n'	die Ausgabe des Zeichens \n bewirkt eine neue Zeile.
return 0;	Unser Programm läuft einwandfrei, es gibt daher 0 zurück. Diese Anweisung darf im main()-Programm fehlen, dann wird automatisch 0 zurückgegeben.

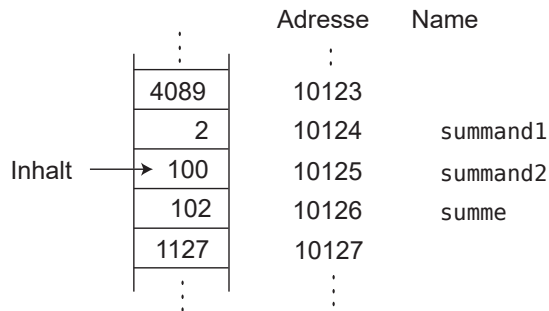
<iostream> ist ein Header. Dieser aus dem Englischen stammende Begriff (head = dt. Kopf) drückt aus, dass Zeilen dieser Art am Anfang eines Programmtextes stehen. Der Begriff wird im Folgenden verwendet, weil es zurzeit keine gängige deutsche Entsprechung gibt. Einen Header mit einem Dateinamen gleichzusetzen, ist meistens richtig, nach dem C++-Standard aber nicht zwingend.

summand1, summand2 und summe sind veränderliche Daten und heißen Variablen. Sie sind Objekte eines vordefinierten Grunddatentyps für ganze Zahlen (int), mit denen die üblichen Ganzzahloperationen wie +, - und = durchgeführt werden können. Der Begriff »Variable« wird für ein veränderliches Objekt gebraucht. Für Variablen gilt:

- Objekte müssen deklariert werden. int summe; ist eine Deklaration, wobei int der *Datentyp* des Objekts summe ist, der die Eigenschaften beschreibt. Entsprechendes gilt für summand1 und summand2. Die Objektamen sind frei wählbar im Rahmen der unten angegebenen Konventionen. Unter *Deklaration* wird verstanden, dass der Name dem Compiler bekannt gemacht wird. Wenn dieser Name danach im Programm versehentlich falsch geschrieben wird, z. B. sume = summand1 + summand2; im Programm auf Seite 35, kennt der Compiler den falschen Namen sume nicht und gibt eine Fehlermeldung aus. Damit dienen Deklarationen der Programmsicherheit.
- Objektamen bezeichnen Bereiche im Speicher des Computers, deren Inhalte verändert werden können. Die Namen sind symbolische Adressen, unter denen der Wert



gefunden wird. Über den Namen kann dann auf den aktuellen Wert zugegriffen werden (siehe Abbildung 1.1).



**Abbildung 1.1:** Speicherbereiche mit Adressen

Der Speicherplatz wird vom Compiler reserviert. Man spricht dann von der *Definition* der Objekte. Definition und Deklaration werden unterschieden, weil es auch Deklarationen ohne gleichzeitige Definition gibt, doch davon später. Zunächst sind die Deklarationen zugleich Definitionen. Abbildung 1.2 zeigt den Ablauf der Erzeugung eines lauffähigen Programms. Ein Programm ist ein Text, von Menschenhand geschrieben (über Programmgeneratoren soll hier nicht gesprochen werden) und dem Rechner unverständlich. Um dieses Programm auszuführen, muss es erst vom Compiler in eine für den Computer verständliche Form übersetzt werden. Der Compiler ist selbst ein Programm, das bereits in maschinenverständlicher Form vorliegt und speziell für diese Übersetzung zuständig ist. Nach Eingabe des Programmtextes mit dem Editor können Sie den Compiler starten.

Ein Programmtext wird auch »Quelltext« (englisch *source code*) genannt. Der Compiler erzeugt aus dem Quelltext den Objektcode, der noch nicht ausführbar ist. Hinter den einfachen Anweisungen `cin >> ...` und `cout << ...` verbergen sich eine Reihe von Aktivitäten wie die Abfrage der Tastatur und die Ansteuerung des Bildschirms, die nicht speziell programmiert werden müssen, weil sie schon in vorübersetzter Form in Bibliotheksdateien vorliegen. Die Aufrufe dieser Aktivitäten im Programm müssen mit den dafür vorgesehenen Algorithmen in den Bibliotheksdateien zusammengebunden werden, eine Aufgabe, die der *Linker* übernimmt, auch *Binder* genannt. Der Linker bindet Ihren Objektcode mit dem Objektcode der Bibliotheksdateien zusammen und erzeugt daraus ein ausführbares Programm, das nun gestartet werden kann. Der Aufruf des Programms bewirkt, dass der *Lader*, eine Funktion des Betriebssystems, das Programm in den Rechnerpeicher lädt und startet. Diese Schritte werden stets ausgeführt, auch wenn sie in den Programmentwicklungsumgebungen verborgen ablaufen. Bibliotheksmodule können auch während der Programmausführung geladen werden (nicht im Bild dargestellt). Weitere Details werden in Abschnitt 2.3 erläutert.

### Wie bekomme ich ein Programm zum Laufen?

Nachdem Sie den Programmtext mit einem Editor geschrieben haben, kann er übersetzt (compiliert) und ausgeführt werden. Integrierte Entwicklungsumgebungen (englisch *Integrated Development Environment, IDE*) haben einen speziell auf Programmierzwecke zugeschnittenen Editor, der darüber hinaus auf Tastendruck oder Mausklick die Überset-