



EUROPA-FACHBUCHREIHE  
Programmierung für die  
IT-Ausbildung

## C++ für IT-Berufe

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

int main()
{
    cout << "Informationsteil:" << endl;

    cout << " - Strukturierte Programmierung
              mit C++" << endl;

    cout << " - Objektorientierte Programmierung
              mit C++" << endl;

    cout << " - Windows-Dektop-Programmierung
              mit C++" << endl;

    cout << endl;
    cout << "Aufgabenpool" << endl;
    cout << endl;
    cout << "Lernsituationen" << endl;
    cout << endl;
    cout << endl;
    return 0;
}
```

### 3. Auflage

VERLAG EUROPA-LEHRMITTEL · Nourney, Vollmer GmbH & Co. KG  
Düsselberger Str. 23 42781 Haan-Gruiten

Europa-Nr.: 85498

**Verfasser:**

Dirk Hardy, 46049 Oberhausen

3. Auflage 2018

Druck 5 4 3 2

Alle Drucke derselben Auflage sind parallel einsetzbar, da sie bis auf die Korrektur von Druckfehlern identisch sind.

ISBN 978-3-8085-8545-0

Alle Rechte vorbehalten. Das Werk ist urheberrechtlich geschützt. Jede Verwertung außerhalb der gesetzlich geregelten Fälle muss vom Verlag schriftlich genehmigt werden.

© 2018 by Verlag Europa-Lehrmittel, Nourney, Vollmer GmbH & Co. KG, 42781 Haan-Gruiten  
[www.europa-lehrmittel.de](http://www.europa-lehrmittel.de)

Satz: Reemers Publishing Services GmbH, Krefeld

Umschlag: braunwerbeagentur, 42477 Radevormwald

Umschlagfotos: matttilda-fotolia.com, Gina Sanders-fotolia.com

Druck: Plump Druck & Medien GmbH, 53619 Rheinbreitbach

## Vorwort

Die Entwicklung von Anwendungen geht heutzutage weit über das reine Programmieren hinaus. Die Anforderungen an den Fachinformatiker bzw. Informatiker mit der Fachrichtung Anwendungsentwicklung sind sehr komplex geworden. Es reicht nicht mehr aus, mit einer speziellen Programmiersprache zu arbeiten. Vielmehr sind umfangreiche Kenntnisse in mehreren Programmiersprachen, in Datenbankmanagementsystemen und weiteren Gebieten wie der Client-Server-Programmierung, Web-Programmierung oder auch App-Programmierung nötig. Das Gemeinsame dieser verschiedenen Aspekte ist die Objektorientierung. Die modernen Programmiersprachen sind objektorientiert, die Datenbankmanagementsysteme werden in Zukunft verstärkt objektorientiert sein. Mischformen wie objektrelationale Datenbanken sind schon länger im Einsatz. Eine gute Grundlage für diese Anforderungen ist die Erlernung der Sprache C++. Die Sprache ist plattformunabhängig und objektorientiert. Sie ist schwerer zu erlernen als andere Sprachen, bietet allerdings dafür einige Vorteile:

- Geschwindigkeit
- Weite Verbreitung
- Sehr viele Tools und Bibliotheken
- Compiler und Entwicklungswerkzeuge auf verschiedenen Plattformen
- Grundlage vieler weiterer Sprachen wie Java, Javascript, Perl oder auch PHP und C#

Die letzte ISO-Standardisierung der Sprache C++ war 2017. Damit verfügt C++ über viele neue Konzepte, die in anderen modernen Programmiersprachen (wie C#) bereits umgesetzt worden sind. Die vorliegende dritte Auflage dieses Buches nimmt diese Neuerungen in einem eigenen Kapitel auf und geht dann auch intensiver auf die STL (Standard Template Library) ein, die mit den Standards C++11, C++14 und C++17 weiter ausgebaut wurde.

Den Lesern dieses Buches bleibt viel Erfolg bei der Erlernung von C++ zu wünschen und mit einem Zitat des Erfinders von C++ (Bjarne Stroustrup) zu schließen: „Die Frage *Wie schreibt man ein gutes C++ -Programm?* hat sehr viel Ähnlichkeit mit der Frage *Wie schreibt man gute englische Prosa?* Darauf gibt es zwei Antworten: Wisse, was Du sagen willst und übe. Halte Dich an gute Vorbilder. Beide Antworten gelten für C++ ebenso wie für Englisch – und sind ebenso schwer zu befolgen.“

Für Anregungen und Kritik zu diesem Buch sind wir Ihnen dankbar (gerne auch per E-Mail).

Dirk Hardy

E-Mail: [Hardy@DirkHardy.de](mailto:Hardy@DirkHardy.de)

Im Oktober 2018

Verlag Europa-Lehrmittel

E-Mail: [Info@Europa-Lehrmittel.de](mailto:Info@Europa-Lehrmittel.de)

## Aufbau des Buches

Das vorliegende Buch möchte die Sprache C++ möglichst anschaulich, praxis- und unterrichtsnah vermitteln. Damit verfolgt dieses Buch einen **praktischen Ansatz**. Es ist die Ansicht des Autors, dass gerade in der schulischen Ausbildung der Zugang zu den komplexen Themen der Programmierung verstärkt durch anschauliche und praktische Umsetzung vorbereitet werden muss. Anschließend können allgemeine und komplexe Aspekte der Programmierung oder auch der Softwareentwicklung besser verstanden und umgesetzt werden.

Das Buch ist in **fünf Teile** getrennt. Die ersten drei Teile des Buches dienen als **Informationsteil** und bieten eine systematische Einführung in die **strukturierte Programmierung** als auch in die **objektorientierte Programmierung mit C++**. Abgerundet wird diese Einführung mit einem **Einstieg in die klassische Windows-Programmierung**, die eine hervorragende Grundlage für alle GUI-Programmierungen bietet.

Der vierte Teil des Buches ist eine **Sammlung von Übungsaufgaben**. Nach der Erarbeitung der entsprechenden Kenntnisse aus dem Informationsteil können die Aufgaben aus diesem Teil zur weiteren Auseinandersetzung mit den Themen dienen und durch verschiedene Schwierigkeitsgrade auch die Differenzierung im Unterricht ermöglichen.

Der **fünfte Teil** des Buches beinhaltet **Lernsituationen** basierend auf dem Lernfeld Entwickeln und Bereitstellen von Anwendungssystemen aus dem Rahmenlehrplan für die IT-Berufe (speziell Fachinformatiker-Anwendungsentwicklung). Lernsituationen konkretisieren sich aus den Lernfeldern und sollen im Idealfall vollständige Handlungen darstellen (Planen, Durchführen, Kontrollieren). Aus diesem Grund werden die Lernsituationen so angelegt, dass neben einer Planungsphase nicht nur die Durchführung (Implementation des Programms) im Blickpunkt steht, sondern auch geeignete Testverfahren zur Kontrolle des Programms bzw. des Entwicklungsprozesses in die Betrachtung einbezogen werden. Die Lernsituationen können aber auch als **Projektideen** verstanden werden.

Das Buch ist für alle berufsbezogenen Ausbildungsgänge im IT-Bereich konzipiert. Durch die differenzierten Aufgabenstellungen kann es in allen IT-Berufen (speziell Fachinformatiker), aber auch von den informationstechnischen Assistenten genutzt werden.

Als Entwicklungswerkzeug wird in diesem Buch die **Community- Edition Visual Studio 2017** von Microsoft genutzt. Diese Entwicklungsumgebung ist kostenfrei als Download im Internet verfügbar.

Dirk Hardy

Oktober 2018

<b>Vorwort .....</b>	<b>3</b>
<b>Aufbau des Buches .....</b>	<b>4</b>
<b>Teil 1 Strukturierte Programmierung mit C++ .....</b>	<b>11</b>
<b>1 Einführung in C++ .....</b>	<b>13</b>
1.1 Historische Entwicklung der Sprache C++.....	13
1.1.1 Von C zu C++.....	13
1.1.2 Prozedurale, strukturierte und objektorientierte Programmierung.....	13
1.1.3 Kleiner Stammbaum der Programmiersprachen .....	15
1.2 Bestandteile eines C++-Programms .....	15
1.2.1 Was ist ein Programm?.....	15
1.2.2 C++-Quellcode.....	15
1.2.3 Grundsätzlicher Aufbau eines C++-Programmes .....	16
1.3 Compiler, Linker und Bibliotheken.....	17
1.3.1 Der Compiler .....	17
1.3.2 Der Linker .....	17
1.3.3 Bibliotheken.....	18
1.3.4 Schematischer Ablauf einer Programmerstellung .....	18
<b>2 Das erste C++-Programm.....</b>	<b>20</b>
2.1 Ausgabe auf dem Bildschirm .....	20
2.1.1 Ein C++-Projekt in Visual Studio anlegen.....	20
2.1.3 Das erste Programm.....	23
2.1.4 Erste Ausgabe auf dem Bildschirm .....	23
2.2 Grundlegende Konventionen in C++ .....	24
2.2.1 Schlüsselworte in C++ .....	24
2.2.2 Bezeichner (Namen) in C++ .....	25
2.2.3 Trennzeichen.....	25
2.2.4 Kommentare .....	25
2.3 Datentypen und Variablen.....	26
2.3.1 Variablen in C++.....	26
2.3.2 Elementare Datentypen.....	27
2.3.3 Operationen auf den elementaren Datentypen.....	29
2.3.4 Anwendungsbeispiel von Variablen .....	30
<b>3 Ein- und Ausgabe in C++ .....</b>	<b>32</b>
3.1 Ausgabe mit cout.....	32
3.1.1 Das Objekt cout .....	32
3.1.2 Ausgabe von Sonderzeichen.....	33
3.1.3 Manipulatoren .....	34
3.2 Eingabe mit cin .....	35
3.2.1 Das Objekt cin .....	35
3.2.2 Der Streamstatus.....	36
<b>4 Operatoren in C++ .....</b>	<b>38</b>
4.1 Arithmetische Operatoren .....	38
4.1.1 Elementare Datentypen und ihre arithmetischen Operatoren .....	38
4.1.2 Der Modulo-Operator .....	39
4.1.3 Inkrement- und Dekrementoperatoren.....	39
4.2 Relationale und logische Operatoren.....	40
4.2.1 Relationale Operatoren.....	40
4.2.2 Logische Operatoren.....	41
4.3 Bit-Operatoren und weitere Operatoren .....	42
4.3.1 Bit-Operatoren .....	42
4.3.2 Die Bit-Schiebeoperatoren << und >>.....	43
4.3.3 Typumwandlung mit cast-Operatoren.....	44
4.3.4 Der sizeof-Operator .....	44
4.3.5 Zuweisung und gekoppelte Zuweisung .....	45
4.4 Prioritäten von Operatoren .....	45
4.4.1 Rang von Operatoren.....	45

---

<b>5</b>	<b>Selektion und Iteration .....</b>	<b>48</b>
5.1	Die Selektion .....	48
5.1.1	Darstellung der Selektion mit einem Programmablaufplan .....	48
5.1.2	Die einseitige Selektion mit der if-Anweisung .....	49
5.1.3	Die zweiseitige Selektion mit der if-else-Anweisung .....	49
5.1.4	Verschachtelte Selektionen mit if und if-else .....	50
5.1.5	Mehrfachselektion mit switch.....	51
5.2	Kopf-, fuß- und zählergesteuerte Iterationen .....	53
5.2.1	Die do-while-Schleife.....	54
5.2.2	Die while-Schleife.....	55
5.2.3	Die for-Schleife.....	55
5.2.4	Abbruch und Sprung in einer Schleife .....	57
<b>6</b>	<b>Funktionen in C++ .....</b>	<b>58</b>
6.1	Entwicklung des Funktionsbegriffs.....	58
6.1.1	Wiederkehrende Programmabschnitte.....	58
6.1.2	Übergabe von Werten .....	59
6.1.3	Rückgabe eines Wertes .....	60
6.1.4	Funktionen in Funktionen aufrufen.....	61
6.1.5	Zusammenfassung der Aspekte aus 6.1 .....	62
6.2	Aufbau der Funktionen in C++ .....	62
6.2.1	Deklaration einer Funktion .....	62
6.2.2	Definition einer Funktion.....	63
6.2.3	Lokale und globale Variablen.....	65
6.2.4	Call by value .....	66
6.2.5	Überladen von Funktionen .....	67
6.2.6	Default-Argumente für Funktionen.....	68
6.2.7	Rekursive Funktionen.....	68
6.3	Modularer Programmaufbau .....	70
6.3.1	Schnittstelle und Implementation.....	71
6.3.2	Umsetzung in C++ .....	71
6.3.3	Namensräume .....	72
6.3.4	Der Präprozessor .....	74
6.3.5	Regeln zur modularen Programmgestaltung .....	76
<b>7</b>	<b>Arrays .....</b>	<b>77</b>
7.1	Ein- und mehrdimensionale Arrays .....	78
7.1.1	Eindimensionale Arrays.....	78
7.1.2	Mehrdimensionale Arrays .....	80
7.1.3	Übergabe von Arrays an Funktionen.....	81
7.2	Zeichenketten in C++ .....	83
7.2.1	Arrays vom Typ char .....	84
7.2.2	Funktionen zur Zeichenkettenbearbeitung .....	85
7.3	Sortieren von Arrays .....	86
7.3.1	Sortieren durch Auswahl .....	87
7.3.2	Der Bubblesort .....	89
<b>8</b>	<b>Zeiger .....</b>	<b>92</b>
8.1	Zeigervariablen .....	92
8.1.1	Deklaration eines Zeigers.....	92
8.1.2	Der Adressoperator .....	92
8.1.3	Der Dereferenzierungsoperator .....	93
8.2	Anwendungen von Zeigervariablen .....	94
8.2.1	Der call by reference .....	94
8.2.2	Zeiger und Arrays.....	95
8.2.3	Zeigerarithmetik.....	96
8.2.4	Zeiger auf Funktionen .....	98
8.2.5	Arrays von Zeigern .....	100
8.2.6	Dynamische Speicherreservierung .....	100
8.3	Die Referenz.....	104
8.3.1	Der Referenzoperator .....	104
8.3.2	Anwendung des Referenzoperators .....	105

<b>9</b>	<b>Strukturen .....</b>	<b>106</b>
9.1	Die Struktur in C++ .....	106
9.1.1	Deklaration einer Struktur .....	106
9.1.2	Zugriff mit Operatoren.....	107
9.1.3	Strukturen in Strukturen.....	108
9.1.4	Arrays von Strukturen .....	109
9.2	Höhere Datenstrukturen.....	110
9.2.1	Die doppelt verkettete Liste .....	111
9.2.2	Der Binärbaum.....	113
<b>Teil 2 Objektorientierte Programmierung mit C++ .....</b>		<b>116</b>
<b>10</b>	<b>Das Klassenkonzept in C++ .....</b>	<b>117</b>
10.1	Die Klasse in C++ .....	119
10.1.1	Aufbau einer Klasse in C++ .....	119
10.1.2	Die Konstruktoren einer Klasse .....	121
10.1.3	Der Destruktor einer Klasse.....	124
10.1.4	Get- und Set-Methoden .....	125
10.2	Dynamische Speicherreservierung in Klassen .....	127
10.2.1	Die Klasse CKette .....	127
10.2.2	Call by value und der Copy-Konstruktor .....	129
10.3	Weitere Elemente einer Klasse .....	131
10.3.1	Der this-Zeiger .....	131
10.3.2	Statische Klassenelemente .....	132
10.4	Deklaration und Implementation bei Klassen.....	133
10.4.1	Header- und cpp-Datei.....	133
<b>11</b>	<b>Dateioperationen .....</b>	<b>134</b>
11.1	Ein- und Ausgabeströme .....	135
11.1.1	Eine Datei im Textmodus öffnen.....	135
11.1.2	Fehler bei Dateioperationen .....	137
11.1.3	Methoden der FileStream-Klassen.....	138
11.1.4	Eine Datei im Binärmodus öffnen .....	141
11.2	Wahlfreier Zugriff in Dateien .....	142
11.2.1	Positionieren des Dateizeigers .....	142
11.2.2	Lesen der Dateizeiger-Position.....	144
<b>12</b>	<b>Das Überladen von Operatoren .....</b>	<b>145</b>
12.1	Globale überladene Operator-Funktion .....	145
12.1.1	Die globale Operator-Funktion .....	145
12.1.2	Addition von Zeichenketten .....	146
12.1.3	Weitere Beispiele für globale Operator-Funktionen .....	148
12.2	Überladene Operatorfunktion als Methode .....	149
12.2.1	Überladener Operator als Methode .....	149
12.2.2	Addition von Zeichenketten .....	150
12.2.3	Weitere Beispiele für überladene Operatoren als Methoden.....	150
12.3	Überladen der Ein- und Ausgabeoperatoren.....	151
12.3.1	Überladene Operatoren der iostream-Klassen.....	151
12.3.2	Überladen des Eingabeoperators für eigene Klassen .....	152
12.3.3	Überladen des Ausgabeoperators für eigene Klassen.....	153
<b>13</b>	<b>Vererbungskonzept in C++ .....</b>	<b>154</b>
13.1	Die einfache Vererbung.....	154
13.1.1	Umsetzung einer einfachen Vererbung in C++.....	155
13.1.2	Attribute als protected deklarieren.....	156
13.1.3	Aufruf der Basisklassenkonstruktoren .....	156
13.1.4	Weitere Formen der Vererbung .....	157
13.2	Die Mehrfachvererbung .....	159
13.2.1	Umsetzung der Mehrfachvererbung in C++.....	159
13.2.2	Virtuelle Vererbung .....	160
13.2.3	Aufruf der Basisklassenkonstruktoren .....	160

---

<b>14</b>	<b>Polymorphismus und virtuelle Methoden .....</b>	<b>161</b>
14.1	Zuweisungen innerhalb einer Vererbungshierarchie.....	161
14.1.1	Zuweisung von Objekten .....	161
14.1.2	Zeiger auf Basisklassen .....	162
14.2	Polymorphismus .....	162
14.2.1	Virtuelle Methoden .....	163
14.2.2	Regeln im Umgang mit virtuellen Methoden.....	164
14.2.3	Arrays von Basisklassenzeilern.....	164
14.2.4	Abstrakte Basisklassen.....	165
<b>15</b>	<b>Fortgeschrittene Programmierung in C++ .....</b>	<b>166</b>
15.1	Templates .....	166
15.1.1	Funktionen-Templates .....	166
15.1.2	Klassen-Templates .....	167
15.2	Ausnahmen – Exceptions.....	168
15.2.1	Versuchen und Werfen – try und throw .....	169
15.2.2	Auffangen – catch .....	169
15.3	Die C++-Standardbibliothek .....	172
15.3.1	Die Klasse string .....	172
15.3.2	Intelligente Zeiger .....	175
15.3.3	Containerklassen .....	177
15.3.4	Algorithmen.....	179
15.3.5	Lambda-Ausdrücke.....	181
15.4	Nützliche Zusatzfunktionalitäten .....	183
15.4.1	Gelöschte Funktionen / Methoden .....	183
15.4.2	Methoden mit override und final kennzeichnen .....	183
15.4.3	Methoden mit const kennzeichnen .....	184
<b>Teil 3 Windows-Desktop-Programmierung mit C++ .....</b>		<b>186</b>
<b>16</b>	<b>Grundlagen der Windows-Desktop-Programmierung .....</b>	<b>187</b>
16.1	Grundlagen der Windows-Programmierung .....	187
16.1.1	Grundkonzept der GUI-Progammierung.....	187
16.1.2	Aufbau einer Windows-Desktop-Anwendung und Nachrichtenbearbeitung.....	188
16.1.3	Eine Windows-Desktopanwendung anlegen.....	188
16.1.4	Die Quellcode-Dateien der Windows-Desktopanwendung.....	189
16.1.5	Die erste Windows-Desktopanwendung .....	192
16.2	Nachrichtenbearbeitung .....	197
16.2.1	Die Nachricht WM_PAINT .....	197
16.2.2	Die Nachrichten WM_CREATE und WM_SIZE .....	199
16.2.3	Tastatur- und Maus-Nachrichten .....	200
16.2.4	Die Nachricht WM_COMMAND .....	202
16.3	Text- und Grafikausgabe .....	202
16.3.1	Einfache Textausgabe mit DrawText .....	202
16.3.2	Texte positionieren mit TextOut .....	204
16.3.3	Punkte und Linien zeichnen .....	205
16.3.4	Rechtecke und Ellipsen zeichnen .....	206
<b>17</b>	<b>Fortgeschrittene Windows-Desktop-Programmierung .....</b>	<b>207</b>
17.1	Textausgabe und Bildlaufleisten .....	207
17.1.1	Schriftarten und Textausgabe .....	207
17.1.2	Bildlaufleisten programmieren.....	208
17.2	Windows-Standarddialoge und Menüs .....	211
17.2.1	Datei-Öffnen- und Datei-Speichern-Dialog .....	211
17.2.2	Schriftwahl-Dialog.....	213
17.2.3	Menübefehle ergänzen .....	215
<b>Teil 4 Aufgabenpool .....</b>		<b>217</b>
1	Aufgaben zum Umfeld der Sprache C++ .....	218
2	Aufgaben zum ersten Programm in C++.....	218
3	Aufgaben zur Ein- und Ausgabe .....	219

---

4	Aufgaben zu Operatoren.....	220
5	Aufgaben zur Selektion und Iteration .....	222
6	Aufgaben zu Funktionen in C++ .....	226
7	Aufgaben zu Arrays in C++ .....	228
8	Aufgaben zu Zeigern.....	231
9	Aufgaben zu Strukturen .....	234
10	Aufgaben zu Klassen in C++ .....	237
11	Aufgaben zu Dateioperationen mit C++ .....	240
12	Aufgaben zur Überladung von Operatoren.....	242
13	Aufgaben zur Vererbung in C++ .....	246
14	Aufgaben zu virtuellen Methoden .....	248
15	Aufgaben zur fortgeschrittenen Programmierung.....	249
16	Aufgaben zur Windows-Desktop-Programmierung .....	252
17	Aufgaben zur fortgeschrittenen Windows-Desktop-Programmierung.....	253

**Teil 5 Lernsituationen..... 255**

Lernsituation 1:	Erstellen einer Präsentation mit Hintergrundinformationen zu der Sprache C++ (in Deutsch oder Englisch) .....	256
Lernsituation 2:	Anfertigen einer Kundendokumentation für den Einsatz einer Entwicklungsumgebung in C++ (in Deutsch oder Englisch) .....	257
Lernsituation 3:	Entwicklung eines Verschlüsselungsverfahrens für ein internes Memo-System der Support-Abteilung einer Netzwerk-Firma.....	259
Lernsituation 4:	Entwicklung eines Informationstools für die Anzeige von Umgebungsvariablen.....	260
Lernsituation 5:	Planung, Implementierung und Auswertung eines elektronischen Fragebogens.....	262
Lernsituation 6:	Realisierung einer Klasse zur Speicherung von Messwerten .....	265
Lernsituation 7:	Implementierung einer Klasse zur Simulation der echten Bruchrechnung....	267
Lernsituation 8:	Entwicklung eines einfachen Readers .....	269

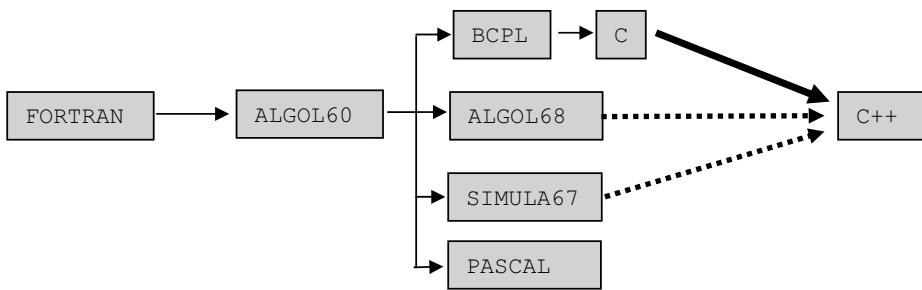
**Anhang A: Strukturierte Dokumentationstechniken..... 271**

Der Programmablaufplan (PAP): .....	271
Beispiel eines Programmablaufplanes:.....	272
Das Struktogramm (auch Nassi-Shneiderman-Diagramm): .....	273
Beispiel eines Struktogrammes:.....	274

**Anhang B: Index .....** **275****B: Index.....** **275**

### 1.1.3 Kleiner Stammbaum der Programmiersprachen

1950.....1960.....1970.....1980



#### C++ im Überblick:

- ▶ Entwickelt von Bjarne Stroustrup in den 80er-Jahren.
- ▶ Basiert auf der Sprache C und hat Konzepte der Sprachen SIMULA67 und ALGOL68 übernommen.
- ▶ C++ ist eine objektorientierte Sprache.
- ▶ Sie ist plattformunabhängig, systemnah und sehr schnell.
- ▶ Viele Betriebssysteme wurden in C++ programmiert.
- ▶ Die Sprache ist sehr weit verbreitet und kommt in technischen, kaufmännischen und wissenschaftlichen Anwendungen zum Einsatz.

## 1.2 Bestandteile eines C++-Programms

### 1.2.1 Was ist ein Programm?

Ein Programm besteht aus einer Folge von Anweisungen an den Computer. Diese Anweisungen (Befehle) werden in einer bestimmten Form gegeben – und zwar in einer speziellen Sprache (Programmiersprache).

Diese Programmiersprache wird vom Computer nicht direkt verstanden, sondern muss erst „übersetzt“ werden, und zwar in eine für den Computer verständliche Sprache, den **Maschinencode**.

Nach der Übersetzung kann das Programm gestartet werden.

```

Anweisung 1;
Anweisung 2;
Anweisung 3;
Anweisung 4;
Anweisung 5;
:
:
:
Anweisung N;
  
```



Eine endliche Folge von eindeutigen Anweisungen an den Computer nennt man **Algorithmus**.

### 1.2.2 C++-Quellcode

Die Anweisungen an den Computer in der Sprache C++ werden in Dateien gespeichert. Der Inhalt dieser Dateien wird als **Quellcode** bezeichnet.

Normalerweise haben C++-Quellcode-Dateien die Endungen

- ▶ `cpp` für Cplusplus
- ▶ `h` bzw. `hpp` für Header bzw. Header-plusplus.

# 2 Das erste C++-Programm

## 2.1 Ausgabe auf dem Bildschirm

Für die Ausgabe auf dem Bildschirm sind einige Vorbereitungen zu treffen.

Neben dem Kennenlernen der Entwicklungsumgebung, damit überhaupt ein Programm geschrieben werden kann, müssen noch weitere Punkte geklärt werden:

- Wie werden ein C++-Projekt und eine Quellcode-Datei angelegt?
- Welche Bibliothek muss für die Ausgabe auf dem Bildschirm eingebunden sein?
- Wie sieht das „Hauptprogramm“ aus?
- Mit welcher Anweisung wird eine Bildschirmausgabe erreicht?

Die folgenden Kapitel geben Aufschluss über diese Punkte.

### 2.1.1 Ein C++-Projekt in Visual Studio anlegen

Die integrierte Entwicklungsumgebung **Visual Studio 2017** ist eine komfortable Umgebung, um C++-Programme zu entwickeln. Besonders erfreulich ist der Umstand, dass die Umgebung kostenfrei als *Community-Edition* im Internet bereit steht. Ein C++-Programm besteht aus einer oder mehreren Quellcode dateien. Diese Dateien werden in einem Projekt organisiert.

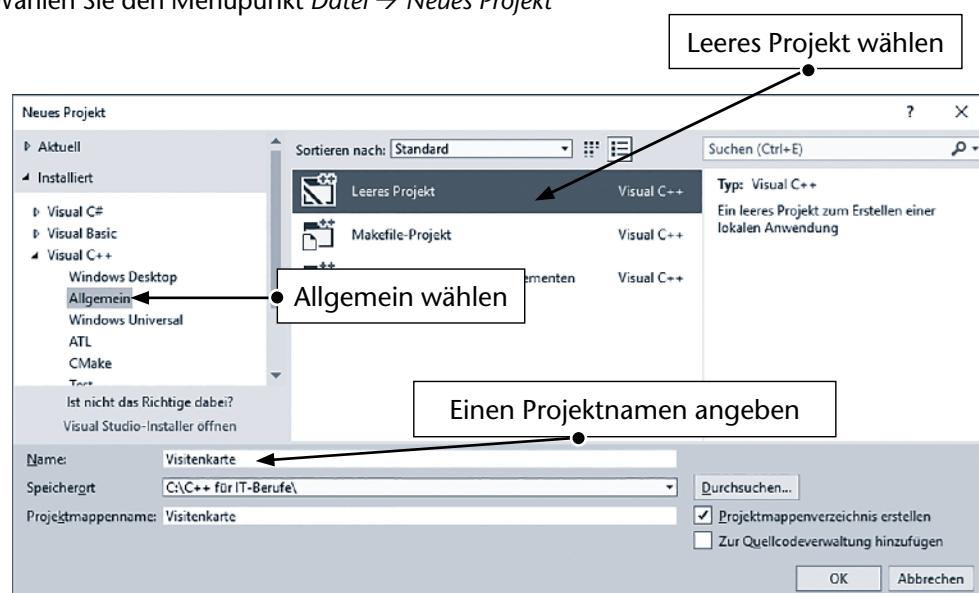
Visual C++ unterscheidet im Prinzip folgende Projektarten:

- Windows-Desktopanwendungen (eine Windows-Anwendung)
- **Windows-Konsolenanwendung / leere Anwendung (ähnlich einem DOS-Programm)**
- Universal Windows Platform – Anwendungen (Apps)
- Dynamic Link Library (Bibliotheksanwendung)

In diesem Buch ist hauptsächlich eine Projektform von Bedeutung: **die Windows-Konsolenanwendung bzw. die leere Anwendung**. Diese Form ist ausreichend, um die Programme in C++ auf dem Bildschirm darzustellen. Die Konsolenanwendung ist natürlich nicht so ansprechend wie ein Windows-Programm oder eine App, aber, um die Sprache C++ zu lernen, völlig ausreichend. Zum Abschluss des Buches wird die Windows-Desktopanwendung eingeführt, mit der klassische Windows-Programme realisiert werden können.

#### Anlegen eines neuen Projektes:

- Starten Sie *Visual Studio 2017*
- Wählen Sie den Menüpunkt *Datei* → *Neues Projekt*



# 5 Selektion und Iteration

Die Selektion (Auswahl) und die Iteration (Wiederholung) sind zwei sehr wichtige Konstrukte in einer Programmiersprache. In den vorherigen Kapiteln sind schon einige Probleme ohne diese Konstrukte gelöst worden, aber große und immer komplexer werdende Programme können ohne Selektion und Iteration nicht auskommen.

## 5.1 Die Selektion

### 5.1.1 Darstellung der Selektion mit einem Programmablaufplan

#### Problemstellung:

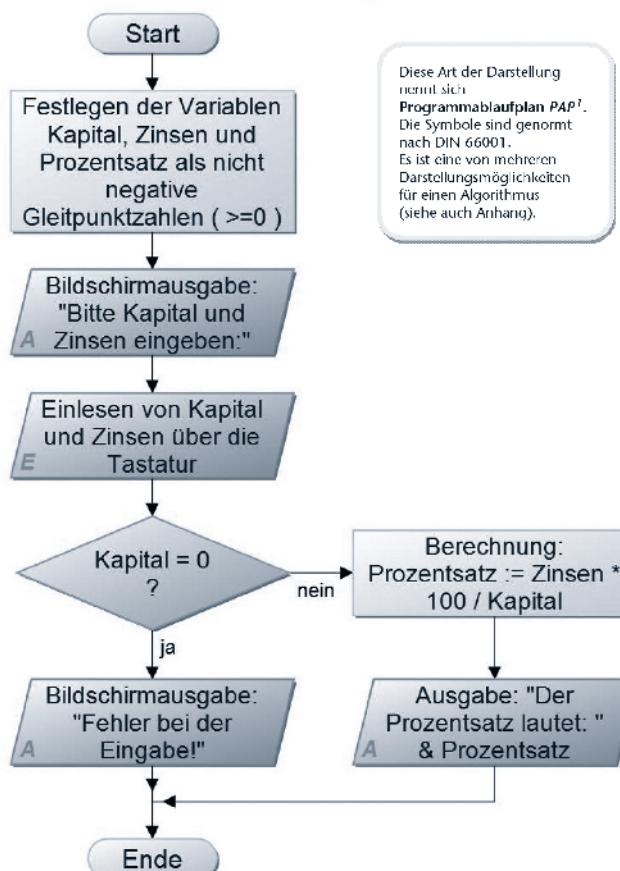
Es soll eine Berechnung des Prozentsatzes bei gegebenem Kapital und Zinsen durchgeführt werden.

$$\text{Formel: } p = \frac{Z * 100}{K}$$

Das Programm kann einen Fehler verursachen, wenn für das Kapital der Wert Null eingegeben wird (die Division durch Null ist verboten).

#### Lösungsmöglichkeit:

Das Programm erkennt, ob eine Null eingegeben wurde, und führt dann keine Berechnung durch. Das Problem wird durch eine Dokumentationstechnik, den Programmablaufplan, zuerst schematisch erfasst. Anschließend wird dann auf die Umsetzung in C++ eingegangen.



<sup>1</sup> Der Programmablaufplan wurde mit dem Programm PapDesigner erstellt. Diese Software wurde speziell für die Ausbildung im IT-Bereich entwickelt und steht kostenfrei im Internet zum Download bereit.

**Beispiel:**

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

int main()
{
    int Werte[5] = { 3, 4, 7, 2, 10 };
    for ( auto wert : Werte)
    {
        cout << wert << endl;
    }
    return 0;
}
```

Das Array wird vom ersten bis zum letzten Element durchlaufen- Die Variable `wert` erhält automatisch den korrekten Datentyp. Auf die Variable ist allerdings nur lesend zugreifbar.

Bildschirmausgabe des Programmlaufs:

```
3
4
7
2
10
Drücken Sie eine beliebige Taste . . .
```

### 7.1.2 Mehrdimensionale Arrays

Die Vorstellung von mehrdimensionalen Arrays ist gerade am Anfang relativ schwer. Deshalb ist es sinnvoll, zuerst eine Vorstellung von einem zweidimensionalen bzw. dreidimensionalen Array zu bekommen.

Die Vorstellung eines zweidimensionalen Arrays entspricht einer Tabelle. Tabellen sind allgemein bekannt: Sie bestehen aus Zeilen und Spalten, die den entsprechenden Index des Arrays widerspiegeln.

**Beispiel:**

Es wird ein zweidimensionales Array angelegt.

```
float Tab [ 3 ] [ 4 ];
```

entspricht der Anzahl der Spalten

entspricht der Anzahl der Zeilen

	Spalte 0	Spalte 1	Spalte 2	Spalte 3
Zeile 0	1.5	7	12.33	77.5
Zeile 1	124	99.99	453	67.89
Zeile 2	12	90.2	2727.5	22

```
Tab [ 1 ] [ 2 ] == 453
```

Spaltenangabe

Zeilenangabe

Dreidimensionale Arrays kann man sich als eine Sammlung von Tabellenblättern vorstellen, die hintereinander angeordnet sind.

```

        cin >> Knoten->Rechts->Inhalt;
        Knoten->Rechts->Links = NULL;
        Knoten->Rechts->Rechts = NULL;
    }
    break;
}
}while (wahl != 3);

```

#### 4. Die Funktion menu:

Je nachdem, ob es bereits einen Eintrag bei Vater bzw. Mutter gibt, wird die Bildschirmausgabe gestaltet.

```

int menu(TKnoten * Aktuell)
{
    cout << "\t\t" << Aktuell->Inhalt;
    cout << endl;
    if (Aktuell->Links != NULL)
        cout << "\t" << Aktuell->Links->Inhalt;
    else cout << "\t Vater?";
    if (Aktuell->Rechts != NULL)
        cout << "\t\t" << Aktuell->Rechts->Inhalt;
    else cout << "\t\t Mutter?";
    cout << endl;
    cout << "<0> Zur Wurzel gehen" << endl;
    if (Aktuell->Links != NULL)
        cout << "<1> Weitergehen zum Vater" << endl;
    else cout << "<1> Eingabe des Vaters" << endl;
    if (Aktuell->Rechts != NULL)
        cout << "<2> Weitergehen zur Mutter" << endl;
    else cout << "<2> Eingabe der Mutter" << endl;
    cout << "<3> Ende" << endl;
    cout << "Ihre Eingabe bitte: " << endl;
    int wahl;
    cin >> wahl;
    return wahl;
}

```

#### Hinweis:

Die höheren Datenstrukturen wie Liste und Baum haben Vorteile bezüglich der beliebigen Speicherung von Daten. Das Einfügen und Löschen ist bei der Liste besonders einfach. Das Suchen hingegen aufwändig, da die Liste immer sequenziell durchlaufen werden muss. Bei einem Baum erfolgt die Suche nach einem Element schneller, allerdings muss der Baum dann auch weiteren Bedingungen genügen – das führt schnell zu komplexeren Theorien wie dem AVL-Baum, auf die hier nicht weiter eingegangen wird.

Bevor man in einem Programm eine Liste oder einen Baum implementiert, sollte man prüfen, ob nicht ein gewöhnliches Array mit dynamischer Speicherverwaltung ausreicht, um die Daten zu speichern. Der Zugriff ist jedenfalls deutlich schneller und die Handhabung über den Indexoperator weniger mühsam.

Mithilfe des Indexoperators und der Methode `substr()` wird der String in den beiden `for`-Schleifen zeichenweise ausgegeben.

Die Methode `c_str()` liefert einen Zeiger vom Typ `char`. Damit können dann alle C-Zeichenkettenfunktionen benutzt werden.

```
C:\WINDOWS\system32\cmd.exe
Hallo
Hallo
2
Drücken Sie eine beliebige Taste . . .
```

### 15.3.2 Intelligente Zeiger

Die dynamische Speicherreservierung liegt in den Händen des Entwicklers, da C++ kein verwalteter Code ist und deshalb auch kein *garbage-collector* wie in Java oder C# für die Aufräumarbeiten sorgt. Dementsprechend ist die Speicherreservierung immer eine Quelle von Fehlern. Die intelligenten Zeiger sorgen dafür, dass dieses Problem entschärft wird. Dazu werden neue Arten von Zeigern zu Verfügung gestellt: `unique_ptr` und `shared_ptr`.

**Der `unique_ptr`:**

Der `unique_ptr` sorgt dafür, dass ein Zeiger immer nur auf ein dynamisches Objekt verweisen kann. Damit ist ausgeschlossen, dass es zusätzliche Verweise auf ein Objekt im Speicher gibt. Das folgende Beispiel zeigt die Verwendung eines solchen Zeigers:

**Beispiel:**

```
#include <memory>
#include <iostream>

using namespace std;
void Test()
{
    unique_ptr<double> zeiger (new double(1.5));
}

int main()
{
    unique_ptr<int> zeiger_1 (new int(10));
    unique_ptr<int> zeiger_2;
    //zeiger_2 = zeiger_1;

    if (zeiger_1) cout << "Zeiger_1 ist gueltig!" << endl;
    else cout << "Zeiger_1 ist nicht gueltig!" << endl;
    cout << "Inhalt von Zeiger_1: " << *zeiger_1 << endl;
    zeiger_1.reset();
}
```

Einbinden der Header-Datei `<memory>`!

Ein `unique_ptr` erhält ein dynamisch erstelltes Objekt vom Typ `double`. Wenn die Funktion beendet wird, wird auch der lokale `unique_ptr` gelöscht und damit automatisch auch das `double`-Objekt.

Zwei `unique_ptr` werden angelegt. Der erste Zeiger erhält ein dynamisch erstelltes Objekt vom Typ `int`.

**FEHLER:** Der Compiler verhindert die Zuweisung eines `unique_ptr` auf einen anderen `unique_ptr`.

Wie gewohnt mit dem Inhaltsoperator arbeiten.

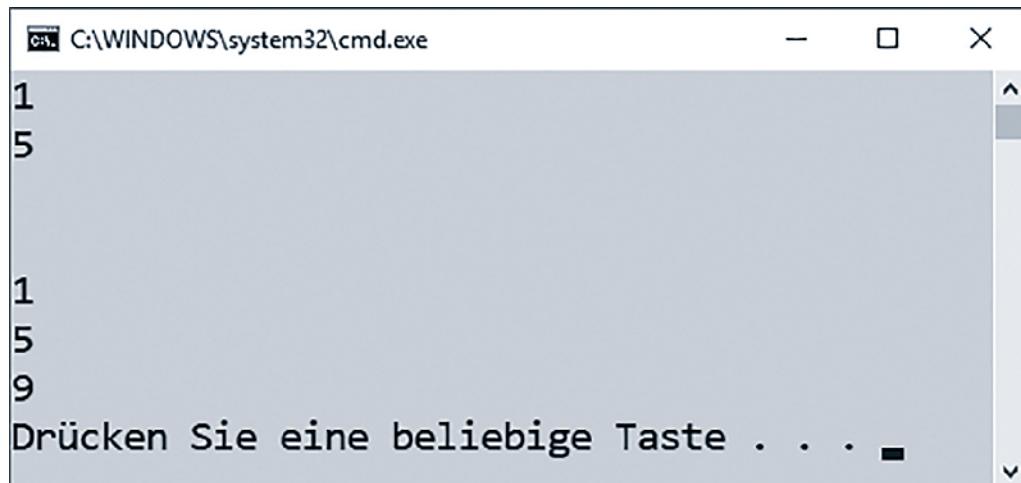
Den Speicher explizit freigeben!

```

        for (int i = 0; i < intContainer.capacity(); i++)
            cout << intContainer[i] << endl;
    return 0;
}

```

Nach dem Starten erscheint die folgende Ausgabe:



```

C:\WINDOWS\system32\cmd.exe
1
5
1
5
9
Drücken Sie eine beliebige Taste . . . -

```

Der `vector` speichert zuerst zwei Werte, danach wird der Platz erhöht und ein weiterer Wert wird zugewiesen.

Das folgende Beispiel zeigt die Verwendung der `map`-Klasse:

**Beispiel:**

Header einbinden

```

#include <map>
#include <iostream>
#include <string>
using namespace std;
int main()
{
    map < string, string > testMap;
    testMap["Deutschland"] = "Berlin";
    testMap["Italien"] = "Rom";
    testMap["Frankreich"] = "Paris";
    cout << testMap["Deutschland"] << endl;
    return 0;
}

```

Es wird eine Instanz der Template-Klasse `map` gebildet. Schlüssel und Element sind beide vom Typ `string`.

Einfügen von Elementen mit entsprechenden Schlüsseln.

Nach dem Starten erscheint die folgende Ausgabe:



```

C:\WINDOWS\system32\cmd.exe
Berlin
Drücken Sie eine beliebige Taste . . . -

```

Das Beispiel zeigt die Verwendung eines assoziativen Arrays. Die Hauptstädte sind die `string`-Elemente, die über einen Schlüssel (Land) abrufbar sind.

```

ShowWindow(hWnd, nCmdShow);
UpdateWindow(hWnd);

return TRUE;
}

```

Ansonsten bringt **ShowWindow** das Fenster auf den Bildschirm und **UpdateWindow** sorgt für die erste Aktualisierung.

**Die Fensterprozedur – das Herz der Anwendung:**  
hier werden alle Ereignisse (Benutzeraktionen, Systemnachrichten) verarbeitet und der Inhalt des Fensters entsprechend dargestellt.

```

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{

```

**switch (message)**  
{

Der Kern der Funktion ist eine große **switch**-Anweisung, die auf die Nachrichten in den entsprechenden **case**-Blöcken reagiert.

Die Variable **message** enthält Nachrichten vom Typ **WM...**  
(= Windows Message ...). Die **WM\_COMMAND**-Nachricht zeigt eine Benutzereingabe an (z.B. Auswahl eines Menüpunktes).

```
case WM_COMMAND:
```

```
{  
    int wmid = LOWORD(wParam);
```

```
    switch (wmid)
```

Der Benutzer hat die Infobox aus dem Menü gewählt. Mit der Funktion **DialogBox** wird sie angezeigt.

```
case IDM_ABOUT:
```

```
    DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
```

```
    break;
```

Der Benutzer hat die „Beenden“ aus dem Menü gewählt. Mit Hilfe der Funktion **DestroyWindow** wird die Nachricht **WM\_DESTROY** an das Betriebssystem gesendet. Das Betriebssystem reiht die Nachricht in die Nachrichtenschlange ein.

```
case IDM_EXIT:
```

```
    DestroyWindow(hWnd);
```

```
    break;
```

Die Funktion **About** zeigt die „Infobox“ an. So wie in der Fensterprozedur werden auch hier Nachrichten ausgewertet.

```
INT_PTR CALLBACK About (HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
        case WM_INITDIALOG:
            return (INT_PTR)TRUE;

        case WM_COMMAND:
            if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
            {
                EndDialog(hDlg, LOWORD(wParam));
                return (INT_PTR)TRUE;
            }
            break;
    }
    return (INT_PTR)FALSE;
}
```

## 16.2 Nachrichtenbearbeitung

### 16.2.1 Die Nachricht WM\_PAINT

Die Nachricht **WM\_PAINT** wurde bereits in der Darstellung des ersten Programms erläutert. Diese Nachricht ist aber so wichtig, dass noch einmal speziell darauf eingegangen werden muss: Wenn ein Fenster in irgendeiner Form verdeckt bzw. überdeckt wird, so wird genau dann die Nachricht **WM\_PAINT** gesendet, wenn das Fenster erneut den Focus erhält. Ebenso wird die **WM\_PAINT** gesendet, wenn das Fenster in seiner Größe geändert wird. **Dadurch wird sichergestellt, dass der Inhalt eines Fensters immer auf dem aktuellen Stand gehalten wird.** Im Gegensatz zu Konsolen-Applikationen ist es also sehr wichtig, dass das Programm jederzeit den gültigen Fensterinhalt ausgeben kann – und zwar angepasst an mögliche Veränderungen wie eine Größenänderung.

Das folgende Beispiel zeigt, wie oft diese Nachricht ausgesendet wird, indem jeder Aufruf der Nachricht gezählt wird und auf dem Bildschirm angezeigt wird. Dazu kann eine API-Funktion verwendet werden, die eine Art Meldungsfenster (**MessageBox**) anzeigt:

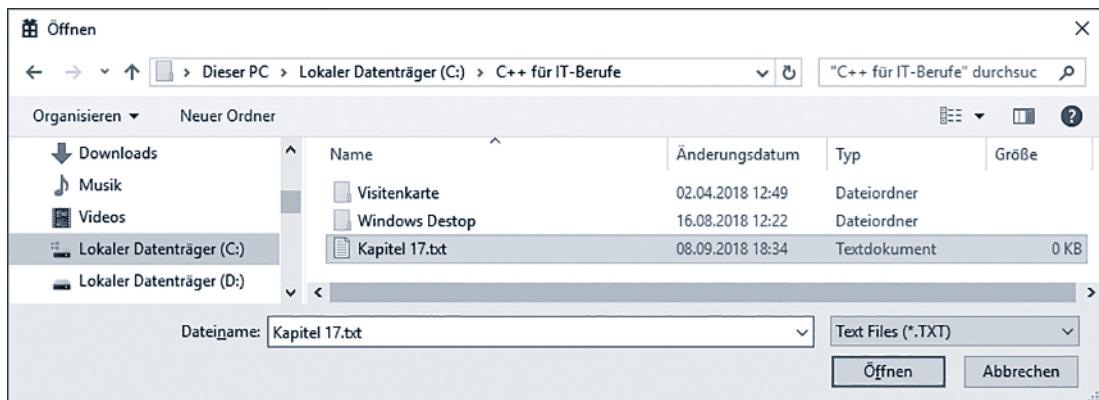
```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
                           WPARAM wParam, LPARAM lParam)
{
    static int counter = 1;
    :
    :
    :
    case WM_PAINT:
    {
        PAINTSTRUCT ps;
        HDC hdc = BeginPaint(hWnd, &ps);

        WCHAR Text[100+1];
```

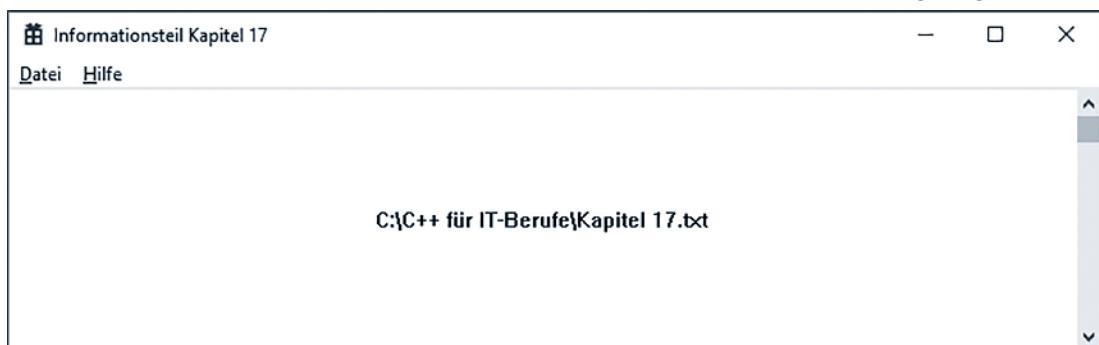
Statt einer globalen Variablen für einen Zähler, kann eine lokale Variable als **static** deklariert werden. Damit ist die Sichtbarkeit lokal, aber die Variable behält ihren Wert über die Funktionsaufrufe hinweg.

Ein statisches Zeichenketten-Array vom Typ **WCHAR** (Unicode).

Nach dem Starten öffnet sich sofort der Datei-Öffnen-Dialog:



Nach der Auswahl einer Textdatei wird der Name (incl. Pfad) im Clientbereich angezeigt:



### 17.2.2 Schriftwahl-Dialog

Mit dem `ChooseFont`-Dialog kann eine der installierten Schriftarten unter Windows ausgewählt werden. Dazu muss eine Struktur vom Typ `CHOOSEFONT` mit Werten gefüllt werden und an den Dialog übergeben werden. Der Dialog füllt anschließend diese Strukturvariable mit den Werten der gewählten Schriftart. Die API-Funktion `ChooseFont` ist wie folgt deklariert:

```
BOOL ChooseFont (LPCHOOSEFONT lpcf);
```

Die Funktion liefert ein `TRUE` zurück, wenn der Benutzer eine Schriftart gewählt und den OK-Button geklickt hat.

Das Füllen der `CHOOSEFONT`-Struktur ist etwas aufwändiger. Deshalb ist es sinnvoll die Struktur in einer eigenen Header-Datei etwas zu modifizieren. In der `CHOOSEFONT`-Struktur muss auch eine `LOGFONT`-Struktur angegeben werden, die ebenfalls durch den Aufruf der Funktion gefüllt wird. Anschließend kann diese Struktur-Variable dann für das Einsetzen der eigentlichen Schrift in den Gerätekontext genutzt werden.

Die folgende Header-Datei (FontDlg.h) zeigt diese Modifizierung:

```
#include <commndlg.h>           Header-Datei für die Dialoge einbinden.
#include "stdafx.h"
CHOOSEFONT cf;                  Globale Variablen (der Einfachheit halber) für die Schriftart-Daten.

void DialogInit(HWND hwnd)
{
    cf.lStructSize = sizeof(CHOOSEFONT);
    cf.hwndOwner = hwnd;
    cf.hDC = NULL;
    cf.lpLogFont = &lf;           Strukturvariable für die Schriftart angeben!
    cf.iPointSize = 0;
    cf.Flags = CF_INITTOLOGFONTSTRUCT | CF_SCREENFONTS | CF_EFFECTS;
    cf.rgbColors = 0;
    cf.lCustData = 0;
    cf.lpfnHook = NULL;
```

Ihre Wahl: 3

Verkauf eines Power-PCs:

Bitte geben Sie die Modembezeichnung ein: *Fritz 1.5*  
Bitte geben Sie die Grafikkarte ein: *Hercules 6*  
Bitte geben Sie den Speicher ein: *512 MB*  
Bitte geben Sie den Prozessor ein: *Intel 2.2 GHz*  
Bitte geben Sie die Kundenummer ein: *4443*

Ihre Wahl: 4

Verkauf eines Server-PCs:

Wie viele Netzwerkkarten?: 2  
Bitte geben Sie die 1. Karte ein: *MAXCOM 1*  
Bitte geben Sie die 2. Karte ein: *MAXCOM 2*  
Bitte geben Sie den Speicher ein: *1024 MB*  
Bitte geben Sie den Prozessor ein: *Intel 3.2 GHz*  
Bitte geben Sie die Kundenummer ein: *3337*

Ihre Wahl: 5

Liste der verkauften PC-Systeme:

1. Multimedia-PC an Kunde: 1737
2. Internet-PC an Kunde: 3428
3. Power-PC an Kunde: 4443
4. Server-PC an Kunde: 3337

### Aufgabe 14.2

Finden Sie drei sinnvolle verschiedene Beispiele für Vererbungshierarchien, die eine abstrakte Basisklasse haben.

## 15 Aufgaben zur fortgeschrittenen Programmierung

### Aufgabe 15.1

Entwickeln Sie eine Klasse `CList`, die ein Container für verschiedene Datentypen sein soll. Nutzen Sie dazu die Klassen-Templates unter C++.

#### Beispiel eines möglichen Hauptprogrammes:

```
int main()
{
    CList <float> FloatListe;
    FloatListe.Einfuegen(10.5);
    FloatListe.Einfuegen(22.44);
    FloatListe.Einfuegen(12.12);

    for ( int i = 0 ; i < FloatListe.Anzahl() ; i++)
        cout << FloatListe[i];

    return 0;
}
```

#### Hinweise:

- Das Speichern der Elemente soll in Form einer verketteten Liste erfolgen.
- Benutzen Sie die Ausnahmebehandlung, um Speicherfehler und andere Fehler professionell abzufangen.
- Überladen Sie den Indexoperator, so dass ein Zugriff wie bei gewöhnlichen Arrays möglich ist.
- Implementieren Sie weitere sinnvolle Methoden und Operatoren wie beispielsweise eine Methode `Loeschen()`, um Elemente zu löschen.

**Beispiel: DAS IST EIN TEST**

Verschlüsselung: 32 15 44 62 22 44 45 62 23 22 24 62 45 23 44 45

Die Realisierung der Verschlüsselung setzt eine eingehende Auseinandersetzung mit den ein- und mehrdimensionalen Arrays aus dem Informationsteil voraus.

**Durchführung:**

Implementieren Sie ein Modul mit geeigneten Funktionen, die die o. a. Verschlüsselung durchführen können. Die verschlüsselten Zeichenketten sollen dabei in Arrays vom Datentyp `int` gespeichert werden. Das Modul soll für beliebige Schlüsselworte sowohl verschlüsseln als auch entschlüsseln. Beachten Sie bei der Umsetzung die Regeln der modularen Programmgestaltung. Trennen Sie Schnittstelle und Implementation.

**Kontrolle:**

Jedes Entwicklerteam stellt handschriftlich eine Verschlüsselungsmatrix mit einem selbst gewählten Schlüsselwort auf. Diese Matrizen dienen dann als Testgrundlage für die Kontrolle der korrekten Ver- und Entschlüsselung.

Testen Sie das Modul unter den Bedingungen des Auftrages. Verschlüsseln und entschlüsseln Sie sehr lange Zeichenketten, die vom Umfang her einem Memotext entsprechen. Das könnten ca. 300 Worte oder ca. 2000 Zeichen sein.

**Lernziele:**

- ▶ Sie lernen eine interessante Anwendung der Programmierung kennen – die Verschlüsselungstechnik.
- ▶ Sie erarbeiten die nötigen Kenntnisse über ein- und mehrdimensionale Arrays in C++.
- ▶ Sie erkennen die Besonderheiten der Zeichenkettenverarbeitung in C++.

## Lernsituation 4:

### Entwicklung eines Informationstools für die Anzeige von Umgebungsvariablen

**Ausgangssituation:**

Die Entwickler der Firma **ProSource** arbeiten oft mit den sogenannten Umgebungsvariablen, die das Betriebssystem (Windows oder UNIX) anbietet. In diesen Umgebungsvariablen stehen relevante Daten wie der aktuelle Username, der am System angemeldet ist, oder freigegebene Pfadangaben.

Bisher müssen die Entwickler die Informationen über einen Aufruf der Konsole und den entsprechenden Befehl (z.B.: SET unter Windows) abfragen. Diese Vorgehensweise ist mühsam und zeitaufwändig.

Aus diesem Grund hat das Entwicklerteam den Wunsch geäußert, dass die Firma ein Tool zur Verfügung stellt, das das Auslesen der Variablen angenehmer und schneller gestaltet.

Als Auszubildender der Firma **ProSource** erhalten Sie den Auftrag, dieses Tool zu programmieren. Das Tool sollte menügesteuert sein und verschiedene Filter für die Anzeige der Umgebungsvariablen anbieten.

Instanz 120  
 int 27, 38  
 Integrierte Entwicklungsumgebungen 18  
 iomanip 34  
 iostream 22  
 Ist-Beziehung 154  
 Iterationen 53

**K**

Klasse 117  
 Klassen-Schablone 168  
 Klassentemplates 167  
 Knoten 110, 114  
 Kommentare 25  
 Kommentarkopf 26  
 Komprimierungstool 241  
 kopfgesteuert 55

**L**

LIFO 239  
 lineare Gleichungen 220  
 Linker 17  
 Links-Wert 45  
 LISP 218  
 Listen 110  
 Lochkarten 134  
 Logische Operatoren 41  
 logische Zustände 41  
 lokale Variablen 65  
 long 27  
 long double 28, 38  
 long int 38

**M**

Magnetband 134  
 main() 23  
 Manipulatoren 34, 219  
 Maschinencode 15, 218  
 Median 229  
 Mehrdimensionale Arrays 80  
 Mehrfachselektion 51  
 Mehrfachvererbung 154, 159  
 Methoden 117  
 Mittlere Abweichung 229  
 Modularer Programmaufbau 70  
 Module 71  
 Modulo-Operator 39  
 Multiplikationsmatrix 230

**N**

Namensraum 24, 72  
 namespace 73  
 NEGATION 41  
 new 100  
 NULL 110  
 Nullterminierung 84

**O**

Oberklasse 154  
 Objekt 117  
 objektbasierte Programmiersprache 161  
 ODER 41  
 ofstream 135  
 open() 136

**P**

Palindrom 230  
 Parameter 66  
 Parameterkonstruktoren 122  
 Pfeiloperator „->“ 107  
 PL/I 218  
 Polymorphismus 161  
 Positionieren des Dateizeigers 142  
 Postfix-Notation 40  
 Präprozessor 74  
 private 119, 155  
 private-Vererbung 158  
 Programmablaufplan 48  
 protected 119, 155, 156  
 protected-Vererbung 158  
 public 119, 155  
 public-Vererbung 157  
 Punktoperator „..“ 107

**Q**

qsort 98, 233  
 Quellcode 15

**R**

Rang von Operatoren 45  
 read 141  
 Rechts-Wert 45  
 Records 106  
 Referenz 104  
 Referenzoperator & 104  
 Referenzparameter 95  
 Referenzübergabe 95  
 reinterpret\_cast 44  
 rein virtuelle Methode 165  
 Rekursive Funktionen 68  
 Relationale Operatoren 40  
 return 63  
 Rückgabedatentyp 63  
 Rumpf der Funktion 63

**S**

Schaltjahr 222  
 Schlange 236  
 Schnittstelle 71  
 Scope-Operator :: 66  
 seekg() 142  
 seekp() 142  
 Selektion 48

sequenzielle Organisation 134  
 short 27  
 SIMULA67 15, 218  
 sizeof-Operator 44  
 SMALLTALK 218  
 Sonderzeichen 33  
 Sortieren 86  
 Sortieren durch Auswahl 87  
 Spannweite 229  
 Spezialisierung 154  
 STACK 68, 239  
 Standardausgabe 32  
 Standardkonstruktör 122  
 static\_cast 44  
 Statische Klassenelemente 132  
 statisches Klassenattribut 132  
 statisches Linken 18  
 strcpy 111, 123  
 Stream-Objekte 134, 135  
 Streamstatus 36  
 struct 106  
 Strukturen 106  
 Strukturen in Strukturen 108  
 switch 51

**T**

tellg() 144  
 tellp() 144  
 Templates 166  
 Textmodus 135  
 this-Zeiger 131  
 throw 169  
 Trennzeichen 25  
 true 29  
 try 169  
 type 240  
 Typumwandlung 44

**U**

Überladen des Ausgabeoperators 153  
 Überladen des Eingabeoperators 152  
 Überladen des Indexoperators 151  
 Überladen des Zuweisungsoperators 150  
 Überladener Operator als Methode 149  
 Überladen von Funktionen 67  
 Überladen von Operatoren 145  
 Umdrehen 230  
 Umgebungsvariablen 260, 261  
 UND 41  
 unsigned 27  
 Unterklasse 154