

# C++

**Lernen und professionell anwenden**

Mit Microsoft Visual C++ 2010  
Express Edition auf der CD



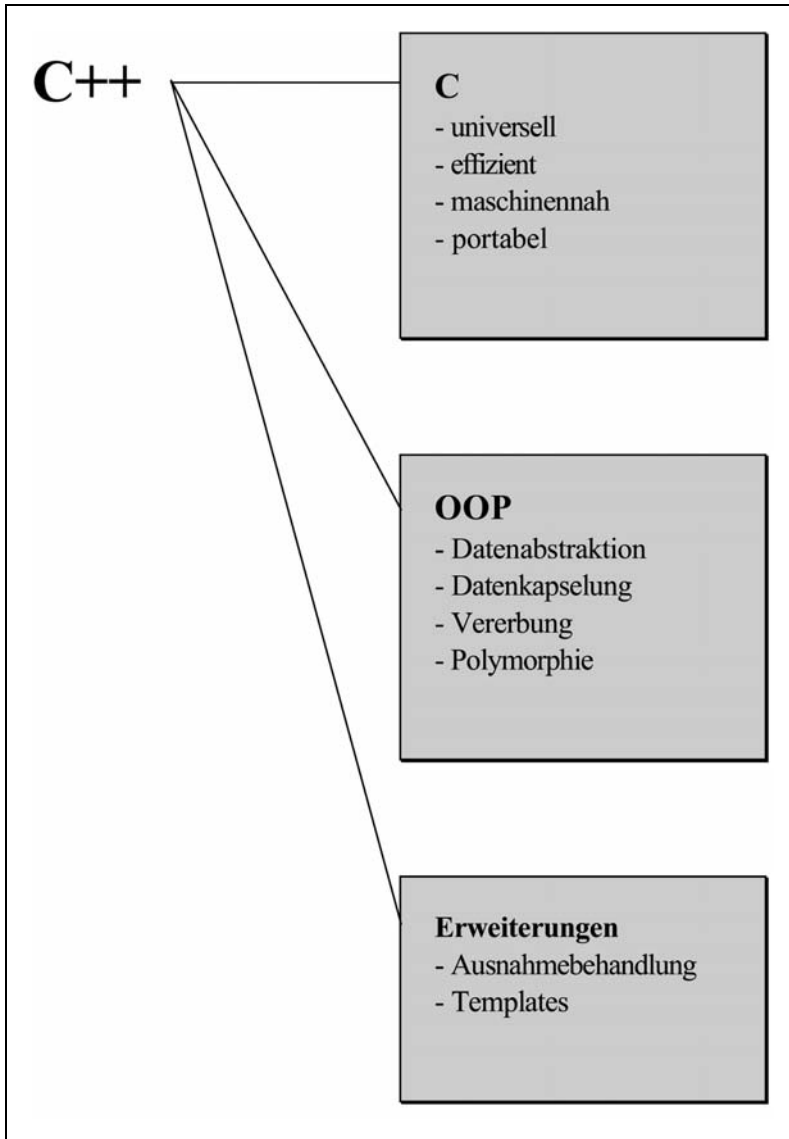
# Kapitel 1

## Grundlagen

Dieses Kapitel beschreibt die grundlegenden Eigenschaften der objektorientierten Programmiersprache C++. Außerdem werden die Schritte vorgestellt, die zur Erstellung eines lauffähigen C++-Programms erforderlich sind. Diese Schritte können Sie auf Ihrem System anhand von einführenden Beispielen nachvollziehen. Die Beispiele dienen auch dazu, die grundlegende Struktur eines C++-Programms darzustellen.

## Entwicklung und Eigenschaften von C++

### Charakteristische Eigenschaften



## Historisches

Die Programmiersprache C++ wurde von Bjarne Stroustrup und seinen Mitarbeitern in den Bell Laboratories (AT&T, USA) entwickelt, um Simulationsprojekte objektorientiert und effizient implementieren zu können. Frühe Versionen, die zunächst als „C mit Klassen“ bezeichnet wurden, gibt es seit 1980. Der Name C++ weist darauf hin, dass C++ aus der Programmiersprache C hervorgegangen ist: ++ ist der Inkrementoperator von C.

Schon 1989 wurde ein ANSI-Komitee (American National Standards Institute) gebildet, um die Programmiersprache C++ zu standardisieren. Hierbei geht es darum, dass möglichst viele Compilerbauer und Software-Entwickler sich auf eine einheitliche Sprachbeschreibung einigen, um die Bildung von Dialekten und „Sprachverwirrungen“ zu vermeiden.

Anfang 1998 wurde von der ISO (International Organization for Standardization) der Standard für C++ (CD 14882) verabschiedet.

## Eigenschaften von C++

C++ ist keine rein objektorientierte Sprache, sondern eine Hybridsprache (= „Mischsprache“): Sie enthält die Programmiersprache C als Teilmenge. Damit hat man zunächst alle Möglichkeiten, die auch C bietet:

- universell einsetzbare, modulare Programme
- effiziente, maschinennahe Programmierung
- Portabilität, d.h. Übertragbarkeit von Programmen auf verschiedene Rechner

Insbesondere kann die umfangreiche in C entwickelte Software auch in C++-Programmen verwendet werden.

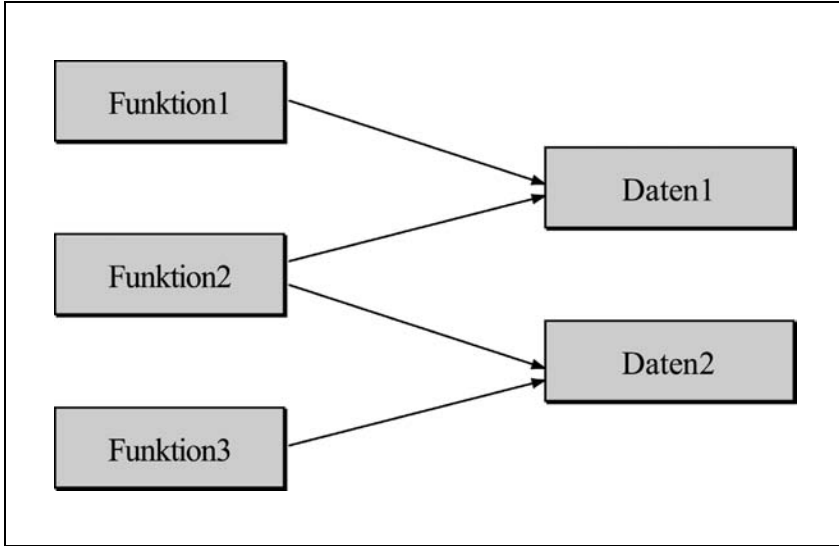
C++ unterstützt die Konzepte der objektorientierten Programmierung (kurz: OOP), nämlich:

- *Datenabstraktion*, d.h. Bildung von Klassen zur Beschreibung von Objekten
- *Datenkapselung* für den kontrollierten Zugriff auf die Daten von Objekten
- *Vererbung* durch Bildung abgeleiteter Klassen (auch mehrfach)
- *Polymorphie* (griech. „Vielgestaltigkeit“), d.h. die Implementierung von Anweisungen, die zur Laufzeit verschiedene Wirkungen haben können

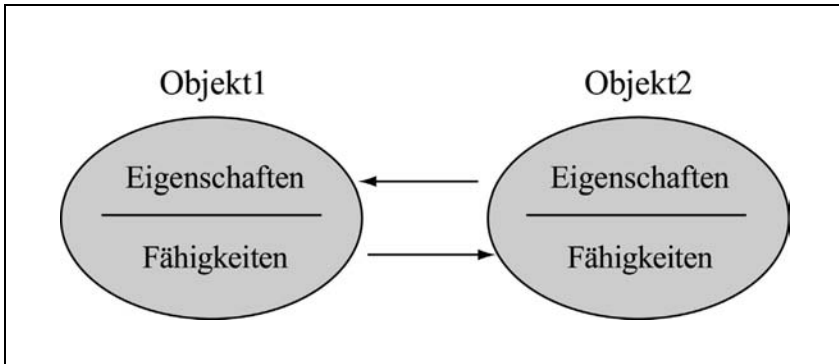
C++ wurde um zusätzliche Sprachelemente erweitert, wie z. B. Referenzen, Templates und Ausnahmebehandlung (engl. Exception Handling). Auch wenn diese Sprachelemente keinen direkten Bezug zur Objektorientierung haben, sind sie für deren effiziente Implementierung wichtig.

## Objektorientierte Programmierung

### Traditionelles Konzept



### Objektorientiertes Konzept



## Klassische, prozedurale Programmierung

Die traditionelle, prozedurale Programmierung trennt Daten und Funktionen (= Unterprogramme, Prozeduren), die diese Daten bearbeiten. Dies hat wichtige Konsequenzen für den Umgang mit den Daten in einem Programm:

- Der Programmierer muss selbst dafür sorgen, dass Daten vor ihrer Verwendung mit geeigneten Anfangswerten versehen sind und dass beim Aufruf einer Funktion korrekte Daten übergeben werden.
- Wird die Darstellung der Daten geändert, z.B. ein Datensatz erweitert, so müssen auch die zugehörigen Funktionen entsprechend angepasst werden.

Beides ist natürlich fehleranfällig und nicht besonders wartungsfreundlich.

## Objekte

Die objektorientierte Programmierung (OOP) stellt die *Objekte* in den Mittelpunkt, d.h. die Dinge, um die es bei der jeweiligen Problemstellung geht. Ein Programm zur Verwaltung von Konten beispielsweise arbeitet mit Kontoständen, Kreditlimits, Überweisungen, Zinsberechnungen usw. Ein Objekt, das ein Konto in einem Programm darstellt, besitzt dann die Eigenschaften und Fähigkeiten, die für die Kontoverwaltung wichtig sind.

Die Objekte in der OOP bilden eine Einheit aus Daten (= Eigenschaften) und Funktionen (= Fähigkeiten). Mit einer Klasse wird ein Objekt-Typ definiert, der sowohl die Eigenschaften als auch die Fähigkeiten von Objekten dieses Typs festlegt. Die Kommunikation zwischen Objekten erfolgt dann durch „Nachrichten“, die die Fähigkeiten von Objekten aktivieren.

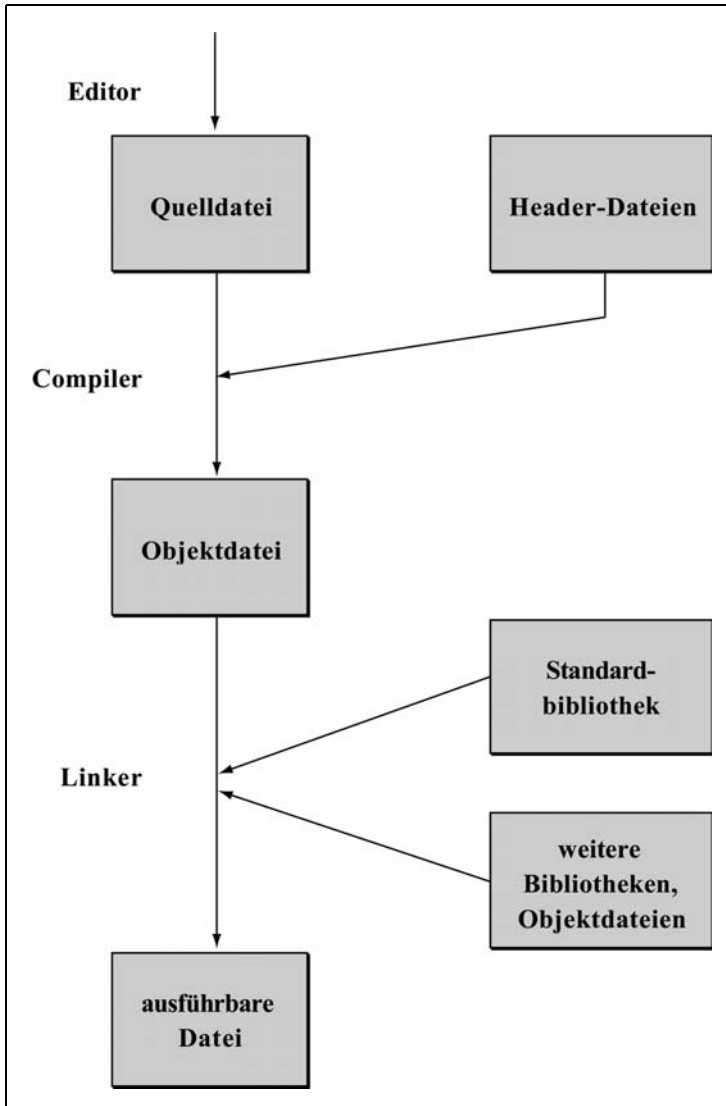
## Vorteile der OOP

Für die Software-Entwicklung hat die objektorientierte Programmierung wesentliche Vorteile:

- **geringere Fehleranfälligkeit:** Ein Objekt kontrolliert den Zugriff auf seine Daten selbst. Insbesondere kann es fehlerhafte Zugriffe abwehren.
- **gute Wiederverwendbarkeit:** Ein Objekt verwaltet sich selbst. Es kann deshalb wie ein Baustein in beliebigen Programmen eingesetzt werden.
- **geringer Wartungsaufwand:** Ein Objekt kann die interne Darstellung seiner Daten an neue Anforderungen anpassen, ohne dass ein Anwendungsprogramm etwas davon merkt.

## Erstellen eines C++-Programms

### Übersetzen eines C++-Programms



Zum Erstellen und Übersetzen eines C++-Programms sind folgende drei Schritte notwendig:

1. Das C++-Programm wird mit einem Texteditor in eine Datei eingegeben. Anders gesagt: Der *Quellcode* wird in einer *Quelldatei* abgelegt.  
Bei größeren Projekten ist es üblich, *modular* zu programmieren. Dabei wird der Quellcode auf mehrere Quelldateien verteilt, die getrennt bearbeitet und übersetzt werden.
2. Die Quelldatei wird dem *Compiler* zur Übersetzung gegeben. Geht alles gut, so erhält man eine Objektdatei, die den *Maschinencode* enthält. Eine Objektdatei heißt auch *Modul*.
3. Schließlich bindet der *Linker* die Objektdatei mit anderen Modulen zu einer *ausführbaren Datei*. Die Module bestehen aus Funktionen der Standardbibliothek oder selbsterstellten, schon früher übersetzten Programmteilen.

Im *Namen* der Quelldateien muss die richtige Endung angegeben werden. Diese hängt vom jeweiligen Compiler ab. Die gebräuchlichsten Endungen sind `.cpp` und `.cc`.

Vor dem eigentlichen Kompilervorgang können *Header-Dateien*, auch Include-Dateien genannt, in die Quelldatei kopiert werden. Header-Dateien sind Textdateien mit Informationen, die in verschiedenen Quelldateien gebraucht werden, wie z.B. Typdefinitionen oder die Deklaration von Variablen und Funktionen. Die Namen von Header-Dateien enden entweder mit `.h` oder haben keine Endung.

Die C++-*Standardbibliothek* enthält fertige Funktionen und Klassen, die standardisiert sind und die jeder Compiler zur Verfügung stellt.

Moderne Compiler bieten eine *integrierte Entwicklungsumgebung*, die die obigen drei Schritte zusammenfaßt. Von einer einheitlichen Benutzeroberfläche aus wird das Programm editiert, kompiliert, gelinkt und ausgeführt. Außerdem können weitere Tools, wie z.B. ein Debugger, gestartet werden.

**Wichtig:** Enthält die Quelldatei auch nur einen *Syntaxfehler*, so zeigt dies der Compiler mit einer *Fehlermeldung* an. Darüber hinaus können weitere Fehlermeldungen ausgegeben werden, die darauf beruhen, dass der Compiler versucht, trotz des Fehlers weiter zu übersetzen. Beginnen Sie also immer mit der Korrektur des ersten Fehlers in einem Programm.

Neben den eigentlichen Fehlermeldungen gibt der Compiler auch *Warnungen* aus. Eine Warnung zeigt keinen Syntaxfehler an, sondern ist lediglich ein Hinweis auf einen möglichen logischen Fehler, wie beispielsweise die Verwendung einer nicht initialisierten Variablen.

## Ein erstes C++-Programm

### Beispielprogramm

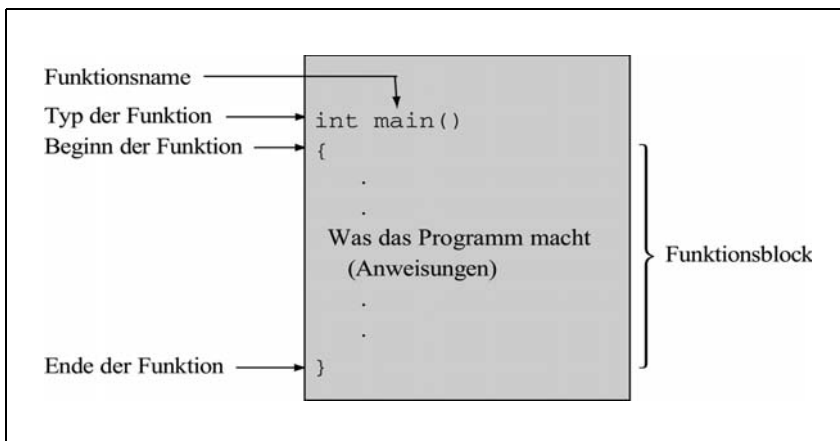
```
#include <iostream>
using namespace std;

int main()
{
    cout << "Viel Spaß mit C++!" << endl;
    return 0;
}
```

### Bildschirmausgabe

```
Viel Spaß mit C++!
```

### Die Struktur der Funktion main()



Ein C++-Programm besteht aus Objekten mit ihren *Elementfunktionen* und aus *globalen Funktionen*, die zu keiner Klasse gehören. Jede Funktion löst eine bestimmte Aufgabe und kann andere Funktionen aufrufen. Sie ist entweder selbsterstellt oder eine fertige Routine aus der Standardbibliothek. Die globale Funktion `main()` muss immer selbst geschrieben werden und hat eine besondere Rolle: Sie bildet das Hauptprogramm.

Anhand des nebenstehenden kurzen Beispielprogramms lassen sich bereits die wichtigsten Elemente eines C++-Programms erkennen. Das Programm enthält nur die Funktion `main()` und gibt eine Meldung auf dem Bildschirm aus.

Die erste Zeile beginnt mit einem Doppelkreuz `#` und ist daher für den *Präprozessor* bestimmt. Der Präprozessor ist Teil der ersten Übersetzungsphase, in der noch kein Objektcode erzeugt wird. Mit

```
#include <dateiname>
```

kopiert der Präprozessor die genannte Datei an diese Stelle in den Quellcode. Dadurch stehen dem Programm alle Informationen zur Verfügung, die in dieser Datei enthalten sind.

Die Header-Datei `iostream` enthält Vereinbarungen für die Ein-/Ausgabe mit Streams. Der Begriff *Stream* (dt. Strom) spiegelt die Situation wider, dass Zeichenfolgen wie ein Datenstrom behandelt werden.

Die in C++ vordefinierten Namen befinden sich im Namensbereich `std` (standard). Die anschließende `using`-Direktive erlaubt, die Namen aus dem Namensbereich (*Namespace*) `std` direkt zu verwenden.

Die Programmausführung beginnt mit der ersten Anweisung in der Funktion `main()`, die daher in jedem C++-Programm vorhanden sein muss. Nebenstehend ist die Struktur der Funktion dargestellt. Sie unterscheidet sich, abgesehen vom feststehenden Namen, nicht von der Struktur anderer Funktionen.

In unserem Beispiel enthält die Funktion `main()` zwei *Anweisungen*. Die erste

```
cout << "Viel Spaß mit C++!" << endl;
```

gibt den Text `Viel Spaß mit C++!` auf dem Bildschirm aus. Der Name `cout` (`console output`) bezeichnet ein Objekt, das die Ausgabe durchführt.

Die beiden Kleiner-Zeichen `<<` deuten an, dass die Zeichen in den Ausgabestrom „geschoben“ werden. Mit `endl` (end of line) wird anschließend ein Zeilenvorschub ausgelöst. Die zweite Anweisung

```
return 0;
```

beendet die Funktion `main()` und damit das Programm. Dabei wird der Wert `0` dem aufrufenden Programm als Exit-Code zurückgegeben. Es ist üblich, den Exit-Code `0` zu verwenden, wenn das Programm korrekt abgelaufen ist.

Beachten Sie, dass die Anweisungen mit einem Semikolon enden. Die kürzeste Anweisung besteht übrigens nur aus einem Semikolon und bewirkt nichts.

## Struktur einfacher C++-Programme

### Ein C++-Programm mit mehreren Funktionen

```
/*
*****
Ein Programm mit mehreren Funktionen und Kommentaren
*****
*/

#include <iostream>
using namespace std;

void linie(), meldung();           // Prototypen

int main()
{
    cout << "Hallo! Das Programm startet in main()."
          << endl;
    linie();
    meldung();
    linie();
    cout << "Jetzt am Ende von main()." << endl;

    return 0;
}

void linie()                       // Eine Linie ausgeben.
{
    cout << "-----" << endl;
}

void meldung()                     // Eine Meldung ausgeben.
{
    cout << "In der Funktion meldung()." << endl;
}
```

### Bildschirmausgabe

```
Hallo! Das Programm startet in main().
-----
In der Funktion meldung().
-----
Jetzt am Ende von main().
```

Das nebenstehende Beispiel zeigt, wie ein C++-Programm strukturiert ist, das aus mehreren Funktionen besteht. Die Reihenfolge, in der Funktionen definiert werden, ist in C++ nicht vorgeschrieben. Zum Beispiel könnte auch zuerst die Funktion `meldung()`, dann die Funktion `linie()` und schließlich die `main`-Funktion geschrieben werden.

Üblicherweise wird die Funktion `main()` zuerst angegeben, da sie den Programmablauf steuert. Es werden also Funktionen aufgerufen, die erst später definiert werden. Dies ist möglich, da der Compiler mit dem *Prototyp* einer Funktion die notwendigen Informationen erhält.

Neu in diesem Beispiel sind die *Kommentare*. Jede Zeichenfolge, die durch `/* . . . */` eingeschlossen ist oder mit `//` beginnt, ist ein Kommentar.

**Beispiele:** `/* Ich darf mehrere Zeilen lang sein */`  
`// Ich bin ein Einzeilen-Kommentar`

Beim Einzeilen-Kommentar ignoriert der Compiler ab den Zeichen `//` alle Zeichen bis zum Zeilenende. Kommentare, die sich über mehrere Zeilen erstrecken, sind bei der Fehlersuche nützlich, um ganze Programmteile auszublenden. Jede der beiden Kommentarformen kann auch benutzt werden, um die andere auszukommentieren.

Zum *Layout* einer Quelldatei: Der Compiler bearbeitet jede Quelldatei sequentiell und zerlegt den Inhalt in „Token“ (kleinste Bestandteile), wie z.B. Funktionsnamen und Operatoren. Token können durch beliebig viele Zwischenraumzeichen getrennt sein, also durch Leer-, Tabulator- oder Newline-Zeichen. Es kommt daher nur auf die Reihenfolge des Quellcodes an, nicht auf ein bestimmtes Layout, wie etwa die Aufteilung in Zeilen und Spalten. Beispielsweise ist

```
void meldung
( ) { cout <<
    "In der Funktion meldung()." <<
endl; }
```

zwar eine schlecht lesbare, aber korrekte Definition der Funktion `meldung()`.

Eine Ausnahme hinsichtlich des Layouts bilden die Präprozessor-Direktiven, die stets eine eigene Zeile einnehmen. Dem Doppelkreuz `#` zu Beginn einer Zeile dürfen nur Leer- und Tabulatorzeichen vorangehen.

Zur besseren Lesbarkeit von C++-Programmen ist es von Vorteil, einen einheitlichen Stil in der Darstellung beizubehalten. Dabei sollten Einrückungen und Leerzeilen gemäß der Programmstruktur eingefügt werden. Außerdem ist eine großzügige Kommentierung wichtig.

## Übungen

### Programm-Listing zur 3. Aufgabe

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

void pause();          // Prototyp

int main()
{
    cout << endl << "Lieber Leser, "
         << endl << "gönnen Sie"
         << " sich jetzt eine ";
    pause();
    cout << "!" << endl;

    return 0;
}

void pause()
{
    cout << "PAUSE";
}
```

### 1. Aufgabe

Erstellen Sie ein C++-Programm, das den folgenden Text auf dem Bildschirm ausgibt:

```
Moegen
taeten wir schon wollen,
aber koennen
haben wir uns nicht getraut!
```

Verwenden Sie den Manipulator `endl` an den entsprechenden Stellen.

### 2. Aufgabe

Das folgende Programm enthält lauter Fehler:

```
*/ Hier sollte man die Brille nicht vergessen //
#include <stream>
int main
{
    cout << "Wenn dieser Text",
    cout >> " auf Ihrem Bildschirm erscheint, ";
    cout << " endl;"
    cout << 'können Sie sich auf die Schulter '
        << "klopfen!" << endl.
    return 0;
}
```

Korrigieren Sie die Fehler, und testen Sie die Lauffähigkeit des Programms.

### 3. Aufgabe

Was gibt das nebenstehende C++-Programm auf dem Bildschirm aus?

## Lösungen

### Zur 1. Aufgabe:

```
// Auf geht's !

#include <iostream>
using namespace std;

int main()
{
    cout << "Moegen " << endl;
    cout << "taeten wir schon wollen" << endl;
    cout << "aber koennen " << endl;
    cout << "haben wir uns nicht getraut!" << endl;

    return 0;
}
```

### Zur 2. Aufgabe:

Die korrigierten Stellen sind unterstrichen.

```
/* Hier sollte man die Brille nicht vergessen */
#include <iostream>

using namespace std;
int main()
{
    cout << "Wenn dieser Text";
    cout << " auf Ihrem Bildschirm erscheint, ";
    cout << endl;
    cout << "können Sie sich auf die Schulter "
        << "klopfen!" << endl;
    return 0;
}
```

### Zur 3. Aufgabe:

Die Bildschirmausgabe des Programms beginnt in einer neuen Zeile:

```
Lieber Leser,
gönnen Sie sich jetzt eine PAUSE!
```