Walter Saumweber

C++ Programmierhandbuch

Walter Saumweber

C++

Programmierhandbuch

entwickler.press

Walter Saumweber
C++ Programmierhandbuch
ISBN: 978-3-86802-220-9

Ein Imprint der Software & Support Verlag GmbH

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Ihr Kontakt zum Verlag und Lektorat:
Software & Support Verlag GmbH
entwickler.press
Geleitsstraße 14
60599 Frankfurt am Main
Tel: +49(0) 69 63 00 89 - 0
Fax: +49(0) 69 63 00 89 - 89
lektorat@entwickler-press.de
http://www.entwickler-press.de

Inhaltsverzeichnis

Teil IV
Fortgeschrittene Programmierkonzepte 475

Liebe Leserin, lieber Leser!

Vielen Dank, dass Sie sich für dieses Buch entschieden haben. Es sollte Ihnen dazu verhelfen, auf möglichst angenehme Weise ein gutes Stück Weg in der Welt der Programmierung im Allgemeinen und in der von C++ im Besonderen hinter sich zu bringen. Allerdings ist wie bei den meisten Dingen im Leben auch hierzu etwas Geduld erforderlich. Gerade dem programmierunerfahrenen Leser sei es deshalb nahe gelegt, diese aufzubringen, auch wenn ihm die ersten Kapitel möglicherweise etwas langweilig erscheinen werden (dem Fortgeschrittenen steht es natürlich frei, an einer späteren Stelle mit dem Lesen zu beginnen).

Ich habe mich grundsätzlich bemüht, die Lerninhalte mit möglichst vielen prägnanten, aber einfachen Beispielen zu verdeutlichen. Die Lektüre dieses Buches sollte daher auch ohne unmittelbaren Rechnerkontakt möglich sein (ich selbst gehöre zu den Leuten, die nicht gerne am Computer sitzen, wenn sie ein Buch lesen). Praktische Übung, zumindest im Nachhinein, ist natürlich immer anzuraten.

Alle abgedruckten Listings plus erweiterte Beispiele finden Sie, nach Kapiteln geordnet, als Quellcodedateien und – soweit es sich um größere oder inhaltlich wichtige Programme handelt – samt ausführbarer Datei auf der Buch-CD im Verzeichnis *Beispiele*. Speziell das Visual-Studio-Projekt aus Kapitel 34 befindet sich als *.zip*-Archiv im Unterordner *K34*.

Die Beispiele können Sie mit dem Borland C++-Compiler kompilieren. Er steht ebenfalls auf der Buch-CD im Verzeichnis *Borland-Compiler* zur Verfügung (Dateiname *freecommandLinetools.exe*). Die Installation und Bedienung ist in Kapitel 2 und 4 beschrieben. Es sei jedoch darauf hingewiesen, dass Sie genauso gut jeden anderen aktuellen Compiler verwenden können.

Auf die Abhandlung einiger Fortgeschrittenen-Themen wie beispielsweise Templates habe ich bewusst – wenn auch nicht ganz ohne schlechtes Gewissen – verzichtet. Als Gegenleistung sollte der Leser eine gut verständliche Darstellung der behandelten Themen in Händen halten (da diese Beurteilung letzten Endes immer auf persönlicher Einschätzung beruht, möchte ich hiermit der Hoffnung Ausdruck geben, dass mir dies gelungen ist).

Um aus dem Buch den größtmöglichen Nutzen zu ziehen, empfehle ich Ihnen, sich für die Lektüre und für das Nachvollziehen der Beispiele Geduld und Zeit zu nehmen.

Eventuell zusätzliche Infos zum Buch sowie gegebenenfalls eine Fehlerliste werde ich beizeiten auf meiner Webseite *http://www.mipueblo.de* veröffentlichen. Über die dort angegebene E-Mail-Adresse können Sie mich jederzeit kontaktieren. Für Anregungen und Hinweise bin ich Ihnen schon jetzt sehr dankbar.

Und nun viel Spaß beim Lesen und viel Erfolg mit der Programmierung.

Walter Saumweber

Teil I

Grundlagen

Dieser einführende Teil soll Ihnen den Einstieg in die Programmierung erleichtern. Bevor wir uns speziell mit der Programmiersprache C++ befassen, werden wir zunächst auf allgemeine Aspekte der Programmierung eingehen. Insbesondere werden Sie mit Fachbegriffen bekannt gemacht, die Ihnen im weiteren Verlauf des Buches immer wieder begegnen werden. Im Anschluss daran werden Sie ein erstes kleines C++-Programm erstellen und ausführen, wobei Sie einige Tipps zu Entwicklungswerkzeugen erhalten. Falls Sie schon einige Programmiererfahrung haben und sich auch die Frage der zu verwendenden Software für Sie nicht stellt, steht es Ihnen natürlich frei, diesen Teil zu überspringen und sogleich mit Teil 2 fortzufahren.

1 Was ist Programmieren?

»… sich um das Entstehen einer ausführbaren Datei zu kümmern.« Diese einfache Antwort ist nahe liegend, da das Endprodukt jeder erfolgreichen Programmentwicklung in einer solchen (ausführbaren Datei) besteht.

Eine ebenso lapidare – und auch mehr oder weniger korrekte – Erwiderung würde lauten: »Programmieren bedeutet, Text in einen Editor einzugeben.«

Hinweis

Die eigentliche Schwierigkeit liegt natürlich darin, dies syntaktisch und logisch korrekt, effizient und auch für andere Programmierer nachvollziehbar zu tun (Letzteres wird beim Modifizieren von Programmen bedeutsam). Was das alles genau bedeutet, werden Sie im Weiteren sehen.

Da eine Textdatei eben nicht ausführbar ist, scheint die zweite Aussage im Widerspruch zur ersten zu stehen. Lassen Sie uns die Klärung mit einer weiteren Frage einleiten.

1.1 Was ist eigentlich ein Computerprogramm?

Ein ausführbares Programm enthält Befehle, die vom Computer ausgeführt werden. Die Reihenfolge der Abarbeitung ist dabei durch die Kodierung der Programmdatei festgelegt.

Hinweis

Die Begriffe »Programmdatei«, »ausführbare Datei«, »Programm« und »ausführbares Programm« werden gleichbedeutend verwendet. Es handelt sich um solche Dateien, die vom Computer zur Ausführung vorgesehen sind. Dies trifft z.B. für Text- oder Grafikdateien nicht zu.

Zu den Begriffen »Programm« sowie »ausführbares Programm« werden bezüglich dieser Definition später noch gewisse Einschränkungen zu nennen sein, die aber hier keine Rolle spielen.

Programmdateien können auf verschiedene Weise zur Ausführung gelangen. Möglich ist ein Aufruf aus bereits laufenden anderen Programmen heraus: Programm A startet Programm B (dieses wiederum Programm C usw.). Einige Programme werden vom Betriebssystem automatisch beim Start des Computers in Gang gesetzt.

Am anschaulichsten geschieht dies jedoch durch einen expliziten Startbefehl des Anwenders – durch einen Doppelklick auf die Programmdatei im Windows-Explorer, mithilfe einer Verknüpfung (Desktop, Startmenü) oder über einen Befehl in der Konsole (die bei DOS und Windows als »Eingabeaufforderung« bezeichnet wird).

Wenn Sie beispielsweise mit dem Tabellenkalkulationsprogramm Microsoft Excel arbeiten möchten, werden Sie dieses vermutlich über einen Eintrag im Startmenü oder mittels eines Symbols auf dem Desktop aufrufen. Da es sich bei beiden um Verknüpfungen zur entsprechenden Programmdatei handelt, wird diese damit gestartet. Der Name der Excel-Programmdatei lautet übrigens *excel.exe*. Sie befindet sich in der Regel im Verzeichnis *C:\Programme\Microsoft Office\Office* bzw. *C:\Programme\Microsoft Office\OFFICE11* oder *C:\Programme\Microsoft Office\Office12*.

Achtung

Für alle Pfadangaben in diesem Buch gilt: Falls sich bei Ihnen die Dateien bzw. Verzeichnisse auf einem anderen Laufwerk befinden, ersetzen Sie den Laufwerksbuchstaben *C* durch den jeweiligen Buchstaben.

Eine Programmdatei wird in den meisten Betriebssystemen an der Dateierweiterung erkannt. Die am häufigsten anzutreffende Erweiterung von ausführbaren Dateien unter Windows und DOS ist *.exe*. Unter Unix und Unix-Derivaten wie Linux gibt es für ausführbare Dateien keine vorgeschriebene Erweiterung, die meisten Programme verzichten daher ganz auf eine Erweiterung.

Auch die Resultate Ihrer zukünftigen Bemühungen werden sich vor allem in *.exe*-Dateien zeigen (vorausgesetzt, Sie arbeiten mit einem Windows-Betriebssystem).

Wenn also Ihr Computer auf eine Datei mit der Erweiterung *.exe* trifft, wird er versuchen, den Inhalt dieser Datei als Befehlsfolge zu verstehen und auszuführen. Wenn dieser Versuch misslingt, wird er dies mit einer Fehlermeldung kundtun.

Hinweis

Genau genommen ist es der Prozessor Ihres Computers, der die Datei zu interpretieren versucht, und der Mittler zwischen Programmdatei und Prozessor ist das Betriebssystem. Mit anderen Worten, das Betriebssystem reicht *.exe*-Dateien an den Prozessor des Computers zur Ausführung weiter.

Was hier am Beispiel von *.exe*-Dateien geschildert wird, gilt natürlich in Abhängigkeit vom Betriebssystem ebenso für bestimmte andere Dateierweiterungen – wie oben erwähnt.

Angenommen, Sie würden eine Textdatei beliebigen (!) Inhalts editieren und diese mit der Erweiterung *.exe* speichern. Auf einen entsprechenden Startbefehl hin würde Ihr Computer auch diese Datei als Programmdatei ansehen. Bleibt die Hoffnung, dass in diesem Fall alle Interpretationsversuche fehlschlagen, andernfalls würde das nicht vorhersehbare Aktionen zur Folge haben. Es verhält sich dann gewissermaßen so, als ob ein

Spanier einem Italiener in der irrigen Annahme, es handle sich um einen Landsmann, Anweisungen erteilt und dabei Worte wählt, die auch im Italienischen vorkommen, nur eben in ganz anderer Bedeutung.

Hinweis

Aus dem genannten Grund handelt es sich um keinen wirklich ernst gemeinten Vorschlag. Möglicherweise interpretiert Ihr Computer eine Zeichenfolge als »Lösche … «. Ich empfehle Ihnen also, von solchen Experimenten abzusehen und mir auch so zu glauben.

Eine Programmdatei ist somit nur dann ausführbar (und verdient auch nur in diesem Fall, so genannt zu werden), wenn die in ihr enthaltenen Befehle vom Computer verstanden werden können. Das setzt voraus, dass sie in der Sprache des Computers, der so genannten Maschinensprache, abgefasst sind.

Befehle in Maschinencode sind binärkodiert, das heißt, sie liegen als Folge von Nullen und Einsen vor:

```
01101010100101010101110001011100
10101010110101100101100111101011
00101000110101110101100100101010
```

Der offensichtliche Nachteil von Maschinencode liegt in seiner schlechten Lesbarkeit. Er lässt sich kaum sinnvoll strukturieren und ist somit nur schwer nachvollziehbar. Hinzu kommt, dass sich Maschinenbefehle nur schwer aus dem Gedächtnis reproduzieren lassen. Alles zusammengenommen gestaltet sich die Entwicklung eines Programms auf diese Weise sehr schwierig, ganz zu schweigen von einer eventuellen späteren Überarbeitung.

Wie viel erfreulicher wäre es, wenn ein Computer etwa folgendermaßen kodierte Anweisungen akzeptieren würde:

```
Nimm eine Eingabe des Anwenders entgegen.
Prüfe, ob die Eingabe als Zahlenwert interpretierbar ist.
Falls nicht, schreibe die Zeichenfolge "Falsche Eingabe" auf den Bildschirm.
…
```

Nun, ganz so einfach ist es leider nicht (andernfalls würden Sie dieses Buch nicht lesen). Allerdings lassen sich Wortschatz und Grammatik moderner Programmiersprachen wie C++ etwa auf einem Niveau zwischen Maschinensprache und dieser gedachten Variante ansiedeln, was die Sache schon sehr viel angenehmer macht.

Hinweis

Es sei angemerkt, dass nahezu alle Programmiersprachen sich angloamerikanischer Begriffe bedienen.

In gewisser Weise erscheint eine Programmiersprache wie C++ sogar eleganter und unkomplizierter als jede Umgangssprache, da sie über geeignete Schlüsselwörter und Zeichen verfügt, mit denen sich extrem präzise das formulieren lässt, was man erreichen möchte – Konfusionen wie in der Alltagssprache sind ausgeschlossen.

1.2 Entwicklungsschritte zum ausführbaren Computerprogramm

Welche Verbindung besteht nun zwischen der unvermeidbaren Maschinensprache (Nur diese versteht ja der Computer!) und der Programmiersprache C++? Um dies aufzuzeigen, wollen wir in aller Kürze die Entwicklungsstationen zum fertigen Programm skizzieren, und zwar von Anfang an.

1.2.1 Aufsetzen des Quellcodes

Ursprung jeder fertigen Anwendung ist der so genannte Quellcode. Dieser besteht aus reinem Text, setzt sich also aus Buchstaben, Zahlen und anderen Zeichen zusammen, die Ihnen von der Tastatur Ihres PCs her bekannt sein dürften. Der Quellcode (Englisch »source code«) wird daher auch als »Quelltext« bezeichnet. Vor allem, wenn man nur einen Teil des Quellcodes meint, spricht man auch kurz von »Code«.

Die Hauptaufgabe für Sie als Programmierer besteht im Erstellen dieses Quellcodes. Dazu müssen Sie diesen, wie eingangs erwähnt, in einen Editor eingeben, wobei Sie den Vorgaben der verwendeten Programmiersprache unterworfen sind. Eben das Erlernen dieser Vorgaben – oder positiv ausgedrückt: Möglichkeiten – der Programmiersprache C++ bildet das Thema dieses Buches.

Hinweis

Die Bezeichnung »Programm« wird – je nach Sichtweise – nicht nur für die fertige Anwendung, sondern mitunter auch für den – noch nicht lauffähigen – Quellcode verwendet. Während der Anwender unter »Programm« in der Regel die ausführbare Anwendung versteht, meint der Programmierer mit dieser Bezeichnung häufig in erster Linie den von ihm verfassten Quellcode.

1.2.2 Kompilierung

Nichtsdestotrotz ist es letzten Endes erforderlich, dass die ausführbare Programmdatei in Maschinensprache vorliegt. Es muss also eine binärkodierte Entsprechung des C++-Quellcodes hergestellt werden. Das heißt nichts anderes, als dass der Quelltext in Maschinensprache zu übersetzen ist. Dies besorgt eine Software, der so genannte Compiler.

Der Compiler übersetzt (kompiliert) den Quellcode als Ganzes in Maschinensprache und speichert das Endprodukt in einer eigenen Datei (eben unter Windows mit der Dateierweiterung *.exe*). Dies geschieht in verhältnismäßig kurzer Zeit und ohne Ihr wei-

teres Zutun. Da es sich beim Compiler ebenfalls um ein Computerprogramm handelt, müssen Sie dieses lediglich starten.

Allerdings sind auch Compiler keine Multitalente. Sie unterscheiden sich in Abhängigkeit sowohl von der Programmiersprache als auch vom System, für das die zukünftige Anwendung bestimmt ist. So ist etwa ein C++-Compiler nicht in der Lage, einen in einer anderen Programmiersprache wie z.B. Pascal geschriebenen Quellcode zu übersetzen. Dazu bedarf es eines Compilers, der die Sprache Pascal versteht, wie z.B. Turbo Pascal. Da wir ja C++-Programme schreiben wollen, liegen wir jedoch mit einem C++-Compiler völlig richtig.

Aber auch ein unter Windows kompiliertes C++-Programm ist z.B. auf Unix/Linux-Systemen grundsätzlich nicht lauffähig. Wenn Sie also ein entsprechendes Linux-Programm erzeugen möchten, dann müssen Sie denselben Quelltext mit einem für diese Plattform vorgesehenen C++-Compiler erneut kompilieren.

Hinweis

Dass dies überhaupt möglich ist, ist keineswegs selbstverständlich. In den Urzeiten der Programmierung war es nicht die Regel, dass ein in einer bestimmten Programmiersprache verfasster Quelltext auf mehreren Betriebssystemen kompilierbar war. Wenn ein Quelltext auf mehreren Systemen kompiliert werden kann, sagt man, er (bzw. die entsprechende Programmiersprache) ist portierbar. Die Portierbarkeit von Programmiersprachen hat sich erst in den Achtzigerjahren zum Standard entwickelt.

Natürlich verrichtet der Compiler seine Arbeit nur dann, wenn der Quelltext syntaktisch korrekt ist, ansonsten quittiert er den Kompilierversuch mit entsprechenden Fehlermeldungen. In diesem Fall müssen Sie den Quellcode nochmals überarbeiten und den Kompiliervorgang erneut in Gang setzen.

Nach erfolgreichem Übersetzungsvorgang liegt das Programm nunmehr in seiner ausführbaren Form vor (Abbildung 1.1).

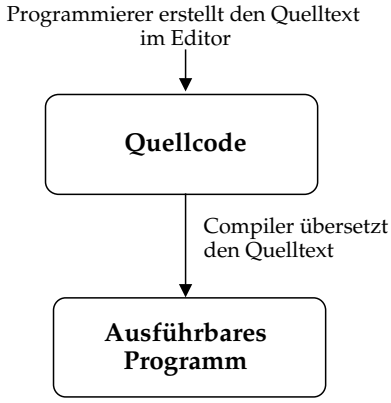

Abbildung 1.1: Der Weg zum lauffähigen Programm

Hinweis

Selbstverständlich durchläuft zumindest jede professionelle Anwendung in der Regel eine mehr oder weniger umfangreiche Testphase, bevor die Entwicklung als abgeschlossen betrachtet werden kann. Diese haben wir bei unseren Betrachtungen großzügig außer Acht gelassen.

Beachten Sie, dass die Quellcodedatei beim Übersetzungsvorgang unverändert erhalten bleibt. Es liegen nunmehr als Ergebnis der Programmierarbeit zwei Dateien vor: die Quellcodedatei – oder kurz »Quelldatei« – und die ausführbare Programmdatei (*.exe*-Datei). Streng genommen sind es sogar drei Dateien, da, sozusagen als Zwischenprodukt, zusätzlich eine so genannte Objektdatei (Endung *.obj*) entsteht. Diese soll uns aber zum jetzigen Zeitpunkt nicht weiter interessieren.

1.2.3 Präprozessor und Linker

Genau genommen leistet der eigentliche C++-Compiler nur einen – wenn auch den größten – Teil des Übersetzungsvorgangs, an dem außerdem noch Präprozessor und Linker beteiligt sind (das Wort »Präprozessor« soll Sie nicht irritieren; wie beim Linker und dem eigentlichen Compiler handelt es sich auch beim Präprozessor um Software – diese hat mit dem Prozessor Ihres Computers nichts zu tun).

Hinweis

Da der Compiler bei der Übersetzung des Quelltextes die Hauptaufgabe leistet, ist es allgemein üblich, die ganze Software für den Übersetzungsvorgang als »Compiler« zu bezeichnen. Analog dazu nennt man den vollständigen Übersetzungsvorgang Kompilierung.

Da ein Compiler (also die Gesamtheit der für den Übersetzungsvorgang benötigten Software) in modernen integrierten Entwicklungsumgebungen (»IDE«, dazu gleich mehr) immer enthalten ist, verwendet man das Wort »Compiler« mitunter auch synonym für die ganze integrierte Entwicklungsumgebung. Der Begriff »Compiler« kommt somit in dreifacher Bedeutung vor.

Der Kompiliervorgang stellt sich für den Programmierer in der Wahrnehmung als ein einziger Schritt dar. Wenn Sie z.B. mit Visual Studio oder der Visual C++ 2005 Express Edition arbeiten, erfolgt die Kompilierung, also das Erzeugen einer *.exe*-Datei, auf einen einzigen Mausklick hin. Bei dem C++-Compiler von Borland, den Sie auf der Buch-CD finden, geben Sie dazu einen Befehl auf der Konsole ein. Präprozessor und Linker treten so gesehen nach außen hin nicht sichtbar in Erscheinung[1]. Dennoch soll ihre Wirkungsweise hier kurz skizziert werden. Es sei darauf hingewiesen, dass es für Sie zum jetzigen Zeitpunkt weder erforderlich noch möglich ist (sofern Sie zu den Programmiereinstei-

1 In den meisten Compiler ist der Präprozessor direkt integriert, tritt also gar nicht mehr als eigenständiges Programm auf. Der Linker hat sich dagegen seine Eigenständigkeit bewahrt. Er wird bei der Programmerstellung üblicherweise automatisch vom Compiler aufgerufen.

gern gehören), das Folgende in allen Einzelheiten zu verstehen. Genaueres werden Sie erfahren, wenn wir uns mit Funktionen beschäftigen.

Die Übersetzung von C++-Programmen (mit »Programm« ist hier natürlich der Quelltext gemeint) erfolgt in drei internen Schritten:

- Der Präprozessor entfernt eventuelle Kommentare aus dem Quelltext und führt bestimmte Textersetzungen durch.
- Der eigentliche Compiler übersetzt den so aufbereiteten Quelltext in Maschinensprache. Als Ergebnis entstehen eine oder mehrere so genannte Objektdateien mit der Erweiterung *.obj*.
- Im letzten Übersetzungsschritt fügt der Linker Code aus den Bibliotheken ein und bindet die *.obj*-Dateien, soweit es sich um mehrere handelt, zu einer ausführbaren Datei zusammen.

Hinweis

Wie oben angedeutet, ist es auch möglich, eine ausführbare Datei aus mehreren Quellcodedateien zu kompilieren. In diesem Fall erzeugt der Compiler für jede Quelldatei eine *.obj*-Datei (wie das geht, erfahren Sie ebenfalls in Kapitel 20 »Funktionen«).

1.3 Was versteht man unter einer IDE?

Wie Sie gerade erfahren haben, ist es für die Programmerstellung notwendig, Text in einen Editor einzugeben und diesen dann kompilieren zu lassen. Dazu kann jeder beliebige Texteditor benutzt und für den Übersetzungsvorgang ein externer Compiler herangezogen werden.

In modernen Entwicklungsumgebungen ist die Software zum Übersetzen des Quellcodes genauso wie ein mit nützlichen Funktionen ausgestatteter Editor als fester Bestandteil in ein umfassendes Softwareprogramm eingebunden. Dort wird der Compiler meist über einen entsprechenden Befehl in der Menüleiste oder noch einfacher mit einem Klick auf ein Symbol gestartet. Zudem werden in der Regel eine Reihe weiterer Hilfsmittel für die Programmentwicklung bereitgestellt.

Weil dort eben alle Programmierwerkzeuge unter einer Bedienoberfläche zusammengefasst sind, nennt man ein solches System »integrierte Entwicklungsumgebung« oder kurz IDE (für Integrated Development Environment).

Eine beliebte IDE für C++-Programme ist z.B. das Visual Studio von Microsoft. Dieses fungiert gleichzeitig als Entwicklungsumgebung für weitere Programmiersprachen wie z.B. Visual Basic und C#. Soweit nur die Features zur Entwicklung von C++-Programmen gemeint sind, spricht man von Visual C++ bzw. vom Visual C++-Compiler.

Hinweis

Bei der bereits erwähnten Visual C++ 2005 Express Edition handelt es sich gewissermaßen um eine Tochter des Visual Studio bzw. von Visual C++. Sie ist voll funktionsfähig und obendrein kostenfrei erhältlich.

Der Name »Visual« rührt daher, dass zusätzliche Funktionen zur Programmierung von grafischen Benutzeroberflächen bereitgestellt werden (visual – zu Deutsch »visuell«). Diese sind jedoch compilerspezifisch, gehören also nicht zum Standardumfang der Programmiersprache C++. Das heißt, ein Programm (gemeint ist der Quellcode), welches diese (compilerspezifischen) Funktionen verwendet, kann nicht auf einem anderen Compiler übersetzt werden – auch wenn es sich dabei um einen anderen C++-Compiler handelt.

Es sei hier ausdrücklich darauf hingewiesen, dass C++ ein Sprachstandard, also keine Entwicklungsumgebung, ist. Eben dieser Sprachstandard bildet das Thema dieses Buches. Wir werden in den Programmierbeispielen daher nur auf Features zurückgreifen, die von diesem Standard (dem so genannten ANSI/ISO-Standard) unterstützt werden.

Hinweis

Mit einer Ausnahme: Im letzten Teil dieses Buches werde ich Sie anhand einer mehr oder weniger umfangreichen Beispielanwendung mit der Programmierung von grafischen Benutzeroberflächen bekannt machen. Dieser Teil ist gewissermaßen als Bonus zu verstehen.

Prinzipiell können Sie also mit jedem beliebigen Compiler arbeiten, solange dieser den ANSI/ISO-Sprachstandard unterstützt. Im Übrigen wurden die Beispiele im Buch mit verschiedenen C++-Compilern getestet, u.a. mit Bloodshed, Visual C++ und der Kommandozeilenversion des Borland C++-Compilers. Ein Kommandozeilen-Compiler wird durch Eingabe eines Startbefehls in der Konsole in Gang gesetzt, ist also nicht Teil einer IDE.

1.4 Was sind Konsolenprogramme?

Das wirft die Frage auf, was unter einer Konsole zu verstehen ist. Zu alten DOS-Zeiten lief die Kommunikation zwischen Anwender und Computer allein über die Konsole, die den ganzen Bildschirm einnahm. Oder anders ausgedrückt: Der Bildschirm war die Konsole.

Der Computer nahm Eingaben des Benutzers (Befehle an das Betriebssystem, aber auch Startbefehle für Computerprogramme) an einer bestimmten Stelle des Bildschirms entgegen. In der Regel forderte dort das Blinken eines Cursors zur Eingabe auf. Diese Stelle wurde eben »Eingabeaufforderung«, »Befehlszeile« oder kurz »Prompt« genannt.

Falls am Prompt der Startbefehl für ein Programm eingegeben wurde (durch Eingabe der dafür notwendigen Zeichenfolge über die Tastatur und anschließendes Betätigen der ↵-Taste), übernahm dieses sozusagen die Konsole. Das heißt, es nutzte die Konsole – damals also den ganzen Bildschirm – für Mitteilungen an den Benutzer (»Geben Sie bitte … ein«, »Die von Ihnen zu zahlende Einkommensteuer beträgt …«) und um Daten vom Benutzer entgegenzunehmen.

Heutzutage verläuft die Kommunikation zwischen Anwender und Programm meist über grafische Benutzeroberflächen. Dies gilt auch für die Interaktion mit dem Betriebssystem. So bietet Windows verschiedene grafische Schnittstellen zum Starten von Computerprogrammen an: Eintrag im Startmenü, Symbol auf dem Desktop, Auflistung von ausführbaren Dateien im Windows-Explorer.

GUI

GUI ist die Abkürzung für Graphical User Interface, zu Deutsch grafische Benutzeroberfläche. Darunter versteht sich die Gesamtheit aller Oberflächenelemente wie Fenster, Menüleisten, Schaltflächen etc., welche für den Dialog mit dem Benutzer zur Verfügung stehen.

Das ändert aber nichts an der Tatsache, dass die Konsole nach wie vor zur Verfügung steht. Nur nimmt sie nicht mehr wie früher den ganzen Bildschirm ein. Außerdem erscheint sie nicht sofort beim Start des Computers, sondern ist ebenfalls über grafische Schnittstellen erreichbar.

Unter Windows-Betriebssystemen erreichen Sie die Konsole über Start/Programme/(Zubehör/)Eingabeaufforderung bzw. Start/Ausführen…. Bei der Variante Start/Ausführen… müssen Sie in dem dann angezeigten Dialogfeld zunächst `cmd` eingeben und mit einem Klick auf die Schaltfläche OK bestätigen.

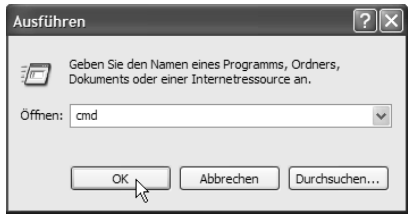

Abbildung 1.2: Dialog zum Aufrufen der Konsole

Hinweis

Weitere Bezeichnungen für die Konsole sind neben »Eingabeaufforderung«, »Befehlszeile« und »Prompt« auch »DOS-Fenster« und »DOS-Box«. Unter Linux heißt die Konsole auch »Terminal«.

Eine Anwendung wie beispielsweise das Tabellenkalkulationsprogramm Excel ließe sich auch über die Konsole starten. Falls der Verzeichnispfad zur zugehörigen Programmda-

tei *C:\Programme\Microsoft Office\OFFICE11* lautet, geben Sie an der Eingabeaufforderung nacheinander die Befehle

```
cd Programme
cd Microsoft Office
cd OFFICE11
```

ein, um in das Verzeichnis zu gelangen, in dem sich die ausführbare Datei *excel.exe* befindet. Diese starten Sie dann mit dem Befehl `excel` (Eingabe des Dateinamens ohne Erweiterung).

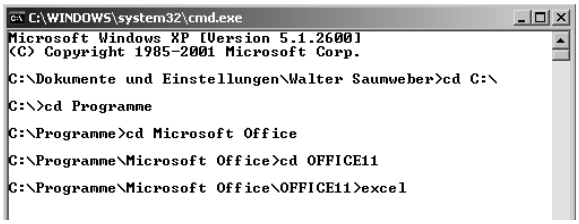

Abbildung 1.3: Starten von Excel über die Konsole

Hinweis

Es sei erneut darauf hingewiesen, dass der Startbefehl für ausführbare Programme vom verwendeten Betriebssystem abhängt. Unter Linux beispielsweise wird eine Datei mit dem Namen *xy.out* auch mit `xy.out` gestartet, also mit Angabe des vollständigen Dateinamens (einschließlich Dateierweiterung).

Auch wenn es (wie jedes andere Programm) von der Konsole aus gestartet werden kann, ist Excel dennoch ein Windows-Programm, da es ja selbst eine grafische Benutzeroberfläche zur Verfügung stellt.

Vielmehr zeichnet sich ein Konsolenprogramm dadurch aus, dass es eben keine GUI zur Verfügung stellt, sondern für die Interaktion mit dem Anwender ausschließlich die Konsole benutzt. Dies gilt auch für die Kommandozeilenversion des Borland C++-Compilers (das Wort »Kommandozeile« steht hier ebenfalls für die Konsole). Wie Ihnen ja bekannt ist, handelt es sich auch bei einem Compiler um nichts anderes als um ein ausführbares Computerprogramm.

Entscheidend ist also die Bereitstellung einer grafischen Benutzeroberfläche seitens des Programms. Von wo aus ein Programm gestartet wird – über eine grafische Benutzerschnittstelle oder durch Eingabe eines Startbefehls in der Konsole –, spielt für die Einordnung als Windows- bzw. Konsolenprogramm keine Rolle.

Abgesehen vom letzten Teil des Buches, werden wir uns bei der Programmierung auf Konsolenprogramme beschränken. Dies mag Sie zunächst etwas enttäuschen, ist aber tatsächlich von Vorteil. Ziel des Buches ist es schließlich, den Standard der Programmiersprache C++ zu erlernen. Die Konzentration auf andere Themen wäre dabei nur hinderlich.

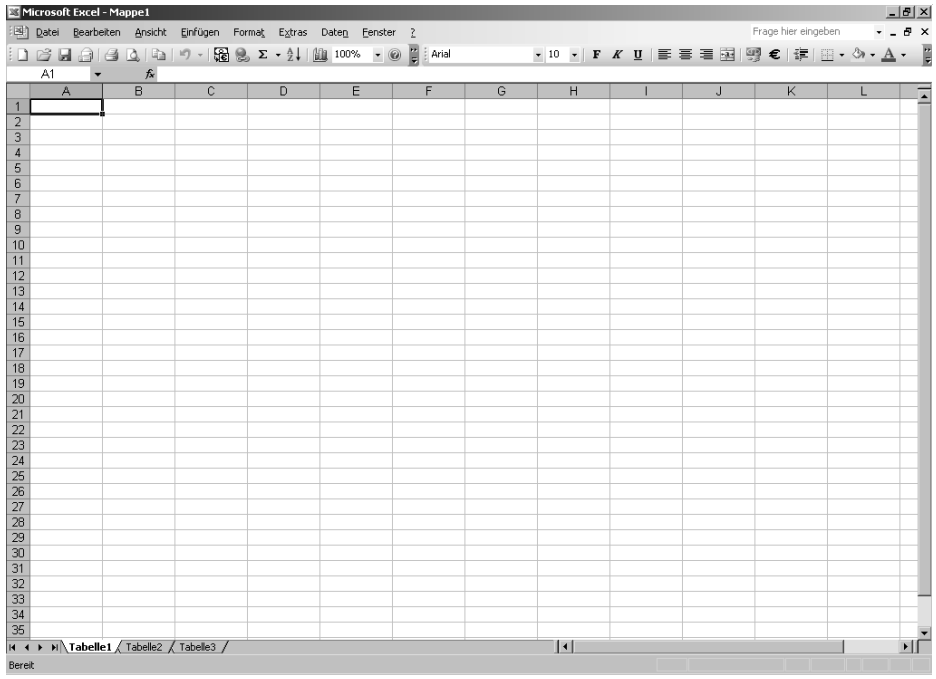

Abbildung 1.4: Die GUI von Excel

In diesem Zusammenhang sei darauf hingewiesen, dass die Kenntnisse, die Ihnen im Weiteren vermittelt werden (vor allem auf dem Gebiet der objektorientierten Programmierung), für ein tiefer gehendes Verständnis der Windows-Programmierung unabdingbar sind.

Weitere Gründe, um aus didaktischen Erwägungen von einer Beschäftigung mit der GUI-Programmierung vorerst abzusehen, sind:

- Windows-Programme besitzen den Nachteil der Plattformabhängigkeit, da sie an den auf einem System integrierten Window-Manager gebunden sind. Der Quelltext von Windows-Programmen ist also nicht portierbar (zur Portierbarkeit siehe oben). Mit anderen Worten: Jede IDE stellt spezielle Funktionen zur Programmierung von grafischen Benutzerschnittstellen zur Verfügung. Dagegen wird eine Funktion, die zum Standardsprachumfang von C++ gehört, von jedem Compiler unterstützt (ist das nicht der Fall, so handelt es sich wahrscheinlich um einen veralteten Compiler, und es empfiehlt sich der Umstieg auf eine neuere Compilerversion).
- Die Programmierung von GUIs ist in der Regel sehr aufwändig. Verschiedene IDEs, darunter auch das Visual Studio, können allerdings den Quellcode mehr oder weniger automatisch erzeugen – zum Teil erhält man dann für bestimmte Aufgaben mehrere Seiten entsprechend komplexen Codes.

Von den letztgenannten Möglichkeiten (die auch Visual C++ bietet) werden wir aber, wie gesagt, erst in den Bonuskapiteln Gebrauch machen. Selbstverständlich bleibt es Ihnen freigestellt, sich nach dem Lesen dieses Buches entsprechende Lektüre zu besorgen (z.B. spezielle Literatur über Visual Studio). Sie haben dann die besten Voraussetzungen dazu.

Des Weiteren sei darauf hingewiesen, dass grafische Benutzerschnittstellen zwar erhebliche Verbesserungen des Bedienkomforts mit sich bringen, die Leistungsfähigkeit von Programmen jedoch nicht erhöhen. Überdies sind auch heute noch eine Reihe von professionellen Programmen im Einsatz, bei denen grafische Schnittstellen keine Rolle spielen.

2 Welchen Compiler sollten Sie verwenden?

Bevor Sie Ihr erstes Programm schreiben, werden wir zunächst auf die Frage der benötigten Software eingehen. Einerseits ist natürlich auch die Wahl des Compilers, wie vieles andere, eine Frage des persönlichen Geschmacks. Auf der anderen Seite spielt es eine Rolle, ob Sie bereits Programmiererfahrung – in C++ oder möglicherweise in einer anderen Programmiersprache – mitbringen. Gerade wenn Sie zu den Programmiereinsteigern gehören, sollten Sie besonders darauf achten, dass der Compiler, den Sie verwenden, den ANSI/ISO-C++-Sprachstandard in vollem Umfang unterstützt. Des Weiteren rate ich Ihnen in diesem Fall, fürs Erste einen einfachen Kommandozeilen-Compiler zu verwenden. Natürlich bietet eine aufwändige Entwicklungsumgebung wie z.B. das Visual Studio von Microsoft viele Vorteile. Diese werden Sie aber erst richtig nutzen können, wenn Sie schon einiges Verständnis und Erfahrung in der Programmierung besitzen. Dem Anfänger wird die Komplexität einer modernen IDE jedoch eher zum Nachteil gereichen.

2.1 Borland C++-Compiler

Lesern, die unter einem Windows-Betriebssystem arbeiten, sei die Kommandozeilen-Version des C++-Compilers von Borland empfohlen. Sie finden ihn in der zur Zeit der Drucklegung aktuellen Version 5.5.1 auf der Buch-CD im Verzeichnis *Borland-Compiler* (Dateiname: *freecommandLinetools.exe*). Ansonsten können Sie den C++-Compiler über die Website *http://www.codegear.com* – klicken Sie hierzu auf den Link DOWNLOADS, dann auf C++BUILDER und anschließend auf COMPILER – oder gleich unter *http://www.codegear.com/tabid/139/Default.aspx*, Link COMPILER, kostenlos herunterladen. Der Kommandozeilen-Compiler von Borland ist zuverlässig und leistungsfähig. Außerdem unterstützt er den ANSI/ISO-C++-Sprachstandard in vollem Umfang. Die Installation wird weiter unten beschrieben. Insbesondere werden wir die einzelnen Schritte aufzeigen, die für das Übersetzen unseres ersten Programms speziell mit diesem Compiler nötig sind.

Für Leser, die mit einem anderen Compiler arbeiten, ist es leider unerlässlich, sich mit dessen Bedienung bis zu einem gewissen Grad selbständig vertraut zu machen. Bemühen Sie gegebenenfalls die Dokumentation Ihres Compilers. Damit sich diese Lesergruppe nicht benachteiligt fühlt, sei noch einmal vermerkt, dass der Fokus dieses Buches auf dem Erlernen der Programmiersprache C++ liegt, nicht auf der Beschreibung einer bestimmten IDE. Zudem sind die Anleitungen, die Sie in Kapitel 4 im Zusammenhang mit unserem ersten Programm in Bezug auf den Borland-Compiler erhalten, mehr oder weniger auch auf andere Compiler anwendbar. Mit Beginn des zweiten Teils wird eine gewisse Vertrautheit mit der von Ihnen verwendeten Software vorausgesetzt, sodass in dieser Hinsicht alle Leser gleich behandelt werden.

Im Übrigen ist es nicht immer zwingend notwendig, dass Sie am Computer sitzen, wenn Sie dieses Buch lesen (manche Leser mögen dies generell als unangenehm empfinden). Die meisten Beispiele sollten Sie auch ohne direkten Rechnerkontakt gut verstehen können. Praktische Übung empfiehlt sich natürlich in jedem Fall.

2.2 Wie finde ich einen passenden Compiler?

Diese Frage stellt sich natürlich nur, wenn Sie nicht mit dem oben genannten Borland-Compiler arbeiten können oder aus einem bestimmten Grund nicht wollen. Leider ist der Borland C++-Compiler nicht für Linux-Betriebssysteme verfügbar. In aller Regel ist jedoch dort ein C++-Kommandozeilen-Compiler (GCC) bereits vorinstalliert, sodass Sie diesen verwenden können. Er gewährleistet ebenfalls eine ausreichende Unterstützung des ANSI/ISO-C++-Sprachstandards. Im Zuge der Ausführungen zu unserem ersten Programm werden wir auch zum GCC-Compiler einige Hinweise geben.

Tipp

Noch ein Tipp, der möglicherweise zu einem späteren Zeitpunkt für Sie von Interesse sein mag: DJGPP, eine sehr aufwändig zu installierende, aber sehr leistungsfähige IDE, wird von der Firma delorie software auf der Website

http://www.delorie.com/djgpp

kostenlos zum Download angeboten. Die Informationen sind allerdings in Englisch gehalten, und es ist trotz umfangreicher Dokumentation nicht unbedingt leicht, sich zurechtzufinden.

Die Standardversion von DJGPP ist auch auf der deutschsprachigen Website

http://c.theflow.de/djgpp.htm

erhältlich. Dort finden Sie auch eine ausführliche Installationsbeschreibung und weitere interessante Links zu Tutorials, Compilern etc.

Leider ist uns zum Zeitpunkt der Drucklegung dieses Buches kein frei erhältlicher C++-Compiler für den Macintosh bekannt. Möglicherweise werden Sie auf der Suche nach einem passenden C++-Compiler für den Mac jetzt oder später auf den Entwickler-Seiten von Apple fündig (*http://developer.apple.com/de*). Ansonsten bleibt Lesern, die mit einem Macintosh arbeiten, nichts anderes übrig, als gegebenenfalls einen passenden Compiler käuflich zu erwerben (z.B. Symantec C++) und sich mit der Bedienung hinreichend vertraut zu machen. Aber wie schon gesagt, Thema dieses Buches ist die Programmiersprache C++ und nicht die Beschreibung eines Compilers.