heiko KALISTA

Für WINDOWS und für MAC

C++

FÜR SPIELE-PROGRAMMIERER

4. Auflage

HANSER

Kalista

C++ für Spieleprogrammierer

Bleiben Sie auf dem Laufenden!

Der Hanser Computerbuch-Newsletter informiert Sie regelmäßig über neue Bücher und Termine aus den verschiedenen Bereichen der IT. Profitieren Sie auch von Gewinnspielen und exklusiven Leseproben. Gleich anmelden unter

www.hanser-fachbuch.de/newsletter

Heiko Kalista

C++ für Spieleprogrammierer

4., aktualisierte Auflage

HANSER

Der Autor:
Heiko „The Wanderer“ Kalista, Rodgau

Alle in diesem Buch enthaltenen Informationen, Verfahren und Darstellungen wurden nach bestem Wissen zusammengestellt und mit Sorgfalt getestet. Dennoch sind Fehler nicht ganz auszuschließen. Aus diesem Grund sind die im vorliegenden Buch enthaltenen Informationen mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autor und Verlag übernehmen infolgedessen keine juristische Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieser Informationen – oder Teilen davon – entsteht.

Ebenso übernehmen Autor und Verlag keine Gewähr dafür, dass beschriebene Verfahren usw. frei von Schutzrechten Dritter sind. Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Buch berechtigt deshalb auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

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.

Dieses Werk ist urheberrechtlich geschützt.
Alle Rechte, auch die der Übersetzung, des Nachdruckes und der Vervielfältigung des Buches, oder Teilen daraus, vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung des Verlages in irgendeiner Form (Fotokopie, Mikrofilm oder ein anderes Verfahren) – auch nicht für Zwecke der Unterrichtsgestaltung – reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

© 2013 Carl Hanser Verlag München, www.hanser-fachbuch.de
Lektorat: Sieglinde Schärl
Herstellung: Irene Weilhart
Umschlagdesign: Marc Müller-Bremer, www.rebranding.de, München
Umschlagrealisation: Stephan Rönigk
Gesamtherstellung: Kösel, Krugzell
Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702
Printed in Germany

print-ISBN: 978-3-446-43216-1
e-book-ISBN: 978-3-446-43395-3

Für Naomi

Jede Sekunde ist eine Bereicherung …
Jede Stunde ist ein Geschenk …
Jeder Tag ist etwas Besonderes …

… weil Du an meiner Seite bist

Du hast mir gezeigt, dass der Herbst nicht kalt, sondern farbenprächtig ist.
Du hast mir gezeigt, dass der Winter nicht dunkel, sondern klar und schön ist.
Du hast mir gezeigt, wie man die Dinge auf eine bessere Weise sehen kann.

Inhalt

Vorwort

Hallo und herzlich willkommen! Du möchtest also gerne das Spieleprogrammieren lernen und hast Dich für dieses Buch entschieden. Genau jetzt, wenn Du diese Zeilen liest, befindest Du Dich auf den ersten Metern eines langen Weges, den es sich jedoch definitiv lohnt zu gehen.

Vor über zwei Jahrzehnten brach eine neue Ära an, als die Homecomputer die heimischen Wohnzimmer eroberten. Recht schnell fanden viele Leute Gefallen daran, nicht nur fertige Computerspiele zu spielen, sondern selbst herauszufinden, wie man solche eigenständig entwickeln kann. „Herauszufinden" trifft dabei den Nagel auf den Kopf, denn Medien wie das Internet waren zu dieser Zeit nicht vorhanden und Buchmaterial war kaum bis gar nicht in den Regalen der Buchhandlungen zu finden. Aus diesem Grund trennte sich schnell die Spreu vom Weizen, und manche gaben auf. Manche klemmten sich jedoch so lange dahinter, bis die gewünschten Resultate über den Bildschirm flackerten. Genau dieser „Forschergeist" ist es, der einen Spieleprogrammierer ausmacht. Man muss Geduld haben, bereit sein zu lernen und einen unermüdlichen Drang besitzen, sein Vorhaben in die Tat umzusetzen. Dazu könnte es keine bessere Zeit geben als die heutige. Im Internet gibt es eine Fülle von Informationen, Hilfestellungen und sogar ganze Communities, die sich mit dem Thema Spieleprogrammierung befassen. Früher bestanden die Quelltexte (Programmcodes) von Spielen noch aus einer endlosen Reihe von kryptischen, meist aus drei Buchstaben bestehenden Befehlen und unübersichtlichen Sprunganweisungen. Heute hingegen hat man mit C++ eine mächtige Hochsprache in der Hand, die leicht zu erlernen ist und viele Dinge einfacher und logischer macht. Informationen sind leichter zugänglich, und Spieleprogrammierer sind längst keine kleine Gruppe mehr, die sich gut gehütete Geheimnisse teilt. Heute ist es jedem möglich, sich selbst mit dieser Thematik zu befassen, solange er bereit ist, sich intensiv damit auseinander zu setzen.

Sicherlich ist mit der wachsenden Leistung der Computer und den immer höher werdenden Anforderungen an das System auch die Komplexität gestiegen. Dabei bleiben jedoch bestimmte Grundlagen immer gleich. Wer schon eine Programmiersprache beherrscht oder sich generell bereits mit der Thematik des Programmierens befasst hat, wird hier einige Dinge antreffen, die ihm bekannt vorkommen werden. Trotzdem ist Stillstand der größte Feind eines Spieleprogrammierers. Deshalb ist es enorm wichtig, die mittlerweile unglaublich ergiebige Informationsflut im Internet zu nutzen und immer auf dem neuesten Stand zu bleiben. Schneller als je zuvor kann man sich heute über das Internet die gewünschten Informationen beschaffen. Jedoch birgt diese Möglichkeit der Informationssuche auch eine

Gefahr: Man lässt sich schnell verleiten, den eigenen Kopf zugunsten einer Suchmaschine auszuschalten. Deshalb möchte ich an dieser Stelle unbedingt darauf hinweisen, dass man zuerst selbst versuchen sollte, ein Problem zu lösen, und erst dann auf Diskussionsforen oder Ähnliches zugreift, wenn man selbst wirklich nicht mehr anders weiterkommt. Diese Vorgehensweise ist enorm wichtig, um auf lange Sicht Erfolg zu haben. Oftmals findet man selbst eine bessere oder effektivere Lösung, wenn man darüber nachdenkt, statt andere für sich denken zu lassen. Nutze deshalb das Internet als Werkzeug und nicht, um den eigenen Kopf zu ersetzen.

Viele Bücher über die Spieleprogrammierung setzen an einem Punkt an, an dem schon weitreichende Vorkenntnisse der Programmiersprache C++ gefordert sind, oder bieten nur einen kleinen Crash-Kurs im Programmieren an. Andere Bücher befassen sich zwar mit der Programmiersprache C++, ohne dabei jedoch einen Bezug zur Spieleprogrammierung herzustellen. Dieses Buch versucht nun, diese Lücke zu schließen und den Grundstein zu legen, der auf dem Weg zum Spieleprogrammierer nötig ist: Das Erlernen der Programmiersprache C++. Nach dem Durcharbeiten dieses Buches solltest Du also über das essenzielle Grundlagenwissen verfügen, das nötig ist, um erfolgreich Spiele zu entwickeln.

In diesem Sinne: Let's code!

Anmerkungen zur vierten Auflage

Während sich C++ im Laufe der Jahre kaum verändert hat, ändern sich die verschiedenen Entwicklungsumgebungen umso rasanter. Ständig gibt es neue, bessere Tools mit immer mächtigeren Funktionen. Welche Entwicklungsumgebung man benutzt, hängt nicht nur vom verwendeten Betriebssystem ab, sondern auch vom persönlichen Geschmack und den an das Projekt gestellten Anforderungen.

Seit dem Erscheinen der ersten Auflage dieses Buches bis zur aktuellen, vierten Auflage hat sich einiges getan. Microsoft erweitert und erneuert ständig die Entwicklungsumgebung Visual Studio. Während in der ersten Auflage noch Visual Studio 6.0 verwendet wurde, wurden in den späteren Auflagen auf die Express-Edition von Visual Studio umgestellt. Dabei handelt es sich um eine im Funktionsumfang leicht abgespeckte Version von Visual Studio, die dafür allerdings kostenlos verwendet werden kann. Man muss sich lediglich bei Microsoft registrieren. Für unsere Zwecke (und auch darüber hinaus) ist diese Express-Edition vollkommen ausreichend. Damit haben wir ein mächtiges Werkzeug an der Hand, mit dem sämtliche in diesem Buch abgedruckten Beispiele kompiliert und ausgeführt werden können.

Die bisherigen Auflagen erklärten ausschließlich die C++-Entwicklung auf Systemen mit einem Windows-Betriebssystem. Ich habe mich dazu entschieden, in dieser Auflage auch auf Apple-Betriebssysteme einzugehen, da diese eine immer größere Verbreitung finden und stetig an Beliebtheit gewinnen. Die dabei verwendete Entwicklungsumgebung nennt sich Xcode 4.5.2 und kann auf Apple-Systemen kostenlos heruntergeladen werden.

Hierbei sind allerdings einige Einschränkungen in Kauf zu nehmen, da bestimmte Beispiele in diesem Buch Microsoft-spezifische Funktionen verwenden und somit nicht systemunabhängig sind. Natürlich weise ich an den entsprechenden Stellen darauf hin, wenn ein Beispiel nicht, oder nur mit Einschränkungen auf einem Apple-System funktioniert.

Dazu muss gesagt werden, dass auf Apple-Systemen die Programmiersprache Objective-C vorherrscht. Da dieses Buch sich jedoch nicht der Spiele-Entwicklung für ein bestimmtes System, sondern vielmehr dem allgemeinen Erlernen von C++ widmet, ist diese Tatsache nicht weiter tragisch. Auch wenn auf Apple-Systemen Objective-C das Mittel zur Wahl ist, sind C++-Kenntnisse von Vorteil, da es sogar möglich ist, beide Sprachen zu mischen. Weiterhin ist es natürlich trotzdem möglich, komplett in C++ zu entwickeln.

Im Grunde wurde diese Entscheidung getroffen, um Dir eine weitere Möglichkeit und somit mehr Freiheit in der Wahl der von Dir bevorzugten Werkzeuge an die Hand zu geben.

Da wir uns in diesem Buch hauptsächlich mit dem Erlernen der Programmiersprache C++ beschäftigen, werde ich nicht auf alle Funktionen und Möglichkeiten der Microsoft Visual Studio 2012 Express Edition, respektive Xcode 4.5.2 eingehen. Natürlich wird überall dort, wo es nötig ist, genau erklärt, wie man diese Entwicklungsumgebungen bedient und wie man vorgehen muss. Das ist etwa dann der Fall, wenn wir uns unseren ersten Arbeitsbereich erstellen, später den Debugger kennenlernen oder gegen Ende des Buches spezielle Projekteinstellungen vornehmen.

Sicherlich gibt es viele Leute, die eine komplett andere Entwicklungsumgebung benutzen möchten. Natürlich spricht nichts dagegen – im Gegenteil: Wer sich mit mehreren verschiedenen Entwicklungsumgebungen auskennt, kann sich später schneller in ein bestehendes Projekt eines Teams einarbeiten. Allerdings sollte es verständlich sein, dass ich in diesem Buch nicht auf alle möglichen Entwicklungsumgebungen eingehen kann. Der Quelltext wurde so geschrieben, dass er im Grunde ohne Änderungen mit anderen Entwicklungsumgebungen kompiliert werden kann. Wie man in diesem Fall einen neuen Arbeitsbereich anlegt oder den Debugger verwendet, muss dann in den entsprechenden Dokumentationen und Anleitungen nachgelesen werden.

Danksagung

Bevor es nun losgeht, möchte ich mich an dieser Stelle bei allen Leuten bedanken, die mich beim Schreiben dieses Buches unterstützt haben und mithalfen, es zu verwirklichen. Marc Kamradt und Jörg Winterstein haben sich mühsam durch alle Kapitel geackert und mir überall dort auf die Finger geklopft, wo es angebracht war. Jörg hat es dabei immer wieder geschafft, mich mit seinem unverwechselbaren Humor in seinen Kommentaren zum Lachen zu bringen. Ohne Eure Hilfe wäre dieses Buch nicht zustande gekommen. Danke, Jungs!

Für die Grafiken des kleinen Demo-Spiels möchte ich mich bei Thomas Schreiter und seiner gestalterischen Zauberhand bedanken, die mich immer wieder verblüfft.

Matthias Gall und Mathias Ricken hatten immer dann ein offenes Ohr, wenn eine Frage auftauchte und beantwortet werden musste.

Peter Schraut und David Scherfgen haben sich ebenfalls einige Kapitel vorgenommen und mit vielen nützlichen Hinweisen und Vorschlägen zu diesem Buch beigetragen. Euch beiden ein dickes Dankeschön hierfür. Und David: öle Deine Maus und mach die Tastatur startklar … bald geht ein gewisses Spiel in eine neue Runde. Ach ja, Peter: warum liegt denn da Stroh rum?

Fernando Schneider vom Hanser Verlag hat mir mit Rat und Tat zur Seite gestanden, wenn es um dieses Buchprojekt ging, und uns auf der Dusmania hilfreich unter die Arme gegriffen. Ich hoffe, dass es Dir gut geht und wir uns mal wieder sehen!

Weiterhin möchte ich mich bei Sieglinde Schärl, Julia Stepp und Irene Weilhart sowie allen anderen Mitarbeitern des Hanser Verlages für die gute Zusammenarbeit bedanken. Bei Sieglinde Schärl möchte ich mich vor allem für ihre große Geduld bedanken, denn mein ständiges „Ja, nein, vielleicht, vielleicht doch nicht“ bezüglich der vierten Auflage war sicherlich nicht einfach.

Der größte Dank gilt meiner Mutter und allen Freunden, die daran geglaubt haben, dass die Idee dieses Buches auch in die Tat umgesetzt wird.

Spezieller Dank geht an Marc Grimm, dafür, dass er da ist, und an Oliver Strenge, dafür, dass er wieder da ist.

Dreieich, im April 2013

Heiko Kalista

1 Grundlagen

1.1 Einleitung

1.1.1 An wen richtet sich dieses Buch?

Dieses Buch ist für alle gedacht, die sich für die Thematik der Spieleprogrammierung interessieren und noch keinerlei Vorkenntnisse haben. Der Schwierigkeitsgrad beginnt bei null und steigert sich langsam, aber stetig im Verlauf der einzelnen Kapitel. Weiterführende Bereiche wie etwa die Grafikprogrammierung werden in diesem Buch zwar auch behandelt, aber für dieses Thema ist „nur“ ein Kapitel vorgesehen, das Dich ein wenig tiefer in die Materie einführen soll. Das Buch bezieht sich ausschließlich auf die Programmiersprache C++ und fokussiert dabei die Spieleprogrammierung.

Im Grunde spricht jedoch nichts dagegen, dieses Buch als Nachschlagewerk zu verwenden. Generell ist es für jeden nützlich, der die Programmiersprache C++ erlernen oder vertiefen will.

1.1.2 Welche Vorkenntnisse werden benötigt?

Um mit diesem Buch arbeiten zu können, werden keinerlei Programmierkenntnisse benötigt. Das Einzige, was man beherrschen sollte, ist der Umgang mit Windows oder Mac OS X. Wer weiß, wie man einen Computer einschaltet, Programme startet und mit Dateien arbeitet, ist fast schon überqualifiziert. Es wird erklärt, welche Programme man benötigt und wie man mit ihnen Quelltexte (Programmcode) erstellt, diese kompiliert (in eine für den Computer verständliche Sprache verwandelt) und schließlich lauffähige Programme erzeugt.

Solltest Du bereits über Programmierkenntnisse einer beliebigen anderen Sprache verfügen, so wird dies sicherlich nützlich sein, ist aber – wie schon erwähnt – nicht notwendig.

1.1.3 Wie arbeitet man am effektivsten mit diesem Buch?

Das Buch ist so gestaltet, dass die einzelnen Kapitel aufeinander aufbauen. Jedes Kapitel erfordert es, dass man die Themen aus den vorangegangenen Kapiteln durchgearbeitet und auch verstanden hat.

In den meisten Kapiteln wird es ein paar Aufgabenstellungen geben, die helfen sollen, das bisher Gelernte zu festigen. Dabei werden zu jeder einzelnen Aufgabe verschiedene Tipps gegeben, wie man sich am besten an die Lösung heranmacht. Es soll keinesfalls der Eindruck entstehen, dass es sich wie in der Schule um Hausaufgaben handelt. Vielmehr soll Dir hier die Möglichkeit gegeben werden, selbst zu kontrollieren, ob Du die bisher erklärten Themen wirklich verstanden hast. Praxis ist und bleibt eben ein wichtiger Aspekt, wenn nicht gar der wichtigste.

Am besten schaust Du Dir die jeweilige Aufgabe an und liest die dazugehörigen Tipps durch. Danach überlegst Du, wie man am besten diese Aufgabe lösen könnte. Gib nicht gleich auf, sondern versuche wirklich, zum Ziel zu kommen. Der Schwierigkeitsgrad ist so gewählt, dass man ohne Weiteres zur Lösung kommt, wenn man die vorhergehenden Kapitel gelesen und verstanden hat.

Direkt im Anschluss erfolgt eine Musterlösung, die Du Dir auch dann anschauen solltest, wenn Du die Aufgabe erfolgreich gelöst hast. Auf keinen Fall solltest Du Dir diese Lösung anschauen, bevor Du nicht selbst versucht hast, die Aufgabe zu lösen. Es ist ein gewaltiger Unterschied, ob man sagt „ja klar, so hätte ich das auch gemacht“ oder ob man es tatsächlich erst mal selbst versucht. Es ist zwar nicht zwingend erforderlich, sich mit diesen Aufgaben zu beschäftigen, um die nächsten Kapitel zu verstehen, dennoch ist Eigeninitiative der Schlüssel zum Erfolg.

Zu gegebener Zeit wird es auch sogenannte Fehlerquelltexte geben. Das sind kleine Programmbeispiele, die typische Fehler enthalten, wie sie häufig vorkommen. Das Ganze hat den Sinn, ein Gespür dafür zu entwickeln, wo überall etwas schiefgehen kann, welche Arten von Fehlern es gibt (Compilerfehler, Laufzeitfehler, Warnungen). Jeder wird früher oder später selbst mal die vertracktesten Fehler in seinen Quelltext einbauen und sich dann kopfkratzend auf die Suche danach machen. Da es gerade am Anfang nicht immer gleich ersichtlich ist, wo ein Fehler stecken könnte, halte ich es für sinnvoll, so früh wie möglich darauf einzugehen, wie man in einem solchen Fehlerfall am besten vorgeht. Die Fehlerquelltexte in diesem Buch enthalten diverse kleine Gemeinheiten und natürlich auch Hilfestellungen, wie man sich am besten auf die Suche begibt. Auch ein noch so erfahrener Programmierer ist nicht vor solchen Dingen gefeit. Allerdings kommt mit der Zeit die Erfahrung, solche Fehler schneller einzugrenzen.

Wie auch schon bei den Aufgabenstellungen wird es im Anschluss eine korrigierte Fassung des Quelltextes geben. Dabei wird erklärt, wie man den einzelnen Fehlern schnellstmöglich auf die Schliche kommt und wie man sie hätte vermeiden können. Diese Übungen sind fast noch wichtiger als die Aufgabenstellungen, da es sehr ärgerlich ist, einen Quelltext, den man in fünf Minuten geschrieben hat, zwei Stunden lang nach Fehlern zu durchforsten.

1.1.4 Geduld, Motivation und gelegentliche Tiefschläge

Ich möchte hier nicht um den heißen Brei herumreden, aber gelegentliche Tiefschläge gehören nun leider dazu. Früher oder später kommt jeder mal an den Punkt, an dem einfach nichts klappen will und das Programm nicht das macht, was es eigentlich soll. Genau an diesem Punkt gibt es zwei Wege, die angehende Programmierer einschlagen können: Die einen geben frustriert auf und werfen den Compiler samt PC und Monitor aus dem Fenster.

Danach wird mächtig über die Programmiersprache geschimpft: Sie ist zu schwer, zu kompliziert, man muss ja studiert haben, um das zu verstehen und so weiter.

Die anderen jedoch werfen nicht gleich die Flinte ins Korn und bleiben stattdessen hartnäckig am Ball. Natürlich ist das nicht immer einfach, jedoch gibt es auch hier Tricks, wie man sich ganz schnell wieder motiviert. Manchmal ist es ganz nützlich, den Compiler auszuschalten und sich mit anderen Dingen zu beschäftigen, sei es nun an den See zu gehen oder ein paar coole Spiele mit bombastischen Effekten zu spielen. Gerade wenn man sich auf den Homepages von Hobbyentwicklern umschaut und sich deren Resultate ansieht, ist man schnell wieder motiviert und denkt sich: „Hey, die haben das doch auch hinbekommen, warum sollte mir das nicht auch gelingen?“

Man sollte sich auch vor Augen halten, dass die heute professionellen Entwickler auch nichts in den Schoß gelegt bekommen haben. Alle haben sie mit einem winzig kleinen Programm angefangen, das ein simples „Hallo Welt“ auf dem Bildschirm ausgibt. Damit hat es sogar etwas ganz Besonderes auf sich: Irgendwie ist es zur Gewohnheit geworden, dass viele Bücher über diverse Programmiersprachen mit einem Beispiel beginnen, das den Text „Hallo Welt“ auf dem Bildschirm ausgibt. Das geht nun sogar so weit, dass man mit Microsoft Visual C++-Arbeitsumgebungen automatisch ein solches Beispielprogramm erzeugen kann.

Wenn man also an einem solchen Tiefpunkt angelangt ist, sollte man definitiv nicht von „Scheitern“ reden. Es scheitert nur der, der aufgibt. Es gibt im Internet eine ganze Fülle von Diskussionsforen und Chats, in denen sich Programmierer austauschen und gegenseitig helfen können. Scheue Dich nicht davor, dort Fragen zu stellen. Schneller als Du denkst, wirst Du nicht nur Fragen stellen, sondern sie auch beantworten. Außerdem werden hier Tipps gegeben, wie man am schnellsten in die sogenannte „Szene“ einsteigen kann und was es zu beachten gibt. Es ist also keine schlechte Idee, Kapitel 13 schon etwas früher aufzuschlagen.

Mir ist es nun schon einige Male passiert, dass mich jemand darum gebeten hat, mal seinen Quelltext durchzuschauen und nachzusehen, wo der Fehler liegt. Oft war es so, dass ich den Fehler nach recht kurzer Zeit gefunden habe. Wenn man dann gefragt wird, wie man so schnell das Problem eingegrenzt hat, gibt es eigentlich nur eine einzige Antwort: Ich habe diesen Fehler oft genug selbst gemacht! Zwar habe ich zu diesem Zeitpunkt auch mit dem Gedanken gespielt, der CPU jedes Beinchen einzeln auszureißen, Kaffee in die Tastatur zu kippen oder die Aerodynamik meines Monitors im freien Fall zu testen. Da so etwas aber auf die Dauer recht teuer wird, ist es sinnvoller, sich mit dem Problem zu befassen, um es letzten Endes auch zu lösen.

Das alles hört sich schlimmer an, als es ist, aber ich finde es wichtig, auch auf diese Dinge hinzuweisen. Lass Dich auf keinen Fall dadurch entmutigen, sondern behalte das eben Gesagte einfach im Hinterkopf – es kann und wird sich als nützlich erweisen.

1.1.5 Das Begleitmaterial zum Buch

Unter *http://downloads.hanser.de* findest Du sämtliche Quelltexte aus dem Buch. Wie Du diese Beispiele mit Deiner Entwicklungsumgebung laden und verwenden kannst, erkläre ich Dir ab Abschnitt 1.4.3.

Alle Quelltexte wurden mit Microsoft Visual Studio 2012 Express Edition getestet und kompiliert. Die Beispiele 10.1 und 10.3 aus Kapitel 10 sind unter Mac OS X mit Xcode nicht lauffähig, da sie Windows-spezifische Funktionen verwenden. Diese speziellen Beispiele sind somit nur als Visual Studio 2012-Projekte, nicht jedoch als Xcode-Projekte vorhanden.

Die jeweils aktuellste Version von Visual Studio 2012 Express Edition findest Du unter *http://www.microsoft.com/visualstudio/deu/downloads*. Von den dort angebotenen Versionen ist in der Regel die Version „Visual Studio 2012 Express für Windows Desktop“ die Richtige. Du kannst frei wählen, ob Du das Gesamtpaket herunterladen möchtest, oder lieber einen Installer, der sämtliche benötigten Dateien bei Bedarf herunterlädt. Wie bereits erwähnt ist diese Version kostenlos. Nach Ablauf des Testzeitraums ist jedoch eine (ebenfalls kostenlose) Registrierung nötig.

Wenn Du unter Mac OS X arbeitest, dann kannst Du Dir Xcode kostenlos im App Store herunterladen.

Alle für das Buch notwendigen Materialien und Software-Tools findest Du auf den folgenden Internetseiten:

http://downloads.hanser.de

http://www.microsoft.com/visualstudio/deu/downloads

1.1.6 Fragen zum Buch

Wenn Du rund um das Buch Fragen, Kritik oder Anregungen hast, dann schaue auf der Seite *www.spieleprogrammierer.de* vorbei. Dort findest Du ein Diskussionsforum, Neuigkeiten über das Buch und vieles mehr.

1.2 Die Programmiersprache C++

Nach dieser kleinen Einleitung möchte ich Dir nun einen kurzen Einblick in die Entstehungsgeschichte der Programmiersprache C++ geben. Es soll klar werden, welche Idee hinter dieser Sprache steckt und was man unter objektorientiertem Programmieren versteht. Begriffe wie „OO“ oder „ANSI-Standard“ sollten nach diesem Kapitel zumindest keine böhmischen Dörfer mehr sein, auch wenn böhmische Dörfer durchaus sehr hübsch sind.

1.2.1 Von Lochkarten zu C++

C++ ist eine sogenannte Hochsprache. Um zu verstehen, was eine Hochsprache eigentlich ist, muss man ein klein wenig die Zeit zurückdrehen und sich die Programmiersprachen der früheren Tage anschauen. Zu den Zeiten des Commodore 64 und des Amigas (beide waren die bekanntesten und beliebtesten Homecomputer der 80er- und 90er-Jahre) war Assembler noch die vorherrschende Sprache. Assembler ist eine sogenannte „maschinennahe“ Sprache und zeichnet sich durch einen vergleichsweise geringen Befehlssatz aus. Das bedeutet, dass die zur Verfügung stehenden Befehle nicht gerade mächtig sind. Aus diesem Grund benötigt man sehr viele von ihnen, was zur Unübersichtlichkeit und Komplexität beiträgt. Man hat in der Regel nur Rechenoperationen zur Verfügung und Befehle, um Speicherstellen zu beschreiben oder auszulesen. Weiterhin sind solche Assembler-Programme mit Sprungbefehlen durchsetzt, was die Lesbarkeit noch um einiges verschlechtert. Der Assembler-Befehlssatz des Commodore C64 bot nicht einmal die Möglichkeit, zwei Zahlen einfach so zu multiplizieren oder zu dividieren.

Glücklicherweise haben sich die Zeiten geändert, und die Programmiersprachen sind weitaus komfortabler geworden. Während man früher den Computer mit Lochkarten gefüttert hat, um ihn zu programmieren, stehen einem heute mächtige und sehr leistungsfähige Programmiersprachen zur Verfügung. Eine der Etappen zwischen reiner Maschinensprache und C++ waren sogenannte Interpreter, die es bis heute gibt. Die Programmiersprache Basic vom Commodore C64 wäre hier als Beispiel zu nennen. Man verfügte über einen größeren Befehlssatz und konnte gewisse Aufgaben wesentlich einfacher und schneller bewältigen. Die Ausgabe von Grafik, Text und Musik war um einiges leichter. Dass es sich um einen Interpreter handelte, bedeutete allerdings, dass das Basic-Programm zur Laufzeit – also während das Programm ausgeführt wird – Zeile für Zeile in Maschinencode umgewandelt wurde (jede Hochsprache muss generell in Maschinencode umgewandelt werden). Dieser Vorgang kostete natürlich eine ganze Menge Zeit, und flüssig laufende Spiele konnte man mit Basic kaum erstellen.

C++ ist nun eine Sprache, die nicht mehr zur Laufzeit in Maschinencode umgewandelt (interpretiert) wird. Stattdessen erledigt ein sogenannter Compiler diese Arbeit schon vor dem Ausführen des eigentlichen Programms. Natürlich besteht das endgültige, ausführbare Programm immer noch aus einer schier endlos langen Folge von Nullen und Einsen. Jedoch brauchen wir nicht mit dem Locher vor dem PC zu sitzen und uns unsere Spiele aus Papierstreifen zu erstellen, der Compiler nimmt uns diese Arbeit ab. Der gesamte C++-Quelltext wird also in einen für den Computer verständlichen Maschinencode verwandelt, und eine .exe-Datei (ausführbare Datei) entsteht. Der Vorteil liegt auf der Hand: Man hat eine für den Menschen leicht lesbare, logisch aufgebaute Programmiersprache, die vom Compiler in rasanten Maschinencode umgewandelt wird. Wir brauchen uns keine Sorgen mehr zu machen, dass die Geschwindigkeit stark beeinträchtigt wird, und gleichzeitig sind wir davon entbunden, uns mit kryptischen Maschinenbefehlen herumzuschlagen.

Allerdings gab es noch einen Vorgänger der Programmiersprache C++, nämlich einfach nur „C“. C++ ist nun nicht eine völlig neue Sprache, sondern eine Erweiterung von C. Der eigentliche und wichtigste Unterschied liegt darin, dass C++ „objektorientiert“ ist. Was dies genau bedeutet, klären wir im nächsten Abschnitt.

Trotz all diesen tollen Vorteilen schreibt sich ein Programm natürlich nicht von selbst, und es wird schon gar nicht von alleine strukturiert, ordentlich und lesbar werden. Letztendlich liegen diese Dinge immer noch beim Programmierer selbst, und das wird wohl auch immer so bleiben. Natürlich kann man sämtliche Regeln des guten Programmierstils außer Acht lassen und weiterhin „Spaghetti-Code“, sprich unlesbaren, wirren Code, schreiben. Doch dies liegt dann am Programmierer und nicht an der Programmiersprache.

1.2.2 Objektorientiertes Programmieren

Um zu verstehen, was objektorientierte Programmiersprachen ausmachen, muss man noch mal eine kleine Zeitreise machen. Wie vorangehend schon erwähnt, bestanden Quelltexte früher meist aus Rechenoperationen, Vergleichsanweisungen und Sprungbefehlen. Wenn etwa geprüft werden sollte, ob ein gegnerisches Raumschiff noch über genügend Energie verfügt, mussten Speicherbereiche ausgelesen und ausgewertet werden. Je nach Inhalt dieser Speicherstellen wurde dann an andere Stellen im Programm verzweigt. In kleineren Programmen war dies nicht unbedingt dramatisch, jedoch raufte man sich bei großen Projekten meist die Haare, wenn man sämtliche Verzweigungen in einem Programm nachvollziehen musste. Wollte man nun eine Vielzahl von Gegnern verwalten, hatte man eine Menge Arbeit vor sich. Mit dem recht spartanischen Befehlssatz war es teilweise eine echte Qual, sämtliche Positionen, Lebensenergien und so weiter sinnvoll zu verwalten. Im Grunde mussten jeder Gegner und jedes im Spiel vorkommende Objekt getrennt behandelt werden.

Die Programmiersprache C++ stellt hingegen eine Menge Möglichkeiten zur Verfügung, um Objekte zu gruppieren, ihnen ihre eigenen individuellen Daten und Funktionen zuzuweisen und sie strukturierter zu verwalten. Später wirst Du lernen, was Klassen, Strukturen und Funktionen sind und wie man sinnvoll mit ihnen umgeht.

Ziel der objektorientierten Programmierung ist (nicht nur in meinen Augen) das Zusammenfassen von zusammengehörenden Dingen in immer wiederverwertbare Teile und Abschnitte. Das bedeutet, dass dem Programmierer die Arbeit enorm erleichtert wird, wenn er einmal gewisse grundlegende Eigenschaften festgelegt hat. Er kann beispielsweise festlegen, wie sich ein Raumschiff allgemein verhalten soll, und später davon ein spezielles Raumschiff ableiten, ohne viel im Quelltext zu ändern. In den späteren Kapiteln werde ich noch etwas genauer darauf eingehen, da es ohne pragmatische Beispiele nur schwer möglich ist, das Konzept der objektorientierten Programmierung sinnvoll zu verdeutlichen.

1.2.3 Der ANSI-Standard

Wie Du sicherlich weißt, haben Menschen den Drang, alles zu vereinheitlichen, allen Dingen einen Namen zu geben und alles Mögliche zu standardisieren, was ja auch einen Sinn hat (damit Raumschiffe an eine Station andocken können, müssen sie zum Beispiel eine einheitliche Andockvorrichtung besitzen). Hier bei uns in Deutschland ist die Deutsche Industrie Norm (DIN) das bekannteste Beispiel. In den vereinigten Staaten ist das Äquivalent dazu das „American National Standards Institute“ (ANSI).

Irgendwann kam der Zeitpunkt, an dem es jemand für sinnvoll hielt, auch die Programmiersprache C++ zu vereinheitlichen und gewisse Standards festzulegen. Im ersten Moment mag man sich fragen, wozu so etwas nötig ist. Denkt man etwas darüber nach, kommt man zu dem Schluss, dass es verschiedene Hersteller von Compilern (Microsoft, Borland u. v. m.) gibt, die sich noch dazu je nach Betriebssystem unterscheiden (Windows, Linux). Würde nun jeder Compilerhersteller einige Variationen einbauen, wäre das Chaos im wahrsten Sinne des Wortes schon vorprogrammiert, und das Portieren von Quellcode vom einen System auf das andere wäre mit sehr viel Aufwand verbunden. Unter Portierung versteht man das Umsetzen eines Programms auf ein anderes Betriebssystem. Wenn sich nun sowohl die Compilerhersteller als auch die Programmierer an den ANSI-Standard halten, ist die Portierbarkeit sichergestellt, und es gibt wesentlich weniger Probleme. Wie man sieht, eine sehr nützliche Sache.

Ein weiterer Punkt, der für den ANSI-Standard spricht, ist die sogenannte STL (Standard Template Library). Was genau diese STL ist, lässt sich an dieser Stelle noch nicht so einfach erklären. Allgemein kann man sagen, dass es sich um eine Art Funktionsbibliothek handelt, die einem viele nützliche Dinge bietet, die man nicht mehr selbst neu programmieren muss. Da gerade die STL ein sehr wichtiges Thema ist, wurde ihr in diesem Buch gleich ein ganzes Kapitel (Kapitel 9) gewidmet. Doch was hat diese STL nun mit dem ANSI-Standard zu tun? Nun, alle Funktionen in dieser Bibliothek sind streng nach dem ANSI-Standard programmiert und können somit von jedem Compiler fehlerlos verwendet werden (natürlich nur dann, wenn sich dieser auch an den ANSI-Standard hält).

Wer sich gerne genauer mit den Bestimmungen der ANSI-Norm beschäftigen möchte, sollte sich im Internet etwas umschauen. In diesem Buch werde ich nur so weit auf diese Norm eingehen, dass sichergestellt ist, dass sämtliche Code-Beispiele auf den gängigsten Compilern laufen.

1.2.4 Warum gerade C++?

Möglicherweise stellst Du Dir an dieser Stelle die Frage, warum man eigentlich ausgerechnet C++ für die Spieleprogrammierung verwenden sollte. Schließlich gibt es noch eine ganze Reihe anderer Programmiersprachen wie zum Beispiel Visual Basic, Java oder Delphi. Natürlich ist es möglich, auch mit diesen Sprachen ein funktionierendes Spiel zu programmieren. Die entscheidenden Punkte sind jedoch der Komfort der Sprache sowie die Systemnähe. Damit ist gemeint, wie weit es die Sprache zulässt, die Dinge selbst in die Hand zu nehmen. Nicht alle Programmiersprachen unterstützen sogenannte „Zeiger“, die für effektives Programmieren und schnelle Programme unerlässlich sind. Was es mit diesen Zeigern auf sich hat, werden wir allerdings erst später klären. (Ja, ja, ich weiß, ich verschiebe dauernd Themen nach hinten.) Leider lässt sich das aber nicht immer vermeiden, wie wir später noch sehen werden. An dieser Stelle möchte ich nur erwähnen, dass diese Zeiger sehr wichtig sind und man sie sich eigentlich nicht mehr aus der Spieleprogrammierung wegdenken kann. Man kann mit ihrer Hilfe zum Beispiel selbst Speicher reservieren, was eine ganze Menge Vorteile bringt.

Weiterhin sind viele SDKs (Software Development Kit) für C++ optimiert. Zwar kann man einige dieser SDKs auch mit Visual Basic oder Delphi verwenden, jedoch eben nicht alle. Als

Beispiel könnte man das DirectX-SDK nennen, das sogenannte Bibliotheksdateien und fertigen Quellcode zur Verfügung stellt, um Multimedia-Anwendungen zu programmieren (also die von uns heiß ersehnten eigenen Spiele).

Aus diesen Gründen ist C++ die vorherrschende Sprache bei der Spieleprogrammierung, und sie wird es wohl auch noch eine ganze Weile lang bleiben. Effektivität, eine Menge Freiheiten und die enorme Systemnähe machen C++ zu der Nummer eins der Programmiersprachen, besonders dann, wenn es um das Entwickeln von Spielen geht.

Es gibt eine Menge Leute, die der Meinung sind, dass „Neulinge“ erst einmal eine einfachere Sprache als C++ lernen sollen. Davon halte ich nicht viel, denn C++ ist auch nicht schwerer zu erlernen als andere Programmiersprachen. Wenn man sich mit der Thematik beschäftigt, braucht man nicht erst den Umweg über andere Sprachen zu gehen. Warum sollte man lernen, wie man eine Dampflok fährt, wenn der ICE direkt daneben steht? Wenn man C++ von Anfang an lernt und konsequent am Ball bleibt, trifft man in meinen Augen die richtige Entscheidung.

1.3 Jetzt geht es los … unser erstes Programm

So, nun ist es endlich so weit. Wir haben eine ganze Menge an theoretischem Geschwafel hinter uns gebracht und können uns jetzt endlich unserem ersten Programm widmen. Ich werde im gesamten Buch so verfahren, dass ich zuerst den kompletten Quelltext zeige und danach mit der Erklärung beginne. Alle neuen Befehle und die nötigen Erklärungen dazu erfolgen dann Schritt für Schritt und Zeile für Zeile. Das hat den Sinn, dass man den gesamten Quelltext auf einen Blick hat und nicht die einzelnen Bruchstücke im Kapitel zusammensuchen muss.

Nachdem der Sinn und die Funktionsweise des ersten Programms erklärt wurden, werde ich Dir zeigen, wie man sich einen neuen Arbeitsbereich mit Visual C++ anlegt, das Programm eingibt, kompiliert, linkt und ausführt. Generell ist dieses Buch so geschrieben, dass die darin enthaltenen Quelltexte compilerunabhängig sind, jedoch möchte ich zumindest die wichtigsten Punkte der am meisten verwendeten Compiler abdecken. Solltest Du mit einem anderen als dem hier behandelten Compiler arbeiten, schlage bitte im zugehörigen Handbuch nach oder lies die beigelegten Hilfedateien durch, um zu erfahren, wie Du einen neuen Arbeitsbereich beziehungsweise ein neues Projekt erstellst.

HINWEIS: Um für mehr Übersichtlichkeit zu sorgen, werden die einzelnen Programmzeilen mit Zeilennummern versehen. Diese dürfen nicht mit abgetippt werden, da es sonst zu Fehlern bei der späteren Kompilierung kommt. Natürlich dürfen die Doppelpunkte direkt nach den Zeilennummern auch nicht mit eingegeben werden.

So, nun ist es so weit. Hier kommt der erste Quelltext. Lies hier aber erst weiter, bevor Du Dich dranmachst, das Listing abzutippen. Weiter unten gibt es nämlich noch einige wichtige Erklärungen dazu.

Listing 1.1 Das erste Programm

```
01: // C++ für Spieleprogrammierer
02: // Listing 1.1
03: // Es wird ein Begrüßungstext ausgegeben
04: //
05: #include <iostream>
06:
07: using namespace std;
08:
09: // Hauptprogramm
10: //
11: int main ()
12: {
13:   cout << "Hier kommt die Konkurrenz!\n";
14:   return 0;
15: }
```

Bildschirmausgabe:

```
Hier kommt die Konkurrenz!
```

Tja, das schaut auf den ersten Blick doch ein bisschen verwirrend aus, oder nicht? Im Grunde gibt es hier jedoch nichts, was nicht einem immer wieder gleich ablaufenden Aufbau folgt. Dieses kleine Programm hat eigentlich nur die Aufgabe, einen kurzen Begrüßungstext auf dem Bildschirm auszugeben. Du magst Dich jetzt vielleicht fragen, warum man für eine so kleine Aufgabe denn so viele Zeilen benötigt. Diese Frage ist berechtigt, und es gibt eine recht kurze Antwort darauf: Nur sechs Zeilen dieses Listings sind für den Compiler wirklich wichtig! Die restlichen Zeilen sind sogenannte Kommentare oder Leerzeilen und dienen ausschließlich der Übersichtlichkeit und Lesbarkeit, ohne dabei das eigentliche Programm zu beeinflussen. Was Kommentare sind und wie man sie verwendet, werden wir gleich als Nächstes klären.

1.3.1 Kommentare im Quelltext

Wenn Du Dir den Quelltext genau anschaust, wirst Du feststellen, dass in den Zeilen 1, 2, 3 und 9 Sätze in Klartext stehen. Diese werden beim Kompilieren des Quelltextes einfach ignoriert und haben letztendlich keine Funktion. Der Einzige, den diese Kommentare interessieren, ist derjenige, der mit dem Quelltext arbeitet. In diesem kleinen Beispiel mag es etwas unsinnig erscheinen, alles so genau zu kommentieren. Doch es ist sehr wichtig, sich gleich zu Anfang gewisse Dinge anzugewöhnen. Gerade das vernünftige Kommentieren eines Quelltextes gehört zu den wichtigsten Dingen beim Programmieren überhaupt. Dabei wird man zu Beginn recht häufig denken, dass man dieses oder jenes ja nicht zu kommentieren braucht, da ja alles so wunderbar selbsterklärend ist. Allerdings wird man recht schnell die Erfahrung machen, dass ein Quelltext, den man seit einigen Wochen nicht mehr

angerührt hat, plötzlich die Eigenart aufweist, unverständlich zu erscheinen. Dann ist das Malheur passiert, und man muss sich wieder in Erinnerung rufen, warum man damals etwas so gemacht hat und was man hier und da eigentlich bewirken wollte. Durch sauberes Kommentieren erspart man sich letzten Endes eine Menge Arbeit.

Damit der Compiler „weiß", dass es sich um einen solchen Kommentar handelt, muss man vor den Klartext zwei Slashes stellen (`//`). Alles, was in dieser Zeile nach den Slashes steht, wird beim Kompilieren des Programms einfach ignoriert. Nun kann es ja auch vorkommen, dass ein Kommentar so ausführlich ist, dass er über mehrere Zeilen geht. Es wäre ja jetzt etwas mühsam, jede Kommentarzeile mit zwei Slashes zu beginnen. Aus diesem Grund gibt es noch eine zweite Möglichkeit, Kommentare zu erstellen. Man stellt einfach zu Beginn eines Kommentarblockes die Zeichenfolge `/*` voran, schreibt seinen Text und beendet den Block mit der Zeichenfolge `*/`. Und so schaut das Ganze dann aus:

```
01: // Dies wäre eine sehr
02: // umständliche Art, lange
03: // Kommentare zu schreiben,
04: // oder etwa nicht?
01: /*
02: So ist das Ganze
03: doch schon um einiges
04: einfacher, oder?
05: */
```

Diese Art der Kommentierung kann noch auf eine andere Weise nützlich sein. Stell Dir vor, Du hast ein Spiel programmiert (ziemlich coole Vorstellung, oder?). Nun hast Du einen Programmteil, den Du aus irgendeinem Grund von der Kompilierung ausschließen möchtest. Du brauchst aber die betreffenden Programmzeilen nicht zu löschen, sondern Du kannst sie einfach mit der vorangehend gezeigten Methode „auskommentieren".

HINWEIS: Verwende Kommentare großzügig. Sie zu schreiben ist nicht halb so viel Arbeit, wie einen unkommentierten Quelltext mühsam zu entziffern. Versuche dabei jedoch, Dich auf sinnvolle Kommentare zu beschränken und diese aussagekräftig zu halten.

1.3.2 Die #include-Anweisung

Nachdem die Kommentarzeilen nun abgehakt sind, können wir uns den Programmzeilen widmen, die für die eigentliche Funktion des Programms wichtig sind. In der Zeile 5 gibt es auch schon gleich eine kleine Besonderheit, nämlich das Doppelkreuz (#). Dieses Zeichen ist für den Compiler von besonderer Bedeutung. Jeder Befehl, der mit einem solchen Doppelkreuz beginnt, ist ein sogenannter Präprozessor-Befehl, auch Präprozessor-Direktive genannt. Wenn Dein Quelltext kompiliert wird, dann passiert das nicht in einem Rutsch, sondern in mehreren Schritten. Der erste Schritt ist dabei die Behandlung aller Präprozessor-Befehle. Davon gibt es eine ganze Reihe, die wir im Verlauf des Buches noch kennenlernen werden. Diese Befehle haben im Grunde nichts mit dem fertig kompilierten Programm

zu tun. Die `#include`-Anweisung in der Zeilen 5 dient nun dazu, bereits vorhandene Quelltextdateien zu Deinem Quelltext hinzuzufügen (einzubinden). Die Datei „iostream“ gehört dabei zum Lieferumfang Deines Compilers und enthält alles, was nötig ist, um einfache Textausgaben zu realisieren. Was genau diese Datei beinhaltet, lässt sich aus dem Namen herauslesen. Das „`io`“ steht für „*input/output*“, und „`stream`“ bedeutet übersetzt etwa so viel wie „Strom“, wobei damit der Datenstrom gemeint ist. Dir stehen mit dem Einbinden dieser Datei somit viele Möglichkeiten zur Verfügung, um Datenströme zu verwalten. Das beinhaltet sowohl die Ausgabe von Text auf dem Bildschirm als auch die Eingabe von Text über die Tastatur. Würde man diese Datei nicht einbinden, wüsste der Compiler mit dem Befehl in Zeile 12 nichts anzufangen.

Es gibt eine ganze Reihe dieser Dateien, zum Beispiel für verschiedene Mathematikfunktionen oder auch zum Zeichnen von Grafiken. Im Verlauf des Buches wirst Du auch lernen, wie Du Dir solche Dateien selbst erstellen und verwenden kannst.

Nun noch ein Wort zu den spitzen Klammern (`< >`), in die der Dateiname eingeschlossen ist. Gibt man den Dateinamen in Anführungszeichen ein, dann sucht der Compiler die Datei im aktuellen Arbeitsverzeichnis. Bei den spitzen Klammern hingegen sucht er in einem speziellen Verzeichnis, das in der Regel im Installationsordner des Compilers zu finden ist. Dadurch erspart man es sich, die gewünschte Datei extra ins angelegte Arbeitsverzeichnis kopieren zu müssen.

HINWEIS: Alle Dateien, die mit `#include` eingebunden werden und nicht von uns selbst stammen oder separat installiert wurden, müssen in spitzen Klammern (`<>`) stehen.

Jetzt noch ein paar Worte zu Zeile 7. In C++ gibt es die Möglichkeit, sogenannte Namensbereiche festzulegen. Dies wird zum Beispiel dann gemacht, wenn man mit mehreren Quellcode-Dateien arbeitet. Da man Funktionen und Variablen Namen geben kann, kann es dabei recht leicht zu Konflikten kommen, wenn zwei verschiedene Dinge aus verschiedenen Dateien den gleichen Namen haben. Gerade dann, wenn mehrere Leute an einem Projekt arbeiten, kann dies zum Problem werden. Aus diesem Grund kann man bestimmte Teile eines Quelltextes in einen Namensbereich gruppieren. Möchte man nun Funktionen oder Variablen aus diesem Namensbereich verwenden, so muss man dies dem Compiler mitteilen. Und genau das tun wir hier in Zeile 7. Wir sagen dem Compiler, dass wir den Namensbereich **std** (was für „Standard“ steht) verwenden möchten. Die in den eingebundenen Dateien enthaltenen Funktionen und Variablen befinden sich allesamt in diesem Namensbereich.

Es gibt sicherlich noch eine ganze Menge zu den Namensbereichen zu sagen, jedoch müsste man dazu einfach zu weit ausholen. Deshalb belassen wir es hier einfach mal bei der Tatsache, dass diese Zeile benötigt wird.