C Programmieren für Einsteiger

Markus Neumann

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

www.bmu-verlag.de
support@bmu-verlag.de

Lektorat: Matthias Kaiser
Einbandgestaltung: Pro ebookcovers Angie
Druck und Bindung: Wydawnictwo Poligraf sp. zo.o. (Polen)

Taschenbuch-ISBN: 978-3-96645-060-7
Hardcover-ISBN: 978-3-96645-061-4
E-Book-ISBN: 978-3-96645-059-1

C Programmieren für Einsteiger

Inhaltsverzeichnis

Alle Programmcodes aus diesem Buch sind als PDF zum Download verfügbar. Dadurch müssen Sie sie nicht abtippen:
https://bmu-verlag.de/c

Außerdem erhalten Sie die eBook Ausgabe zum Buch im PDF Format kostenlos auf unserer Website:

https://bmu-verlag.de/c
Downloadcode: siehe Kapitel 19

Kapitel 1
Einleitung

Programmieren zu lernen, stellt sicherlich eine anspruchsvolle Aufgabe dar. Allerdings ist diese auch sehr lohnenswert. Gute Programmierkenntnisse sind in immer mehr Berufen von Bedeutung. Wer sich in dieses Thema einarbeitet, verbessert daher seine Chancen auf eine Anstellung oder auf eine Beförderung deutlich. Außerdem stellt das einen guten Einstieg dar, um eine Berufsausbildung oder ein Studium im Bereich der *Informatik* aufzunehmen. Schließlich stellt das Erstellen eines Programms auch eine interessante intellektuelle Herausforderung dar. Man kann diese Aufgabe wie ein Rätsel auffassen, das es mit logischen Schlussfolgerungen zu lösen gilt.

Wer mit dem Programmieren beginnen will, muss sich dabei zunächst für eine *Programmiersprache* entscheiden. Hierfür gibt es unzählige Möglichkeiten, die sich für ganz unterschiedliche Anwendungen eignen. Zwar ist es später jederzeit möglich, eine weitere Sprache zu erlernen, doch gibt die zu Beginn getroffene Entscheidung häufig bereits die Marschrichtung vor. Dieses Buch befasst sich mit der Programmiersprache C. Daher stellt das erste Kapitel die wesentlichen Eigenschaften dieser Programmiersprache sowie deren Entstehungsgeschichte vor.

Für dieses Buch sind keine Vorkenntnisse erforderlich. Es richtet sich zum einen an Leser, die noch gar keine Erfahrungen mit dem Programmieren gemacht haben. Zum anderen eignet es sich jedoch auch für Personen, die bereits eine andere Programmiersprache kennengelernt haben und nun auf C umsteigen möchten. In diesem Fall wird das eine oder andere Detail, das hier erklärt wird, eventuell bereits bekannt sein. Doch weist die Programmiersprache C zahlreiche Besonderheiten auf, sodass es dennoch sinnvoll ist, sich nochmals genau damit auseinanderzusetzen.

1.1 C: eine Programmiersprache mit langer Geschichte

Die erste Version der Programmiersprache C erschien bereits 1972. Insbesondere in der Informatik stellt das einen sehr langen Zeitraum dar. Doch auch der eine oder andere Leser wird damals noch nicht einmal auf der Welt gewesen sein. Auch die Verwendung von Computern war zu dieser Zeit noch

eine ganz andere. Die Rechengeräte, die damals zum Einsatz kamen, waren zwar riesig groß, doch wiesen sie im Vergleich zu einem heutigen PC eine verschwindend geringe Leistung auf. In Privathaushalten war damals in der Regel noch überhaupt kein Computer zu finden. Deren Verwendung war aufgrund der enormen Kosten im Wesentlichen großen Unternehmen und Forschungseinrichtungen vorbehalten.

Auch das heute übliche Betriebssystem Windows war damals noch in weiter Ferne. Drei Jahre zuvor wurde gerade das *Betriebssystem* Unix vorgestellt, das daraufhin die Computer-Branche für einige Zeit beherrschte. Die Entstehung von C ist eng mit Unix verbunden. Die erste Version des Betriebssystems schrieben die bekannten Informatiker Ken Thompson und Dennis Ritchie 1969 in *Assemblersprache*. Dabei handelt es sich um eine Sprache, die genau an der Prozessorarchitektur ausgerichtet ist. Das bedeutet, dass sie genau die Befehle verwendet, die anschließend dem Prozessor zugeführt werden. Die Assemblersprache wird als Programmiersprache der zweiten Generation bezeichnet – nachdem die Sprachen der ersten Generation noch mit konkreten Zahlencodes arbeiteten.

Die Arbeit mit der Assemblersprache ist ausgesprochen aufwendig und kompliziert. Daher waren die Entwickler von Unix darum bemüht, die nächste Version in einer höheren Programmiersprache zu verfassen. Allerdings stand damals keine geeignete höhere Sprache zur Verfügung, die es erlaubte, ein komplettes Betriebssystem zu erstellen. Daher beschloss einer der beiden Entwickler des Betriebssystems Unix – Dennis Ritchie – hierfür eine eigene Sprache zu entwickeln. Dieser gab er den Namen C. Diese Bezeichnung macht deutlich, dass es sich hierbei um eine Weiterentwicklung der Programmiersprache B handelt, die die beiden genannten Programmierer bereits zuvor entwickelt hatten.

In den folgenden Jahren gewann C immer mehr an Beliebtheit. Bis in die 90er Jahre handelte es sich dabei um eine der am häufigsten verwendeten Programmiersprachen. Sie diente nicht nur der Erstellung der neuen Version des Unix-Betriebssystems, sondern auch vieler daraus abgeleiteter Systeme. Ein bekanntes Beispiel hierfür ist das Betriebssystem Linux, das noch heute bei vielen PCs Verwendung findet. Auch das Apple-Betriebssystem iOS ist in C geschrieben – genauso wie einige Teile von Windows. Hinzu kommen die Kerne der beliebten Smartphone-Betriebssysteme Android und iOS. Das zeigt nicht nur, welchen großen Einfluss diese Programmiersprache hat. Darüber hinaus wird daran deutlich, dass sie auch heute noch für sehr moderne Anwendungen zum Einsatz kommt. Obwohl mittlerweile höhere Programmiersprachen wie Java, C++ oder Python stark an Bedeutung

gewonnen haben, ist es daher nach wie vor sehr lohnenswert, sich mit C zu befassen.

1.2 Die Anwendungsmöglichkeiten von C

Bei der Entscheidung, welche Programmiersprache man erlernen will, spielen die entsprechenden Anwendungsmöglichkeiten eine wichtige Rolle. Daher ist es sinnvoll, sich mit den typischen Einsatzbereichen von C zu befassen.

Im vorherigen Abschnitt wurde bereits erwähnt, dass C entwickelt wurde, um das Betriebssystem Unix zu programmieren. Das zeigt bereits eine der zentralen Anwendungsmöglichkeiten: die Entwicklung von Betriebssystemen. Fast alle heute üblichen Betriebssysteme sind zumindest zum Teil in C verfasst.

Daraus lässt sich eine der wesentlichen Eigenschaften von C ableiten. Das Betriebssystem stellt die Schnittstelle zwischen dem Anwender und der Hardware dar. Das hat zur Folge, dass es für die Steuerung der einzelnen Komponenten zuständig ist. Hierfür eignet sich C ausgezeichnet. Im Vergleich zu vielen anderen Programmiersprachen ist hierbei das Abstraktionsniveau vergleichsweise gering. Das bedeutet, dass man mit C die Abläufe der Hardware sehr präzise steuern kann. Das ist nicht nur für das Erstellen eines Betriebssystems notwendig. Auch wenn man einen Treiber für ein Peripheriegerät erstellen will, bietet es sich daher an, C dafür zu verwenden.

Diese Eigenschaft führt dazu, dass C sich auch hervorragend für die Steuerung von Automatisierungsprozessen und für Anwendungen im Bereich Internet of Things eignet. Obwohl es sich hierbei um eine recht alte Programmiersprache handelt, kommt sie daher auch für viele moderne Anwendungen zum Einsatz.

Das geringe Abstraktionsniveau hat zur Folge, dass man bei Programmen, die in C geschrieben sind, alle Regeln sehr genau beachten muss. Das mag insbesondere für Anfänger lästig erscheinen. Allerdings lernt man auf diese Weise sehr viel über die Funktionsweise eines Computers. Dafür ist zwar etwas mehr Disziplin erforderlich als bei anderen Programmiersprachen. Doch erwirbt man auf diese Weise ausgezeichnete Grundlagenkenntnisse, die auch für viele weitere Anwendungen sehr hilfreich sind.

Nicht besonders gut geeignet ist C hingegen, wenn man Programme mit grafischen Benutzeroberflächen gestalten will. Typische Windows-Programme, die mit Fenstern arbeiten, lassen sich hiermit zwar ebenfalls erstellen.

Allerdings ist diese Aufgabe deutlich komplizierter als bei vielen anderen Programmiersprachen. Wenn man sich hierauf spezialisieren will, stellt C daher sicherlich nicht die richtige Wahl dar.

1.3 C – Grundlage für viele weitere Programmiersprachen

C ist nicht nur eine der ältesten Programmiersprachen, die noch heute eine breite Verwendung haben. Darüber hinaus hatte diese Sprache einen enormen Einfluss auf viele verschiedene Entwicklungen im Bereich der Informatik – insbesondere auf die Gestaltung neuer Programmiersprachen. Das bedeutet, dass es hierbei zahlreiche Ähnlichkeiten gibt. Diese beziehen sich zum einen auf die die *Syntax* – also auf den strukturellen Aufbau und die Regeln zur Gestaltung der Programme. Darüber hinaus sind die Befehle dabei häufig recht ähnlich.

Bei Programmiersprachen wie C++, C#, C-- oder Objective-C wird die Beziehung zu C bereits durch die Namensgebung deutlich. Auch der Name der Programmiersprache D zeigt, dass es sich hierbei um eine Ableitung beziehungsweise um eine Weiterentwicklung von C handelt. Daher übernimmt D ebenfalls zahlreiche Elemente von C. Doch auch die Programmiersprachen Java, JavaScript, Perl, Python, PHP und einige weitere weisen Verbindungen zu C auf – wobei diese bei manchen der genannten Beispiele nur gering ausgeprägt sind. Dennoch bleibt festzuhalten, dass es kaum eine moderne Programmiersprache gibt, die nicht in gewisser Weise auf C aufbaut. Das zeigt, wie groß der Einfluss dieser Sprache ist.

Auch das ist ein Grund, der dafür spricht, C zu lernen – insbesondere wenn es sich hierbei um die erste Programmiersprache handelt, die man sich aneignet. Auf diese Weise lernt man die Grundlagen vieler weiterer Sprachen kennen. Wenn man später eine Anwendung programmieren will, für die die Verwendung einer anderen Programmiersprache sinnvoll ist, fällt der Umstieg in der Regel leicht, da die Grundlagen bereits bekannt sind. Daher stellen gute Kenntnisse in C eine hervorragende Basis für vielfältige Aufgaben im Bereich der Informatik dar.

Kapitel 2

Vorbereitungsmaßnahmen für das Programmieren in C

Wenn man mit dem Programmieren beginnen will, ist es zunächst notwendig, den Computer auf diese Aufgabe vorzubereiten. Zum einen benötigen wir ein passendes Programm, um den Programmcode zu verfassen. Zum anderen ist eine Software notwendig, die diesen in ein ausführbares Programm umwandelt. Daher müssen wir zunächst die entsprechende Software auf unserem Rechner installieren.

2.1 Texteditor und Compiler: die Grundlagen für die Programmierung in C

Wenn wir ein Programm schreiben, verwenden wir hierfür Befehle, Zahlen, Variablen und ähnliche Elemente. Diese bestehen aus Buchstaben, Ziffern und einigen weiteren Zeichen – also aus Text. Die meisten Anwender nutzen ein Textverarbeitungsprogramm wie Word, um Texte auf dem Computer zu erstellen. Wenn wir jedoch ein Computerprogramm schreiben, stellt dies nicht die richtige Wahl dar. Der Grund liegt darin, dass derartige Programme neben dem eigentlichen Text noch zahlreiche weitere Informationen abspeichern – beispielsweise die Schriftart, die Schriftgröße, die Farbe und den Hintergrund. Diese zusätzlichen Informationen sind bei einem Computerprogramm nicht nur überflüssig, sie stören hierbei sogar. Das führt dazu, dass wir ein Programm, das wir mit einem Textverarbeitungsprogramm geschrieben haben, nicht ausführen können.

Aus diesem Grund benötigen wir eine Software für die Programmerstellung, die ausschließlich die Schriftzeichen abspeichert, die wir eingeben. In diesem Fall spricht man davon, dass die Datei in reiner Textform gespeichert wird. Hierfür kommt ein sogenannter *Texteditor* zum Einsatz.

Fast jedes Betriebssystem verfügt bereits über einen standardmäßig installierten Texteditor. Unter Windows ist dies beispielsweise der Microsoft Editor – besser bekannt unter der englischsprachigen Bezeichnung Notepad. Dieser ist zum Programmieren geeignet. Allerdings ist sein Funktionsumfang nur minimal. Daher ist es sinnvoll, einen etwas höherwertigen Texteditor zu verwenden. Dieser sorgt beispielsweise für eine automatische

Syntaxhervorhebung. Das bedeutet, dass unterschiedliche Schlüsselbegriffe des Programms in verschiedenen Farben markiert werden. Darüber hinaus rückt er zusammengehörige Blöcke automatisch ein und sorgt auf diese Weise für übersichtlichere Strukturen, so wie dies in Abbildung 2.1 zu sehen ist.

Abb. 2.1 Ein Computerprogramm in einem Texteditor (hier am Beispiel Geany gezeigt)

Nachdem wir das Programm in einem Texteditor erstellt haben, ist es jedoch noch nicht unmittelbar möglich, es auszuführen. Dazu ist eine Art Übersetzungsprozess notwendig. Die Schriftzeichen, die wir hier verwendet haben, kann der Prozessor nicht interpretieren. Hierfür ist es notwendig, ihm einzelne Kommandos zuzuführen, die wiederum aus binären Informationen – also aus einzelnen Bits – bestehen. Damit wir das Programm ausführen können, müssen wir es daher in eine Abfolge entsprechender Kommandos umwandeln.

Um diese Aufgabe zu erledigen, gibt es zwei unterschiedliche Möglichkeiten. Es gibt sogenannte interpretierte Programmiersprachen. Hierbei kommt ein *Interpreter* zum Einsatz, der den Programmcode bei jeder Ausführung aufs Neue einliest und ihn in die entsprechenden Befehle umwandelt. Diese führt er außerdem direkt dem Prozessor zu, sodass es zur Ausführung kommt. Diese Alternative bietet insbesondere den Vorteil, dass die entsprechenden Programme plattformunabhängig sind und auf jedem Betriebssystem ausgeführt werden können, auf dem ein entsprechender Interpreter installiert ist.

Um ein C-Programm auszuführen, kommt jedoch eine andere Alternative zum Einsatz. Hierfür verwenden wir einen *Compiler*. Dieser übernimmt den Übersetzungsprozess ebenfalls. Die Befehle führt er jedoch nicht direkt dem Prozessor zu, sondern er speichert sie ab. Auf diese Weise entsteht ein ausführbares Programm – unter Windows durch die Programmendung .exe ge-

kennzeichnet. Hierbei muss der Übersetzungsprozess nur ein einziges Mal durchgeführt werden. Danach kann man das Programm beliebig oft ausführen. Das sorgt für eine deutliche Steigerung der Performance.

Wenn wir ein C-Programm ausführen wollen, benötigen wir daher einen Compiler. Diesen müssen wir daher ebenfalls auf unserem Rechner installieren. Für alle gängigen Betriebssysteme sind kostenfreie C-Compiler verfügbar.

2.2 Die IDE als praktische Alternative

Im vorherigen Abschnitt wurde gesagt, dass wir einen Texteditor und einen Compiler benötigen, um ein C-Programm zu erstellen und auszuführen. Nun wäre es einfach möglich, hierfür jeweils eine passende Software zu installieren. Für beide benötigten Programme sind kostenfreie Angebote im Internet erhältlich. In diesem Fall müssten wir das Programm dann im Texteditor schreiben. Wenn es fertiggestellt ist, rufen wir einen Kommandozeileninterpreter auf. Dieser ist auf allen Betriebssystemen bereits verfügbar. Dort geben wir dann zunächst den Befehl zum Kompilieren des Programms ein. Danach können wir es aufrufen, um es auszuführen.

Diese Vorgehensweise ist jedoch recht kompliziert. Deshalb haben wir im vorherigen Abschnitt auch keine Installationsanleitung für diese Programme gegeben. Die Erklärungen dienten lediglich dazu, den grundsätzlichen Ablauf der Programmerstellung zu verdeutlichen. Mittlerweile gibt es aber nur noch wenige Programmierer, die auf diese Weise arbeiten. Heutzutage kommt hierfür fast immer eine *integrierte Entwicklungsumgebung* (IDE – Integrated Development Environment) zum Einsatz. Deren wesentlicher Vorteil besteht darin, dass sie die Funktion des Texteditors und des Compilers in einem Programm vereint. Das bedeutet, dass wir das Programm in der gleichen Umgebung erstellen, kompilieren und ausführen. Das vereinfacht diesen Prozess und steigert die Effizienz bei der Entwicklung neuer Programme dadurch erheblich. Deshalb wollen wir in diesem Lehrbuch von Beginn an mit einer IDE arbeiten.

Anmerkung: Genaugenommen bieten viele integrierten Entwicklungsumgebungen keinen eigenen Compiler an. Sie verwenden stattdessen eine Verknüpfung zu einer separaten Software, die unsere Programme kompiliert. Für den Anwender ist dieser Unterschied jedoch kaum bemerkbar.

Eine IDE bietet noch zahlreiche weitere Funktionen, die bei der Programmerstellung sehr hilfreich sein können. Beispielsweise ist häufig eine automati-

sche Überprüfung auf Syntaxfehler integriert, sodass diese besonders leicht zu beheben sind. Darüber hinaus ist meistens eine Debugging-Funktion zur Beseitigung logischer Fehler vorhanden. Auch eine Autovervollständigung für die am häufigsten verwendeten Befehle ist üblich. Hierbei handelt es sich jedoch nur um einige wenige Beispiele für die vielen praktischen Funktionen, mit denen eine IDE die Programmerstellung erleichtert.

2.3 Code::Blocks für die Programmierung in C installieren

Nun ist es an der Zeit, eine IDE auf dem Computer zu installieren. Zu diesem Zweck ist es jedoch zunächst notwendig, sich für eine passende Software zu entscheiden. Hierfür gibt es vielfältige Angebote. Viele davon sind gratis verfügbar. Doch gibt es auch zahlreiche kostenpflichtige integrierte Entwicklungsumgebungen. Für dieses Buch wollen wir eine kostenlose Software verwenden. Diese reicht für unsere Zwecke vollkommen aus.

Bei der Auswahl einer IDE ist es außerdem wichtig, darauf zu achten, dass sich diese für die entsprechende Programmiersprache eignet. Nur so ist es möglich, von den Vorteilen dieser Software in vollem Umfang zu profitieren. Für alle gängigen Programmiersprachen gibt es geeignete Angebote.

Wenn man nach einer kostenlosen IDE für die Programmiersprache C sucht, stößt man in erster Linie auf zwei Möglichkeiten: auf Visual Studio und auf Code::Blocks. Die erste dieser beiden Alternativen stammt vom Software-Hersteller Microsoft. Hierbei gibt es eine kostenpflichtige Ausführung für professionelle Programmierer. Darüber hinaus ist jedoch auch eine kostenfreie Version mit etwas reduziertem Funktionsumfang verfügbar. Diese würde für unsere Zwecke jedoch vollkommen ausreichen.

Für dieses Lehrbuch verwenden wir jedoch die IDE Code::Blocks. Diese ist ebenfalls hervorragend für die Programmiersprache C geeignet und bietet hierfür viele nützliche Funktionen. Hierbei handelt es sich um ein Open-Source-Projekt. Das bedeutet, dass nicht nur die Nutzung kostenfrei möglich ist. Darüber hinaus kann man auch den Code einsehen und bei Bedarf anpassen. Der entscheidende Vorteil dieser IDE besteht jedoch darin, dass sie für alle gängigen PC-Betriebssysteme verfügbar ist: für Windows. Linux und für MacOS. Daher kann sie jeder Leser installieren – unabhängig davon, welche dieser Alternativen er verwendet. Um Code::Blocks zu installieren, rufen wir zunächst folgenden Link auf:

http://bmu-verlag.de/c1

Hier klicken wir nun auf den Link mit der Bezeichnung „Download binary release". Dieser führt uns zur aktuellen Version der IDE. Alternativ dazu könnten wir auch den Quellcode der Software herunterladen. Das würde es jedoch notwendig machen, das Programm aus dem Quellcode selbst zu erstellen. Das macht nicht nur etwas umfangreiche Kenntnisse erforderlich. Darüber hinaus würde dadurch der Aufwand erheblich ansteigen.

Wenn wir den entsprechenden Link anklicken, gelangen wir zu einer Seite, die uns zahlreiche Download-Optionen für verschiedene Betriebssysteme anbietet. Leser, die Windows nutzen, sollten dabei die Version codeblocks-17.12mingw-setup.exe auswählen – wobei sich die Versionsnummer dabei selbstverständlich im Laufe der Zeit ändert. In den vorherigen Abschnitten haben wir gelernt, dass wir für die Ausführung eines C-Programms einen Compiler benötigen, dass die IDE diese Funktion jedoch anbietet. Code::Blocks verwendet jedoch keinen eigenen Compiler. Diese Software sorgt lediglich für einen automatischen Zugriff auf dessen Funktionen. Es ist allerdings notwendig, den Compiler ebenfalls zu installieren. Für Windows-Nutzer bietet sich der Compiler MinGW an. Wenn wir die oben genannte Auswahl treffen, laden wir nicht nur die IDE herunter, sondern auch gleich den hierfür notwendigen Compiler. Der Installations-Assistent installiert daraufhin beide Programme. Dabei müssen wir lediglich bei der Auswahl der Komponenten darauf achten, dass MinGW ausgewählt ist – so wie dies in Abbildung 2.2 zu sehen ist.

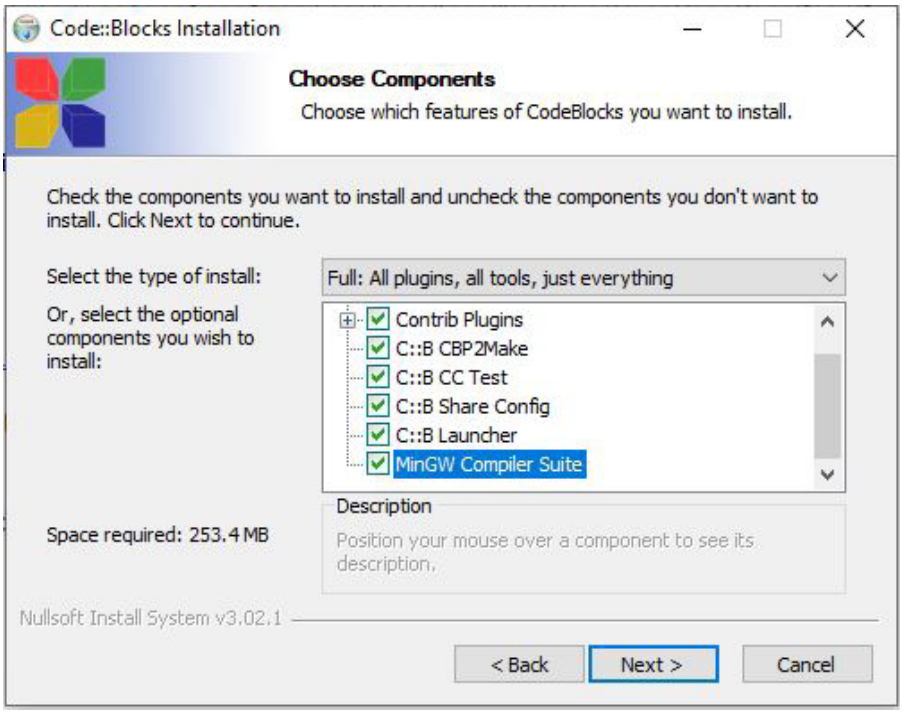

Abb. 2.2 Den Compiler zusammen mit der IDE installieren

Wenn wir Code::Blocks unter Linux oder MacOS installieren, müssen wir den Compiler separat hinzufügen. Beide Systeme bieten hierfür einfache Möglichkeiten an. Für MacOS steht ein Compiler auf der Apple's Developer Page zum Download bereit. Unter Linux muss man hierfür lediglich folgende Programmzeile über das Terminal eingeben: `$ sudo apt install gcc`.

Kapitel 3

Das erste eigene Programm in C erstellen

Nachdem wir die IDE mit dem zugehörigen Compiler installiert haben, kann es losgehen: Jetzt erstellen wir unser erstes eigenes Programm in C. Dabei handelt es sich selbstverständlich um ein sehr einfaches Beispiel. Dieses soll lediglich einen kleinen Text ausgeben. Das ist die einfachste Funktion, die ein Computer-Programm ausführen kann. Daher eignet sich diese hervorragend für den Einstieg. Dabei lernen wir jedoch, wie ein Programm in C grundsätzlich aufgebaut ist und wie wir es kompilieren und ausführen können.

3.1 Der Programmcode für das erste C-Programm

Um unser erstes Programm in C zu schreiben, öffnen wir die IDE Code::Blocks. Dabei erscheint zunächst der Startbildschirm dieser Software, der in Abbildung 3.1 zu sehen ist.

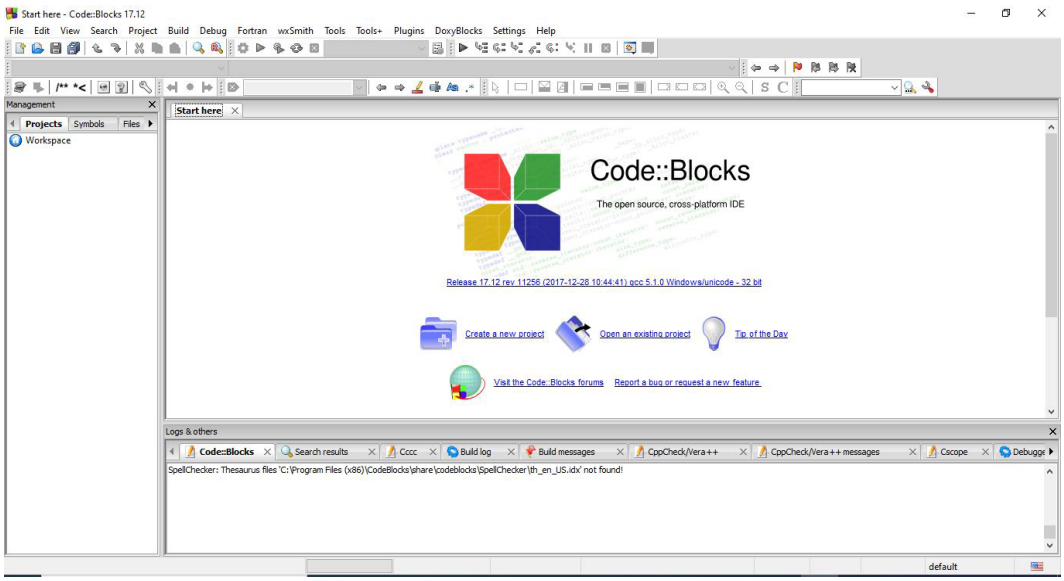

Abb. 3.1 Der Startbildschirm der IDE Code::Blocks

Um mit unserem Programm zu beginnen, erstellen wir zunächst eine neue Datei. Hierzu klicken wir in der Menüleiste auf „File" und anschließend auf „New" und auf „File". Abbildung 3.2 stellt diesen Vorgang dar.

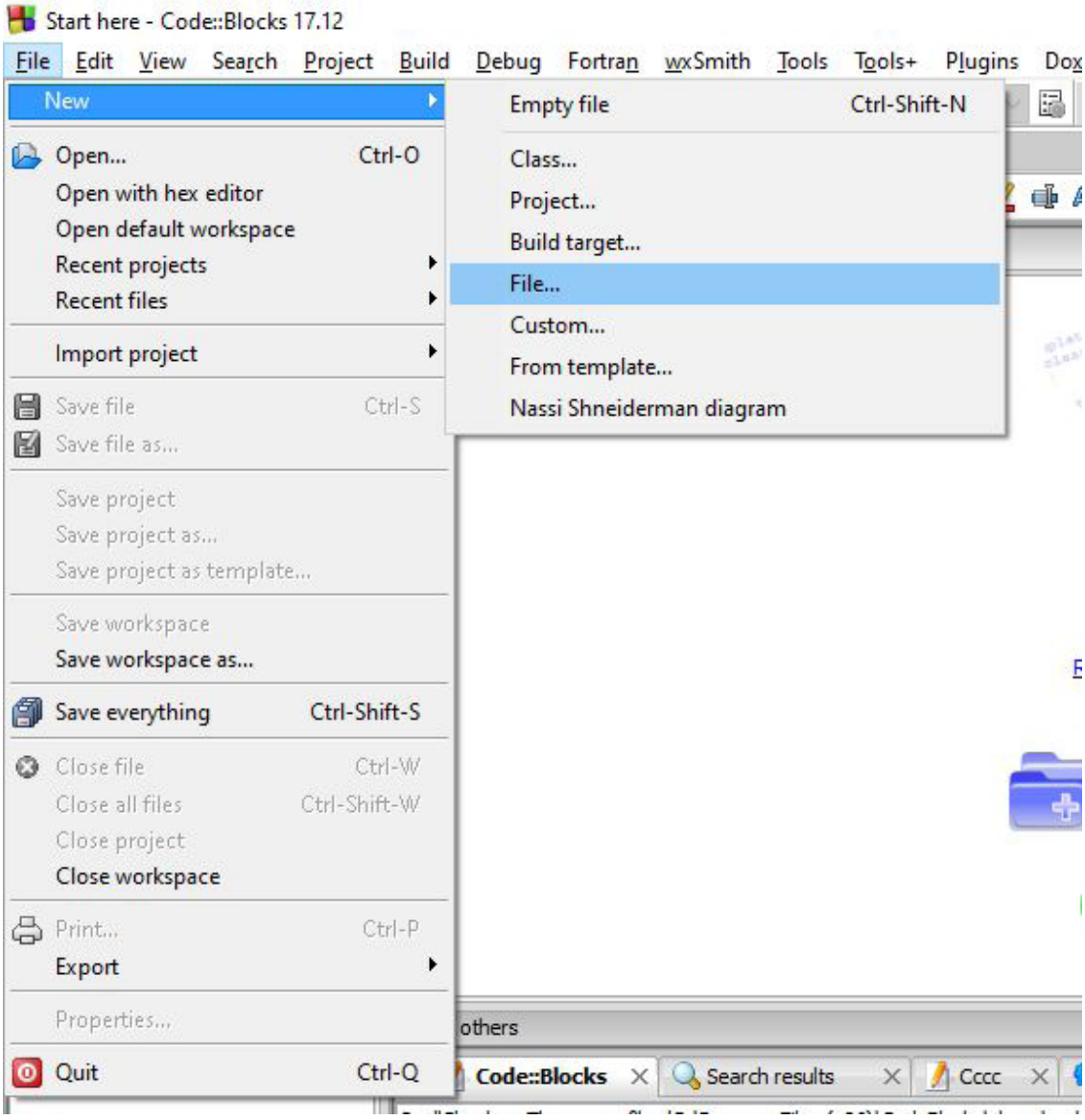

Abb. 3.2 Eine neue Datei erstellen

Daraufhin öffnet sich ein neues Fenster. Dieses ist in Abbildung 3.3 zu sehen. Hier wählen wir nun die Option „C/C++source" aus.

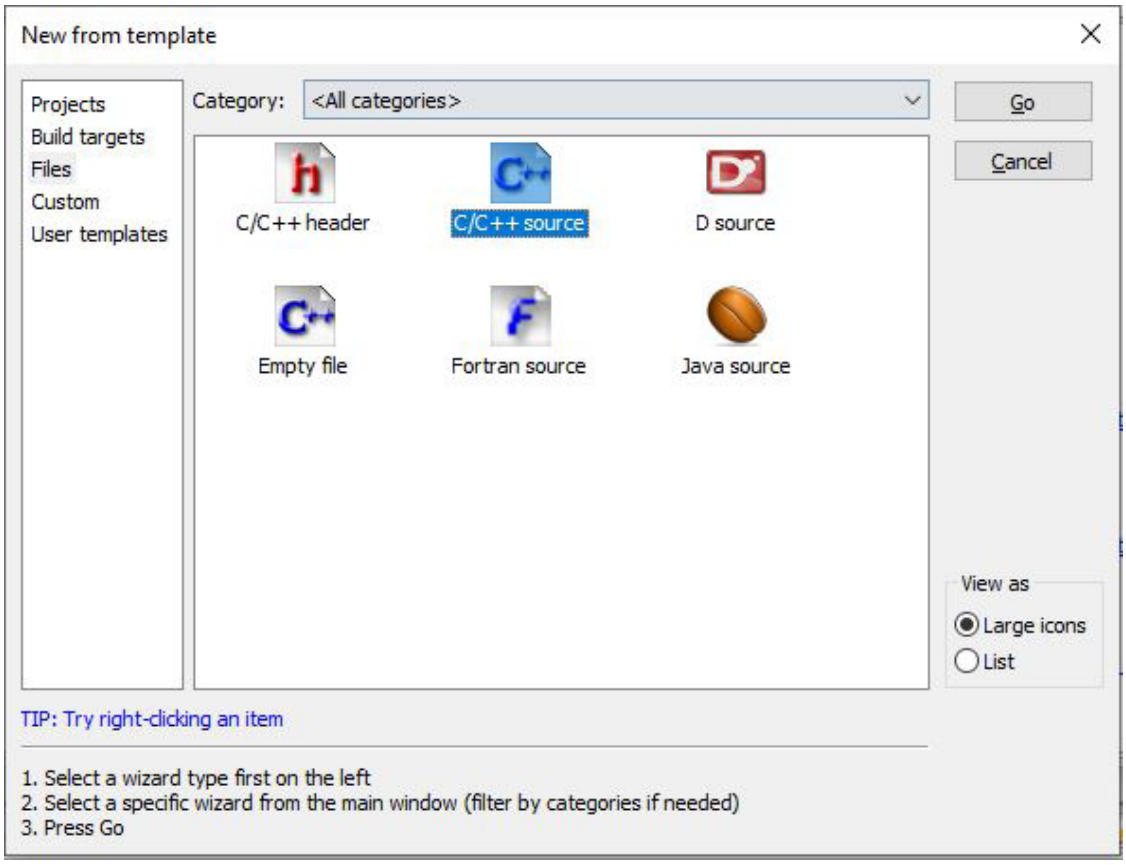

Abb. 3.3 Eine Datei für C-Quellcode erzeugen

Wenn wir diese Aufgabe zum ersten Mal durchführen, erscheint ein weiteres Fenster mit dem sogenannten „C/C++ source file wizard". Dieser gibt uns eine kurze Anleitung, wie wir die entsprechende Datei erstellen können. Wenn wir nicht wünschen, dass dieses Start-Fenster bei jeder neu erstellten Datei erneut erscheint, ist es sinnvoll, die Checkbox mit der Beschriftung „Skip this page next time" anzuklicken, so wie dies in Abbildung 3.4 zu sehen ist.

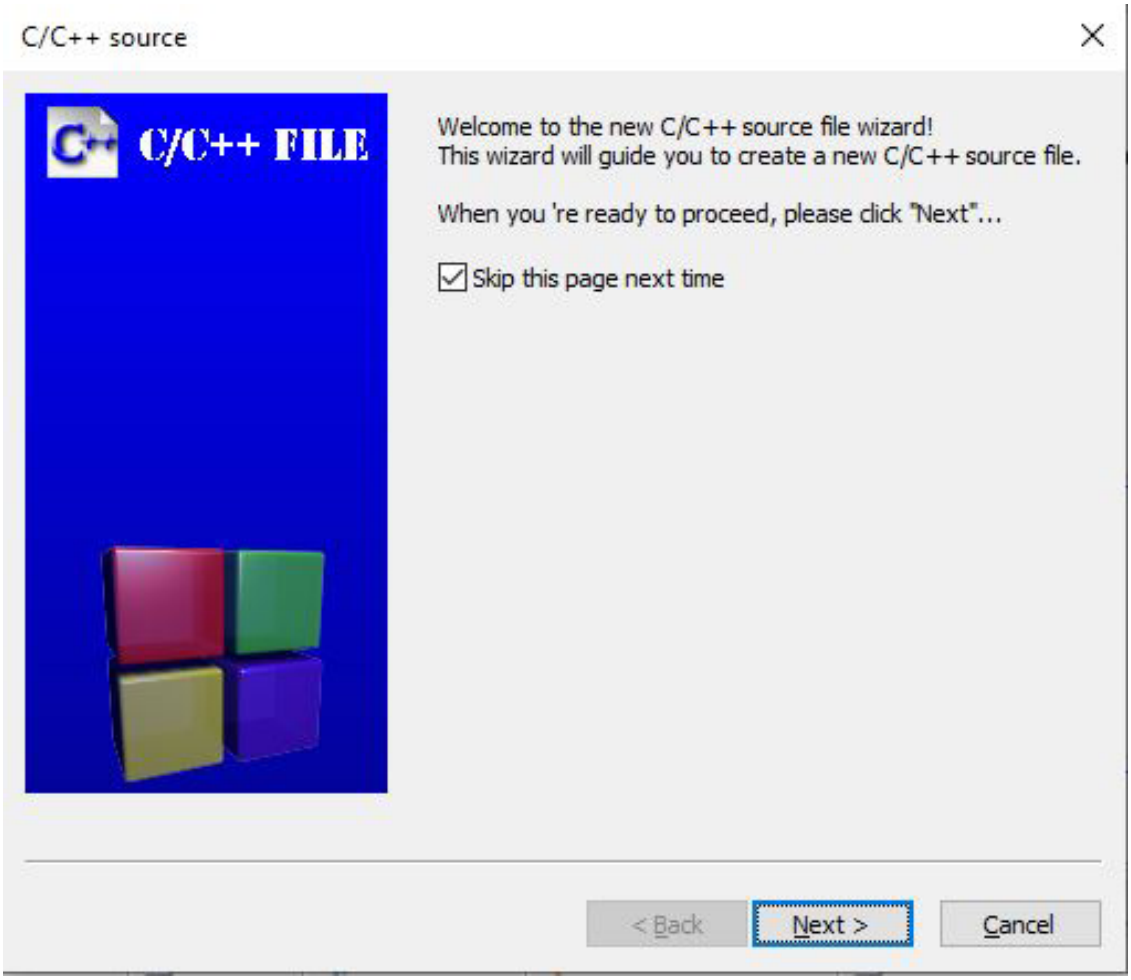

Abb. 3.4 Die Anleitung zur Erstellung einer neuen C-Datei

Wenn wir dieses Fenster mit „Next" bestätigen, können wir auswählen, ob wir eine C- oder eine C++-Datei erstellen möchten. Für unser C-Programm müssen wir selbstverständlich die C-Datei auswählen. Abbildung 3.5 stellt die entsprechenden Auswahloptionen dar.

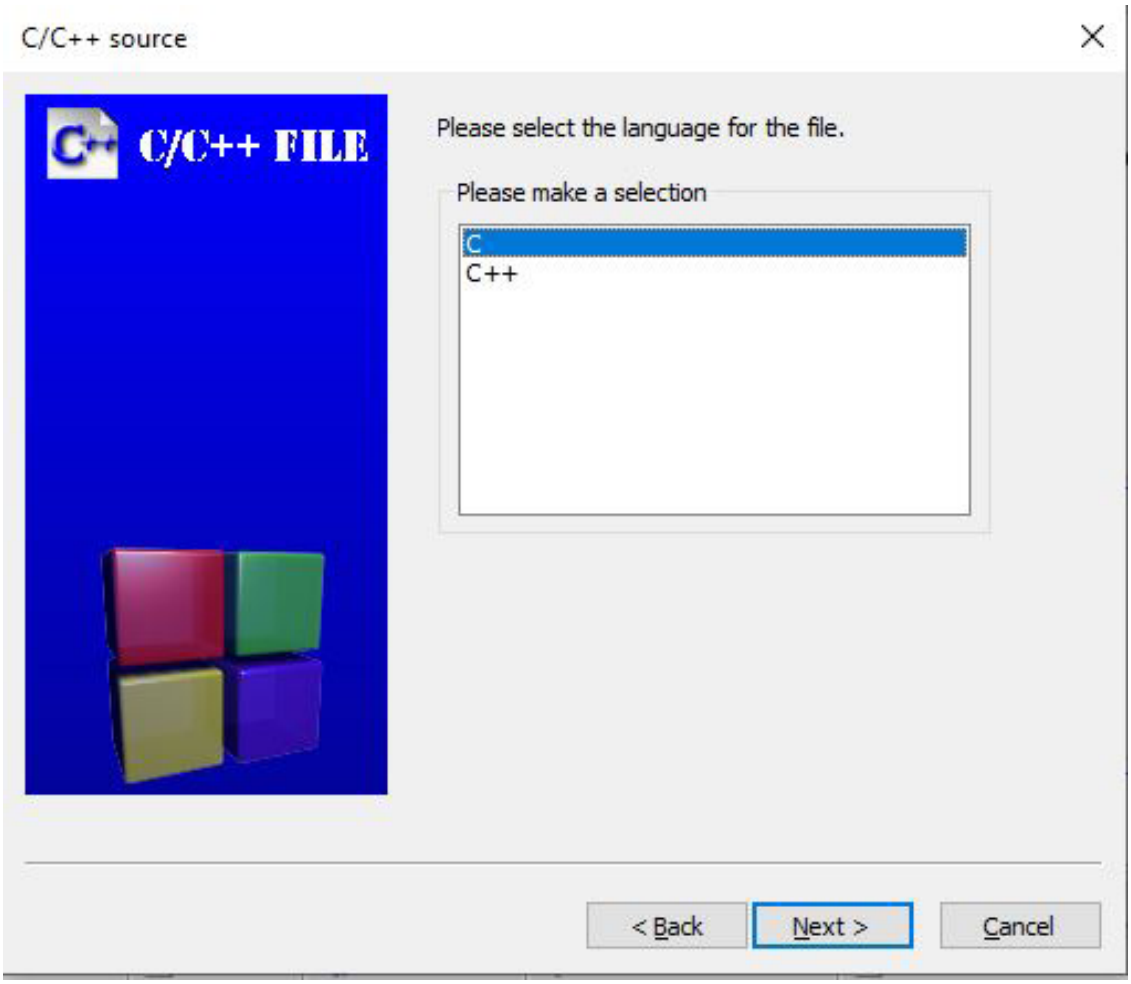

Abb. 3.5 Eine neue C-Datei erstellen

Wenn wir auch hierbei auf „Next" klicken, gelangen wir zu einem weiteren Fenster. Hier müssen wir nun vorgeben, wie unsere Datei heißen soll und in welchem Ordner wir sie abspeichern wollen. Die Wahl einer passenden Ordnerstruktur bleibt jedem Leser selbst überlassen. Für eine gute Übersichtlichkeit ist es jedoch empfehlenswert, für jedes Kapitel einen eigenen Ordner zu gestalten. Beim Dateinamen geben wir begruessung ein – da unser Programm eine kurze Begrüßung ausgeben soll. C-Programme erhalten immer die Endung .c. Wenn wir die vorgegebenen Einstellungen, die in Abbildung 3.6 zu sehen sind, nicht verändern, fügt Code::Blocks diese jedoch automatisch hinzu. Der vollständige Dateiname lautet dann begruessung.c.

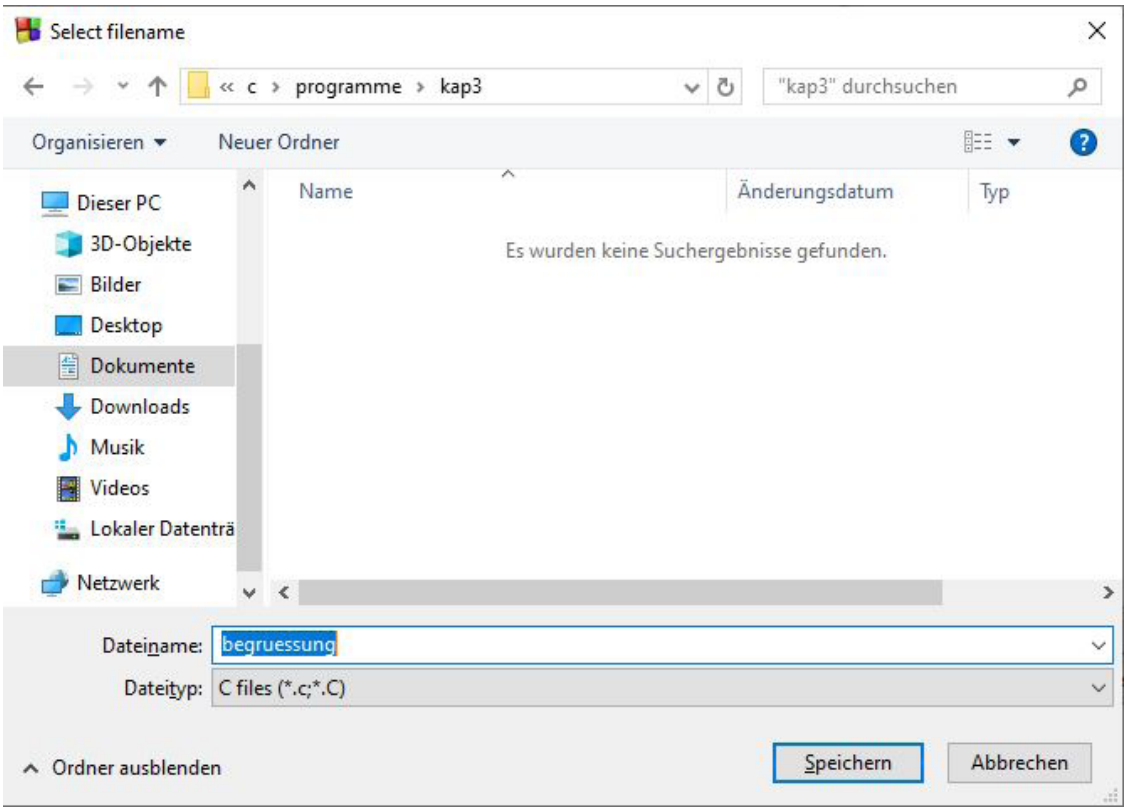

Abb. 3.6 Die Auswahl des Dateinamens

Nun müssen wir das abschließende Fenster noch mit einem Klick auf die Schaltfläche „Finish" bestätigen. Daraufhin kehren wir automatisch zum Hauptfenster zurück. Hier erscheint nun eine leere Eingabefläche, in die wir unseren Programmcode eintragen.

Ein C-Programm beginnt in der Regel mit der Zeile `int main ()`. Daher fügen wir diese nun in das leere Textfeld ein. Diese dient dazu, zu kennzeichnen, dass hier die `main()`-Funktion beginnt. C-Programme sind immer in Funktionen aufgebaut. Ein Programm kann aus vielen unterschiedlichen Funktionen bestehen. Daher müssen wir dabei stets kennzeichnen, wo der Einstiegspunkt gesetzt werden soll. Hierfür dient der Begriff `main`. Dieser sagt aus, dass es sich hierbei um die Hauptfunktion des Programms handelt. Daher beginnt die Ausführung stets an dieser Stelle.

Nun stellt sich noch die Frage, wozu der Ausdruck int und die leere Klammer dienen. Diese Bestandteile haben hierbei in erster Linie einen formalen Charakter. In Kapitel 8 werden wir uns ausführlicher mit Funktionen befassen. Dabei lernen wir, dass wir eine Funktion aus einem beliebigen Teil des Programms aufrufen können. Wir können ihr dabei Werte übergeben und auch Werte von ihr zurückerhalten. Dabei müssen wir jedoch

den Datentyp der Übergabe- und Rückgabewerte angeben. Der Begriff int sagt aus, dass unsere Hauptfunktion als Rückgabewert eine ganze Zahl hat. Die leere Klammer gibt an, dass wir keine Übergabewerte erwarten. Allerdings spielen diese Angaben für unser erstes Programm überhaupt keine Rolle. Daher müssen wir sie nicht weiter beachten. Leser, die hier noch nicht genau verstanden haben, wozu diese Ausdrücke dienen, können daher unbesorgt bleiben. Das wird in Kapitel 8 noch genau erläutert. Bislang ist es lediglich notwendig, zu wissen, dass wir diese Ausdrücke einfügen müssen, um den formalen Anforderungen an eine Funktion zu genügen.

Nun können wir uns dem Inhalt unseres Programms zuwenden. Dieses muss stets in einer geschweiften Klammer stehen. Viele Programmierer setzen die öffnende geschweifte Klammer in eine neue Zeile. Eine andere häufig verwendete Möglichkeit besteht darin, sie direkt an den oben genannten Ausdruck anzuschließen. Wir wählen für unsere Programme die zweite Alternative. Auf die Programmfunktionen hat dies jedoch keinerlei Auswirkungen. Wenn wir die Klammer einfügen und daraufhin die Enter-Taste betätigen, um einen Zeilenumbruch zu erzeugen, stellen wir außerdem fest, dass Code::Blocks automatisch eine Einrückung eingefügt hat. Das dient einer übersichtlicheren Gestaltung des Quellcodes. Doch hat auch diese Einrückung keinen Einfluss auf die Funktionen des Programms.

Nun können wir die Befehle eingeben, die unser Programm ausführen soll. Wie bereits geschrieben, soll es lediglich eine kleine Begrüßung ausgeben. Der Befehl für Textausgaben in C lautet `printf`. Darauf folgt eine Klammer, in die wir den Inhalt der Ausgabe schreiben können. Wenn wir hier direkt einen Text eingeben, müssen wir diesen stets in Anführungszeichen setzen. Außerdem müssen wir in C jeden Befehl mit einem Semikolon beenden. Die komplette Befehlszeile sieht dann so aus:

```
1  printf("Willkommen zum C-Kurs!");
```

Wenn wir die geschweifte Klammer nun schließen, könnten wir das Programm in Code::Blocks bereits ausführen – allerdings nur in Code::Blocks oder in ähnlichen Entwicklungsumgebungen. Das liegt daran, dass diese recht fehlertolerant sind und auch unvollständigen Code ausführen. Wenn wir dieses Programm mit einem herkömmlichen Compiler kompilieren, kommt es jedoch zu einer Fehlermeldung. Das liegt daran, dass dieser den Befehl `printf()` nicht kennt. Die Programmiersprache C enthält überhaupt keine eigenen Befehle, die unmittelbar verfügbar sind. Diese sind alle in Bi-

bliotheken angeordnet. Wenn wir ein Programm ausführen möchten, müssen wir dem Compiler daher mitteilen, welche Bibliotheken er für die Ausführung verwenden soll. Der `printf()`-Befehl ist in einer Bibliothek mit der Bezeichnung **stdio.h** enthalten.

Um diese aufzurufen, müssen wir zunächst das Rautezeichen (#) und daraufhin den Begriff `include` in das Programm einfügen. Danach folgt in spitzen Klammern der Name der Bibliothek:

```
1  #include <stdio.h>
```

Diese Zeile müssen wir ganz oben in unser Programm einfügen – noch bevor wir die `main()`-Funktion öffnen. Nun ist unser Programm eigentlich bereits korrekt und funktionsfähig. Allerdings ist es üblich, noch eine weitere Zeile in das Programm einzufügen – direkt bevor wir die geschweifte Klammer schließen:

```
1  return 0;
```

Wie bereits beschrieben, hat unsere `main()`-Funktion einen Rückgabewert. Mit dem aufgeführten Befehl geben wir diesem den Wert 0. Zwar ist es nicht zwingend notwendig, an dieser Stelle einen konkreten Wert anzugeben. Wenn wir dies jedoch nicht tun, hat die Funktion keinen klar definierten Rückgabewert. Das kann in manchen Situationen Probleme bereiten. Daher ist es sinnvoll, sich von Anfang an anzugewöhnen, diese Zeile am Ende der `main()`-Funktion einzufügen. Unser komplettes Programm sieht dann so aus:

```
2  #include <stdio.h>
3  int main(){
4      printf("Willkommen zum C-Kurs!");
5      return 0;
6  }
```

Abbildung 3.7 zeigt, wie der Programmcode innerhalb der IDE dargestellt wird. Das macht deutlich, dass die farbigen Kennzeichnungen der Schlüsselbegriffe und die automatischen Einrückungen zu einer wesentlich übersichtlicheren Gestaltung beitragen.

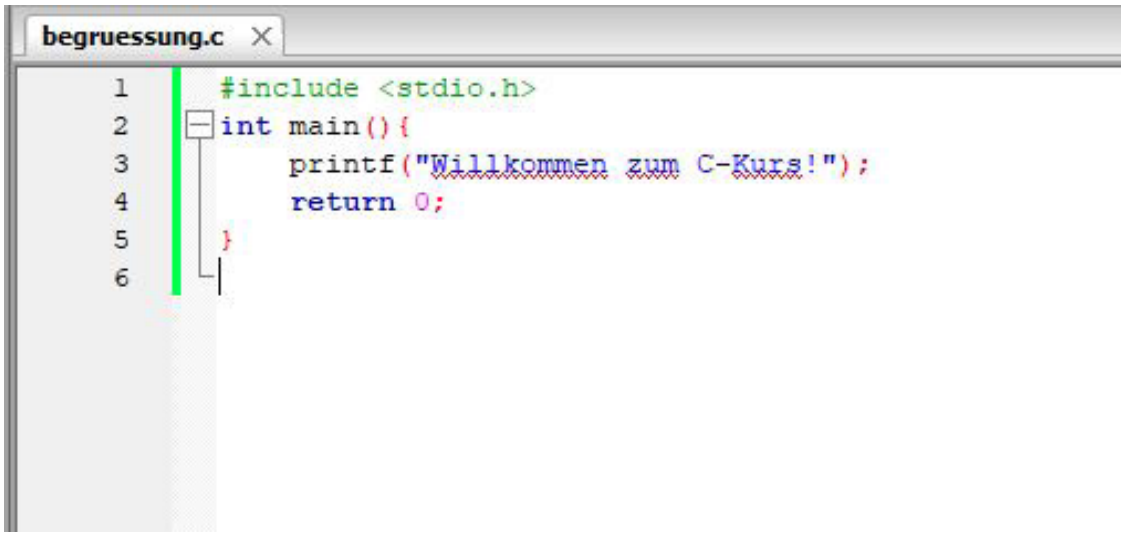

Abb. 3.7 Die Darstellung des Programmcodes in der IDE Code::Blocks

3.2 Das Programm in Code::Blocks ausführen

Der Code für unser erstes Programm ist bereits fertiggestellt. Nun wollen wir dieses selbstverständlich auch ausführen. Das ist mit Code::Blocks jedoch ganz einfach. Zunächst müssen wir das Programm abspeichern. Dafür können wir den entsprechenden Befehl in der Menüleiste auswählen oder ganz einfach die Tastenkombination Strg+S verwenden.

Danach klicken wir in der Menüleiste auf den Begriff „Build". Hier wählen wir nun „Build and run" aus, so wie dies in Abbildung 3.8 zu sehen ist.

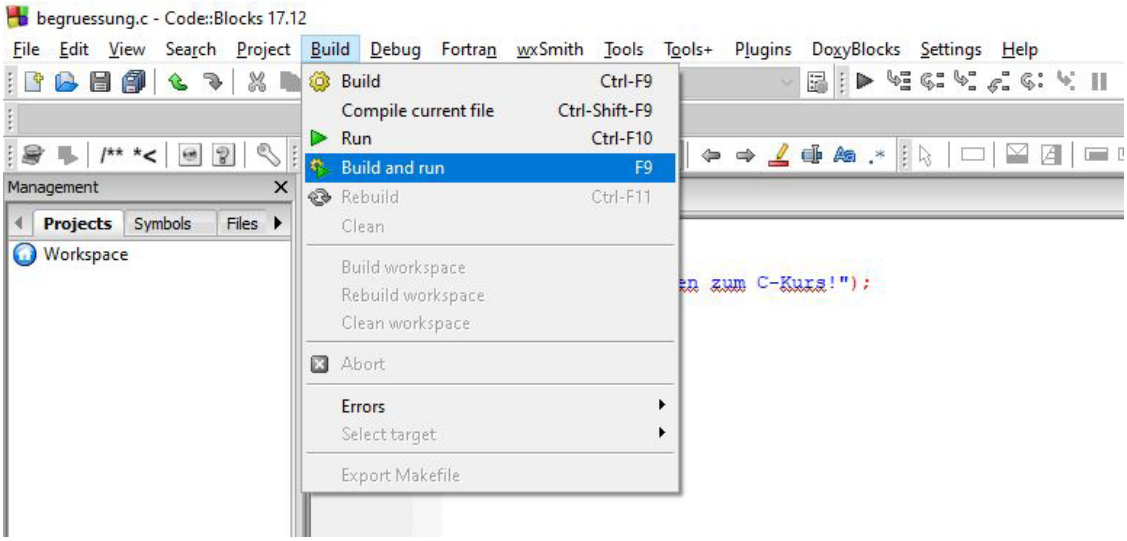

Abb. 3.8 So führen wir ein Programm in Code::Blocks aus

Alternativ dazu können wir auch die Werkzeugleiste verwenden. Hier erkennen wir genau das gleiche Symbol, das neben dem Ausdruck „Build and run" zu sehen ist: ein kleines Zahnrad mit einem grünen Dreieck. Wenn wir dieses anklicken, hat das genau den gleichen Effekt. Eine weitere Möglichkeit besteht darin, einfach die F9-Taste zu betätigen.

Wenn wir eine der aufgeführten Methoden anwenden, öffnet sich ein neues Fenster. Darin erscheint nun der Text, den wir über unser Programm ausgeben. Abbildung 3.9 zeigt, wie dieses Fenster aussieht.

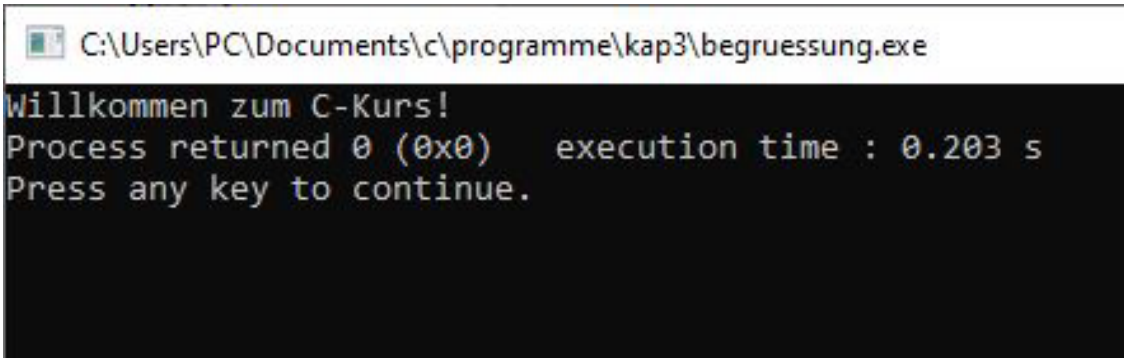

Abb. 3.9 Die Ausgabe unseres ersten Programms

Das zeigt, dass es mit Code::Blocks ganz einfach ist, das Programm zu kompilieren und auszuführen. Dennoch lohnt sich ein genauer Blick darauf, was hierbei passiert ist. Wenn wir den Ordner öffnen, in dem wir unser Programm abgespeichert haben, sehen wir, dass hier nun zwei neue Dateien entstanden sind: begruessung.o und begruessung.exe. Das bezieht sich jedoch nur auf Windows-Rechner. Unter anderen Betriebssystemen kann die zweite Datei auch eine andere Endung aufweisen.

Der Compiler hat nun aus unserem Quellcode ein ausführbares Programm erzeugt. Dieses erlaubt es, das Programm wie gerade gezeigt auszuführen. Dabei handelt es sich um die exe-Datei. Bei der o-Datei handelt es sich hingegen um ein Zwischenprodukt, das der Compiler anfertigt. Diese hat für uns jedoch keine weitere Bedeutung.

Das zeigt, dass beim Kompilieren des Programms eine neue Datei entstanden ist, die dauerhaft bestehen bleibt und die es erlaubt, das Programm auszuführen. Wenn wir das Programm nun erneut aufrufen, ist es daher nicht mehr notwendig, es zu kompilieren. Falls wir es lediglich ausführen wollen, reicht es aus, in der Menüleiste auf den Begriff „Build" und daraufhin auf „Run" zu klicken – beziehungsweise auf das grüne Dreieck in der Werkzeugleiste. Dabei müssen wir jedoch beachten, dass es nach jeder Änderung am Quellcode notwendig ist, das Programm erneut zu

kompilieren. Sonst werden diese Neuerungen bei der Ausführung nicht berücksichtigt.

3.3 Die klassische Vorgehensweise zum Kompilieren und Ausführen eines C-Programms

Um die Abläufe beim Kompilieren und Ausführen des Programms besser zu verdeutlichen, wollen wir nun einmal die herkömmliche Methode hierfür anwenden. Daran wird ersichtlich, wie die einzelnen Schritte ablaufen.

Um ein Programm auf die herkömmliche Weise auszuführen, müssen wir den Quellcode in einem gewöhnlichen Texteditor verfassen. Damit wir keine weiteren Programme installieren müssen, verwenden wir für dieses Beispiel die Software, die bereits auf dem Betriebssystem installiert ist. Unter Windows ist das der Microsoft Editor, unter Linux je nach Edition GEdit, Kate oder ein ähnliches Programm und unter MacOS steht uns TextEdit zur Verfügung. Nun öffnen wir das entsprechende Programm. Für dieses Beispiel können wir einfach den Programmcode aus dem vorherigen Beispiel kopieren und hier einfügen. Windows-Nutzer werden dabei sofort erkennen, dass der Standard-Texteditor hierbei nur einen minimalen Funktionsumfang bietet und dass dabei auch die Darstellung nur wenig ansprechend ist. Die übrigen genannten Betriebssysteme werden bereits mit einem etwas höherwertigen Texteditor ausgeliefert.

Nun speichern wir die Datei im gleichen Ordner wie unser erstes Beispiel ab – unter der Bezeichnung **begruessung2.c**. Notepad-Nutzer müssen hierbei darauf achten, die Standard-Auswahl für das Dateiformat aufzuheben und stattdessen „Alle Dateien" auszuwählen. Sonst speichert der Microsoft Editor die Datei mit der Endung .txt ab. Abbildung 3.10 zeigt, wie wir dabei vorgehen müssen.

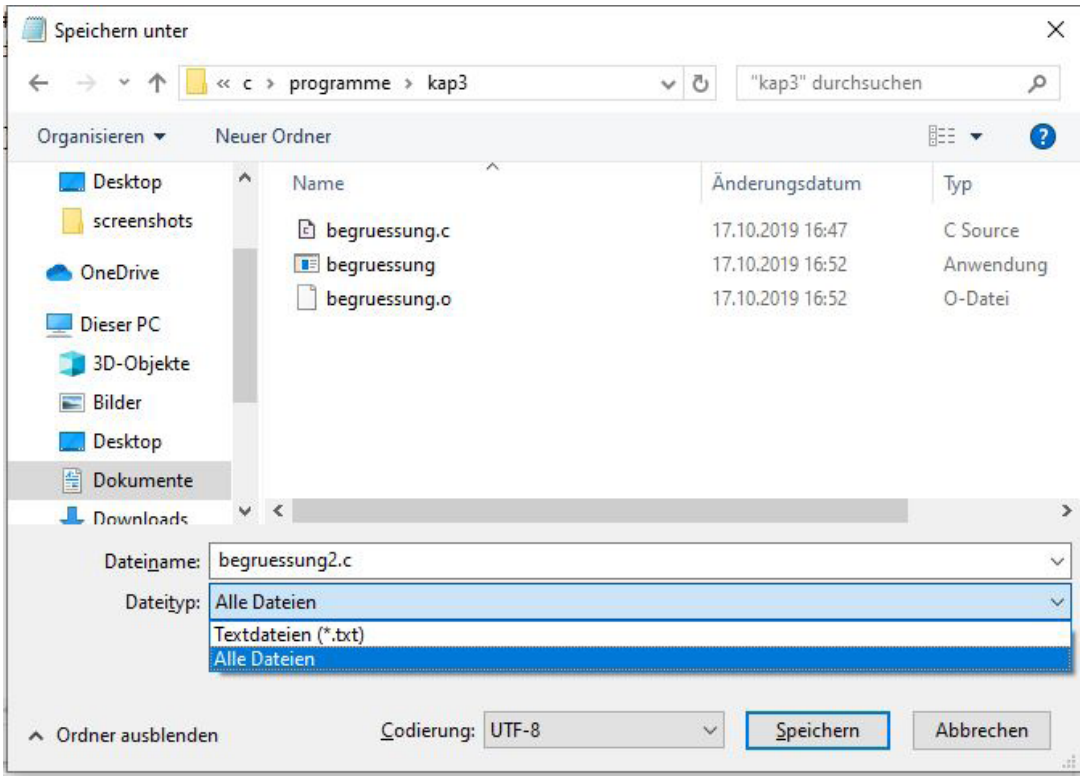

Abb. 3.10 Die Datei mit der richtigen Endung abspeichern

Der nächste Schritt besteht darin, das Programm zu kompilieren. Den Compiler, den wir hierfür benötigen, haben wir bereits zusammen mit Code::Blocks installiert. Diesen können wir auch außerhalb der IDE verwenden. Allerdings ist hierfür noch eine kleine Vorbereitungsmaßnahme erforderlich. Damit wir den Compiler aus jedem beliebigen Ordner aufrufen können, müssen wir eine sogenannte Pfadvariable vorgeben. Diese macht die entsprechende Funktion in jedem Verzeichnis des Computers verfügbar. Dazu rufen wir die erweiterten Systemeinstellungen auf. Dafür gibt es mehrere Möglichkeiten. Besonders einfach ist es, den Begriff „Erweiterte Sytemeinstellungen" in der Windows-Suchfunktion einzugeben. Alternativ dazu ist es möglich, den Windows-Explorer (Windows-Taste + E) zu starten. Wenn man hier mit der rechten Maustaste auf „Dieser PC" klickt und dann im Kontext-Menü den Begriff „Eigenschaften" auswählt, gelangt man zur Startseite der Systemsteuerung. Hier erscheint auf der linken Seite ein Menü, in dem wir die erweiterten Systemeinstellungen auswählen können.

Wenn wir die erweiterten Systemeinstellungen aufgerufen haben, erscheint wie in Abbildung 3.11 zu sehen eine Schaltfläche mit der Aufschrift „Umgebungsvariablen". Diese rufen wir auf.

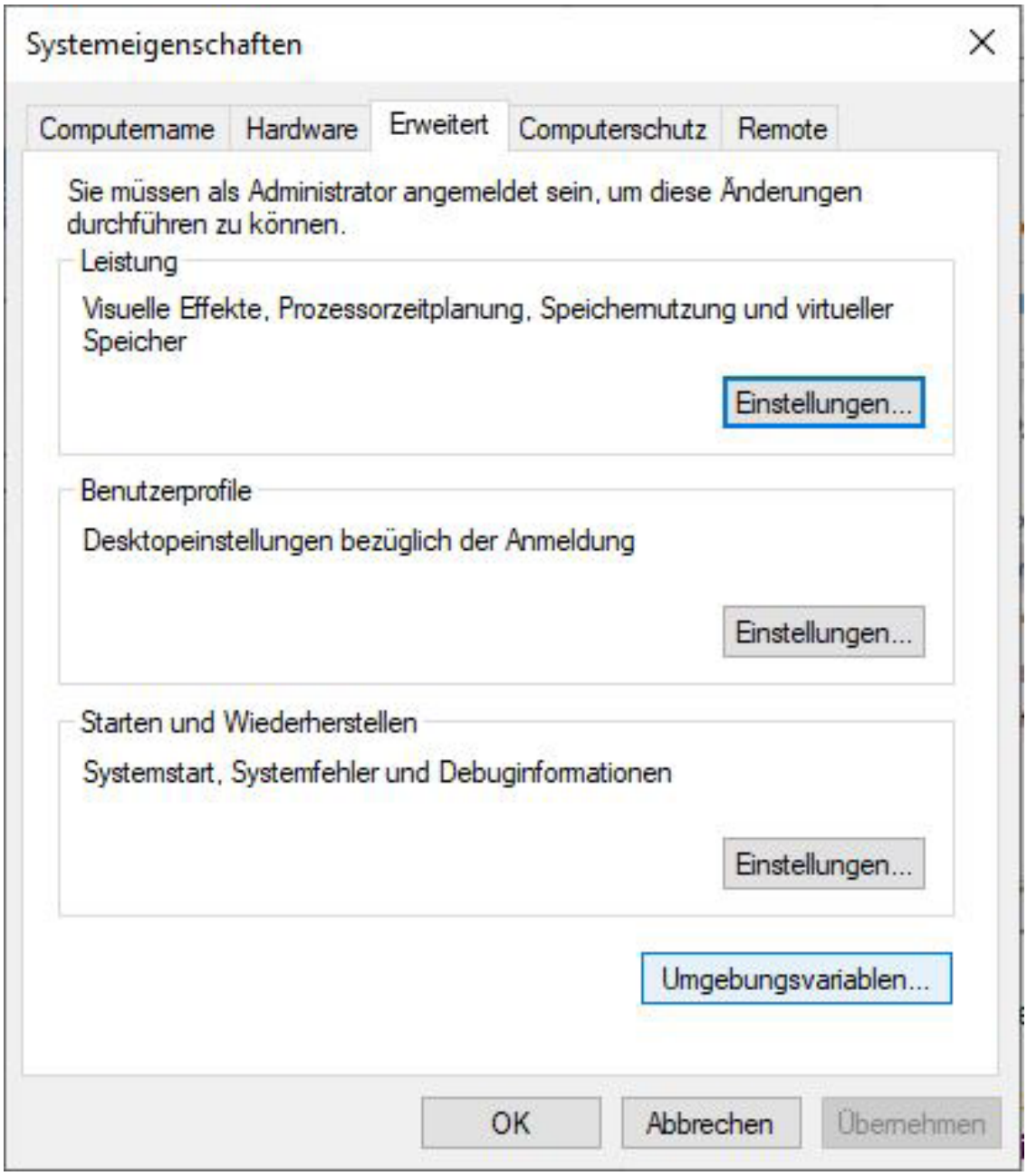

Abb. 3.11 Die erweiterte Systemeinstellungen

Daraufhin erscheint ein weiteres Fenster. Dieses ist in Abbildung 3.12 zu sehen. Im unteren Bereich suchen wir nun nach einem Eintrag mit der Bezeichnung „Path". Dieser ist in der Regel bereits vorhanden. Trifft das nicht zu, müssen wir ihn neu erstellen.

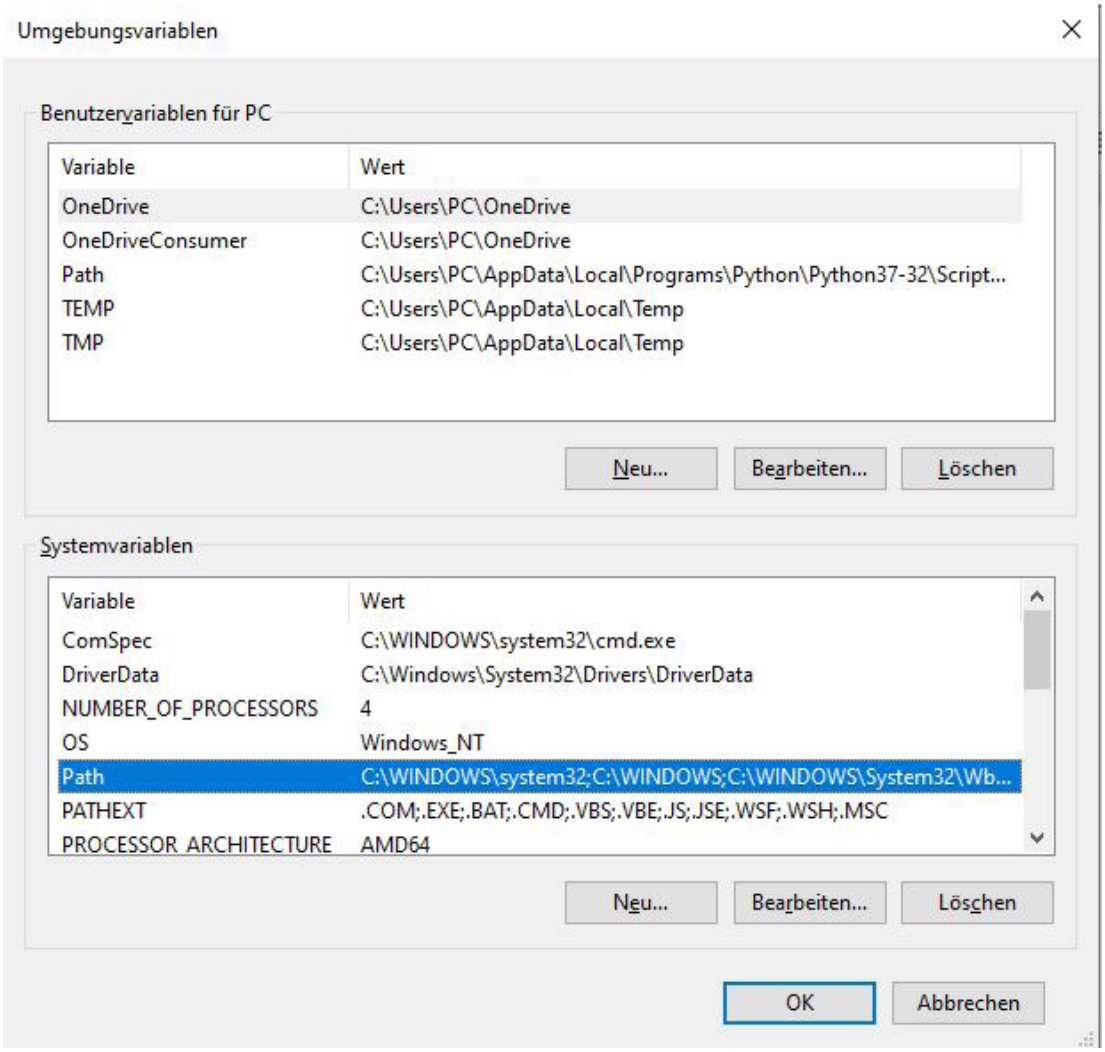

Abb. 3.12 Der Eintrag für die Anpassung der Umgebungsvariablen

Wenn der entsprechende Eintrag bereits vorhanden ist, klicken wir im nächsten Schritt auf „Bearbeiten". Daraufhin öffnet sich ein weiteres Fenster. Dort klicken wir dann auf „Neu". Jetzt müssen wir den Pfad des Verzeichnisses angeben, in dem sich unser Compiler befindet. Wenn wir bei dessen Installation die Standardeinstellungen nicht verändert haben, müssen wir folgenden Eintrag hinzufügen: `C:\MinGW\bin`. Abbildung 3.13 verdeutlicht dies nochmals. Daraufhin müssen wir alle geöffneten Fenster mit „OK" bestätigen. Nun können wir den Compiler in jedem beliebigen Verzeichnis verwenden.

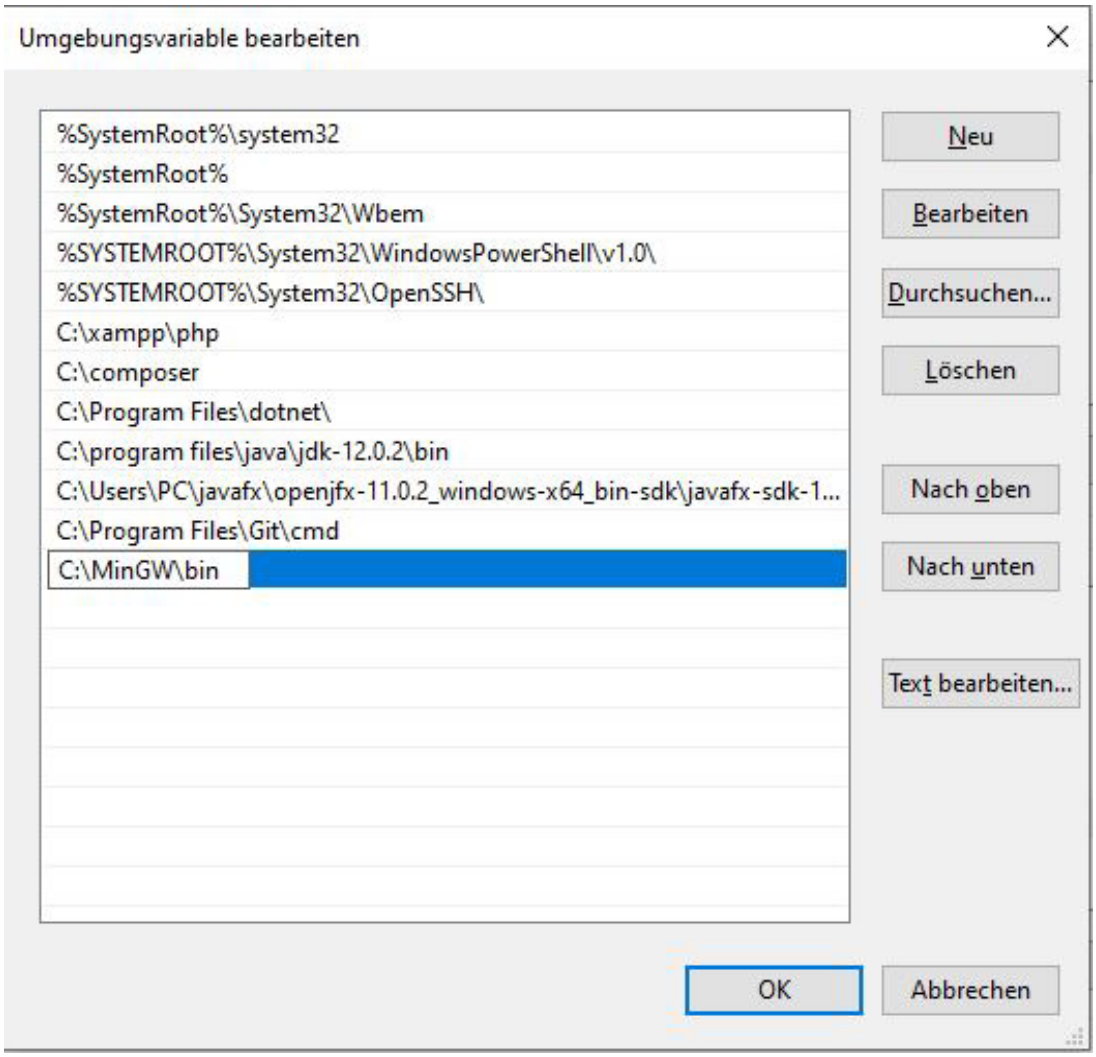

Abb. 3.13 Der Eintrag für die Pfadvariable

Jetzt können wir das Programm kompilieren und ausführen. Dazu verwenden wir einen *Kommandozeileninterpreter*. Unter Windows steht dieser unter der Bezeichnung cmd.exe zur Verfügung. Linux und MacOS verwenden hierfür die Bezeichnung Terminal. Hier müssen wir dann zunächst in das entsprechende Verzeichnis navigieren. Dazu verwenden wir den Befehl cd gefolgt vom Namen des Ordners, in den wir wechseln wollen. Auf diese Weise rufen wir das Verzeichnis auf, in dem wir unsere Datei abgespeichert haben. Danach geben wir den Befehl zum Kompilieren des Programms ein. Um den Compiler aufzurufen, verwenden wir den Ausdruck gcc. Um diesem mitzuteilen, dass er direkt ein ausführbares Programm erstellen soll, fügen wir den Ausdruck -o ein. Daraufhin müssen wir angeben, welchen Namen die ausführbare Datei bekommen soll – allerdings ohne Dateiendung, da diese automatisch vorgegeben wird. Als Letztes müssen wir noch hinzufügen, welche Datei eigentlich kompiliert werden soll – in diesem Beispiel **begruessung2.c**. Der komplette Befehl sieht dann so aus:

```
1 gcc -o begruessung2 begruessung2.c
```

Wenn wir nun den Ordner nochmals aufrufen, stellen wir fest, dass jetzt eine weitere Datei hinzugekommen ist: **begruessung2.exe**. In diesem Fall wurde keine .o-Datei erstellt. Da wir diese jedoch nicht benötigen, stellt das kein Problem dar. Da das Programm jetzt kompiliert ist, können wir es ausführen. Dazu geben wir lediglich den Dateinamen an – ohne die Endung .exe. Daraufhin erscheint wieder der Ausgabetext, den wir in unserem Programmcode vorgegeben haben. Abbildung 3.14 zeigt den kompletten Ablauf.

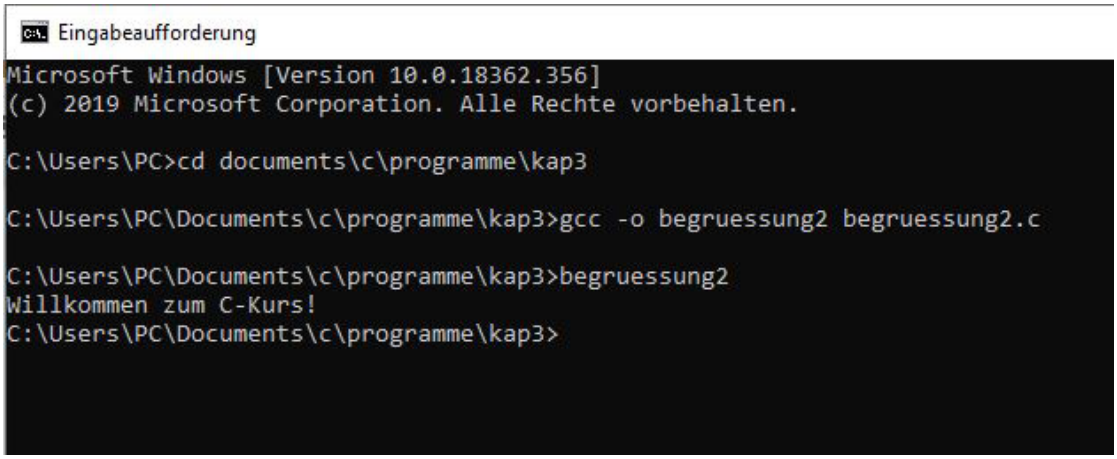

Abb. 3.14 Die Kompilierung und die Ausführung des Programms über den Kommandozeileninterpreter

An diesem Beispiel wird deutlich, wie der Prozess der Programmentwicklung ursprünglich ablief. Gleichzeitig zeigt dies, welche Vereinfachung die IDE bei der Erstellung eines Programms darstellt. Wenn man davon ausgeht, dass man ein umfangreicheres Programm während des Entwicklungsprozesses mehrere Hundert Male kompilieren und ausführen muss, wird klar, welch enorme Effizienzsteigerung hiermit verbunden ist.

3.4 Übungsaufgabe: eigene einfache Programme erstellen

Um umfassende Kenntnisse im Bereich der Programmierung zu erlangen, ist es sehr wichtig, eigene Programme zu erstellen. Nur wenn man selbst versucht, eine Lösung für eine bestimmte Aufgabe zu finden, prägen sich die entsprechenden Techniken und Befehle richtig ein. Deshalb steht am Ende der meisten Kapitel eine kleine Übungsaufgabe. Diese gibt dem Leser die Möglichkeit, das Erlernte anzuwenden und auf diese Weise zu vertiefen. Die praktischen Übungen helfen dabei, die notwendigen Fähigkeiten zu erwerben, um auch eigenständig Programme zu erstellen.

Den Schwerpunkt stellen dabei jeweils die Inhalte aus dem entsprechenden Kapitel dar. Doch selbstverständlich werden auch die Techniken aus den vorherigen Abschnitten als bekannt vorausgesetzt. In vielen Fällen sind diese ebenfalls notwendig, um die Aufgaben zu lösen. Kenntnisse, die noch nicht vermittelt wurden, sind dabei in der Regel nicht erforderlich. Lediglich in einigen Ausnahmefällen müssen wir für die Erstellung der Programme auch neue Befehle verwenden. Diese werden dann jedoch kurz erklärt.

Zum Abschluss der Aufgaben steht jeweils eine Musterlösung. Diese dient dazu, die Ergebnisse abzugleichen. Auch wenn man einmal selbst nicht weiterkommt, ist es möglich, hier nachzuschauen. Es ist jedoch ratsam, stets zu versuchen, die Aufgaben eigenständig zu lösen.

Da es in der Informatik für die meisten Probleme mehrere Lösungen gibt, muss die Musterlösung nicht immer genau mit Ihrem Programm übereinstimmen. Das bedeutet jedoch nicht, dass Sie einen Fehler gemacht haben. Wenn das Programm alle Anforderungen erfüllt und sich problemlos ausführen lässt, ist die Aufgabe dennoch korrekt gelöst.

Um die Kenntnisse aus diesem Kapitel zu vertiefen, sollen nun zwei kleine Programme erstellt werden:

1. Schreiben Sie ein Programm, das die Begrüßungsfloskel „Hallo Welt!" ausgibt. Dieses Beispiel wird in vielen Lehrbüchern als erste Programmbeispiel verwendet. Daher werden solch einfache Beispiele häufig auch als Hallo-Welt-Programm bezeichnet.
2. Gestalten Sie ein Programm mit zwei Ausgabebefehlen beliebigen Inhalts. Damit der zweite von ihnen in einer neuen Zeile erscheint, müssen Sie an den ersten Befehl einen Zeilenumbruch anhängen. Dazu dient der Ausdruck `\n`. Dieser wird nach dem Text innerhalb der Anführungszeichen hinzugefügt.

Lösungen:

1.

```
1  #include <stdio.h>
2  int main(){
3      printf("Hallo Welt!");
4      return 0;
5  }
```

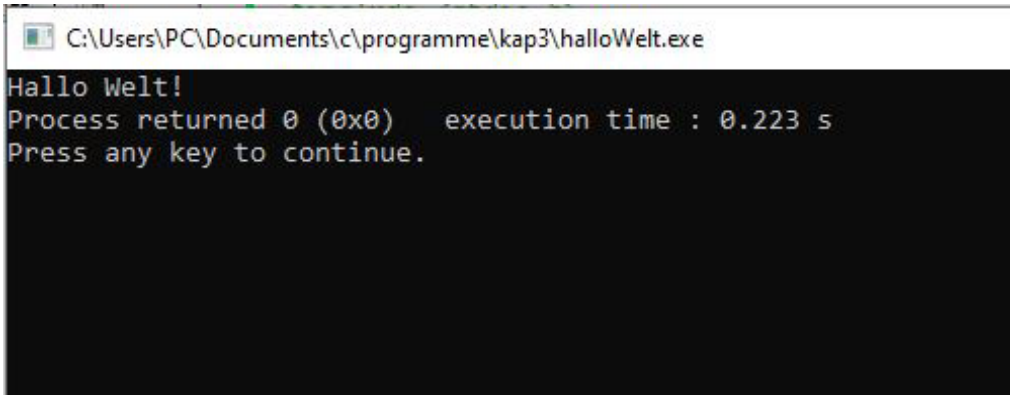

Abb. 3.15 Die Ausgabe der bekannten Begrüßungs-Floskel

2.

```
1  #include <stdio.h>
2  int main(){
3      printf("Willkommen zum C-Kurs!\n");
4      printf("Hier lernen Sie die Programmiersprache C kennen.");
5      return 0;
6  }
```

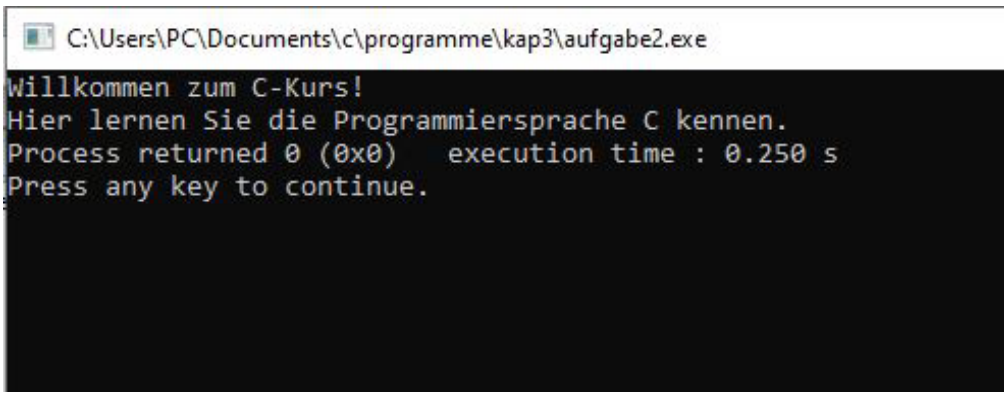

Abb. 3.16 Die mehrzeilige Textausgabe

Alle Programmcodes aus diesem Buch sind als PDF zum Download verfügbar. Dadurch müssen Sie sie nicht abtippen:
https://bmu-verlag.de/c

Außerdem erhalten Sie die eBook Ausgabe zum Buch im PDF Format kostenlos auf unserer Website:

https://bmu-verlag.de/c
Downloadcode: siehe Kapitel 19