1 Einführung

1.1 Einleitung

1.1.1 Programmieren lernen in 14 Tagen

Mit diesem Buch erhalten Sie einen praktischen Einstieg in das Programmieren mit C++. Sicherlich endet Ihre Lernkurve nicht mit der letzten Seite, aber bis dahin werden Sie Einblicke in die verschiedenen Funktionalitäten, Stärken und Fallstricke der Sprache bekommen haben. Dieses Buch bringt Ihnen nicht nur reines C++ bei, sondern zeigt Ihnen auch, wie Sie es auf einfache Weise mit Open-Source-Bibliotheken erweitern, sodass Sie grafische Benutzeroberflächen erstellen oder mit Internetservern kommunizieren können. Es endet mit einem Ausblick darauf, wie Sie Ihr gewonnenes Wissen im Programmieralltag einsetzen und selbstständig erweitern können.

Wenn Sie Zeit genug haben, können Sie jeden Tag ein neues Kapitel durcharbeiten und so innerhalb von zwei Wochen Programmieren lernen. Alle Erklärungen sind leicht verständlich und setzen keine Vorkenntnisse voraus. Sie werden schnell erste Erfolge erzielen und Freude an der Programmierung finden.

1.1.2 Wie man dieses Buch liest

Dieses Buch gibt Ihnen einen Einblick in C++, und zwar in der aktuell verfügbaren Version C++17. Die Unterschiede der verschiedenen Versionen sind für dieses Buch nicht sonderlich relevant – die meisten hier verwendeten Konzepte gibt es schon seit C++11, welches meiner Meinung nach ein Quantensprung für die Modernisierung von C++ war.

Insbesondere Universitäten oder berufliche Fortbildungen beginnen das Lehren von C++ oft über den Vorgänger C und dessen sehr hardware-

nahen Konzepte. Dieses Buch verfährt jedoch andersherum: Sie als Leser sollen ohne Vorkenntnisse in die Lage versetzt werden, möglichst schnell erste Resultate zu erreichen. Dafür ist es nicht relevant, sich von Tag 1 an mit dem Speichermanagement und anderen Grundlagen auseinanderzusetzen. Der Vollständigkeit halber werde ich diese Dinge auch ansprechen, allerdings erst in den späteren Kapiteln.

Verweise auf Kapitel weiter hinten im Buch werden Ihnen häufiger begegnen. Dies ist zweierlei Dingen geschuldet

- Sie sollen einen Ausblick bekommen, in welche Richtungen das bereits Erlernte noch erweitert wird.
- Einige Konzepte bedingen sich gegenseitig, dennoch möchte ich sie Ihnen der Reihe nach präsentieren.

Lassen Sie sich daher nicht davon beirren, es ist ein bisschen wie beim Lernen einer normalen Sprache: Das Erlernen der Grammatik ist schwer, ohne Vokabeln zu kennen, reines Vokabellernen ohne die Anwendung macht aber auch keinen Spaß. Ich empfehle Ihnen, sich nicht den Anspruch aufzuerlegen, jedes Programmbeispiel auf Anhieb Zeichen für Zeichen zu verstehen, sondern im Zweifel Dinge erst mal hinzunehmen und gegebenenfalls später wiederaufzugreifen.

Am Ende eines jeden Kapitels finden Sie kleine Übungsaufgaben, mit denen Sie Ihr Wissen praktisch anwenden und vertiefen können. Manchen Übungen ist ein relativ eindeutiger Lösungshinweis beigelegt – dieser wird dann auf dem Kopf geschrieben, sodass Sie auch erst ohne ihn versuchen können, die Aufgabe zu lösen. Genauso finden Sie auch kurze Zwischenfragen innerhalb der Kapitel, deren Antwort ebenfalls auf dem Kopf geschrieben ist.

Neben den Übungsaufgaben möchte ich Sie ermuntern, auch schon innerhalb der Kapitel ein bisschen herumzuprobieren und den Pfad des Buches mal links und rechts zu verlassen. Ein Hinweis dazu: Leider ist C++ nicht immer besonders experimentierfreudig, sodass gerade in den ersten Kapiteln vielleicht noch weniger Spielraum für eigenständige Versuche ist.

Die Beispiele der ersten Kapitel werden allesamt Konsolenprogramme sein, denn diese sind leichter zu erstellen. Hierzu noch eine Vorwarnung: Die Konsole unter Windows hat standardmäßig Probleme mit Umlauten. Die Lösung hierfür bekommen Sie in Abschnitt 5.1 präsentiert – bis dahin verzichten Sie bei Ihren eigenen Experimenten auf Umlaute oder schreiben Sie ae, oe, ue.

Wie ich später in einem eigens dafür angelegten Abschnitt 2.5 noch ausführen möchte, ist die Welt des Programmierens sehr englischlastig. Viele Fachbegrif-

fe werden auch in deutschen Gesprächen in ihrer englischen Variante verwendet. Zulasten der Sprachschönheit werde ich in diesem Buch die tatsächliche Sprechweise deutscher Programmierer nachahmen, die sich durch einen Mix aus englischen und eingedeutschten Begriffen auszeichnet. Generell ergibt es auch Sinn, sich die englischen Begriffe bevorzugt zu merken, denn das macht die Problemlösung via Internetsuche deutlich einfacher – dazu mehr im Kapitel 14, »Der Alltag eines Programmierers«.

Aufgrund der besseren Lesbarkeit wird im Buch das generische Maskulinum verwendet. Gemeint sind jedoch immer alle Geschlechter.

Wenn Sie nicht sowieso die physikalische Ausgabe dieses Buches erworben haben, so hoffe ich, dass Sie das E-Book zumindest auf einem separaten Lesegerät aufrufen können, denn das ermöglicht es Ihnen, das Buch beim Programmieren der Beispielprogramme nebenher offen auf dem Schreibtisch zu haben, Informationen anzustreichen und noch mal zurückblättern zu können. Kleben Sie Marker an wichtige oder interessante Stellen, sodass Sie auch nach dem Durchlesen schnell die hilfreichen Stellen wiederfinden, sobald Sie sich an Ihre eigenen Projekte machen.

Die gedruckte Ausgabe bewahrt Sie auch vor einer verlockenden Abkürzung, nämlich die Programmbeispiele einfach 1:1: über die Zwischenablage zu kopieren. Das geht zwar schneller, als sie selbst abzutippen, ist aber deutlich weniger lehrreich. Um Verständnisfragen zu klären, finden Sie dennoch sämtliche Programmbeispiele des Buchs unter:

https://cpp.hasper.info/code_im_buch

1.1.3 Programmtexte, Lösungen und Glossar zum Download

Sie erhalten alle Begleitmaterialien des Buches wie Codebeispiele oder die Musterlösungen der Übungen hier zum Download:

https://cpp.hasper.info

Sie werden an den relevanten Stellen auch auf den vollständigen Link hingewiesen.

Neben dem Stichwortverzeichnis am Ende des Buches finden Sie ein Glossar mit Erklärungen der wichtigsten Begriffe dieses Buchs unter:

https://cpp.hasper.info/glossar

1.1.4 Fragen und Feedback

Unsere Verlagsprodukte werden mit großer Sorgfalt erstellt. Sollten Sie trotzdem einen Fehler bemerken oder eine andere Anmerkung zum Buch haben, freuen wir uns über eine direkte Rückmeldung an *lektorat@mitp.de*.

Falls es zu diesem Buch bereits eine Errata-Liste gibt, finden Sie diese unter

https://cpp.hasper.info/korrekturen/

Ich bedanke mich beim mitp-Lektorat und bei Manuel Landsmann für die Unterstützung und wünsche Ihnen viel Spaß beim Lesen!

Philipp Hasper

1.2 Über C++

C++ ist ein richtiger Altmeister unter den Programmiersprachen. Die Sprache wurde Anfang der Achtzigerjahre entwickelt und seitdem kontinuierlich erweitert. Der aktuellste einsetzbare Sprachstandard heißt C++17[1], und C++20 ist schon in der Umsetzung. Die Sprache hat sich wohl deshalb so lange gehalten und rangiert immer noch auf den Bestenlisten der meistverwendeten Programmiersprachen, weil sie gleichzeitig sowohl effiziente als auch verständlich geschriebene Programme erlaubt. Verständlichkeit ist natürlich immer relativ, und sicherlich kann man sagen, dass es mittlerweile intuitivere Sprachen gibt. Aber bezogen auf die Vielseitigkeit und die sehr tiefgreifenden Kontrollmöglichkeiten für den Programmierer ist und bleibt C++ ein Meilenstein in Bezug auf die Erlernbarkeit.

Wenn Sie einen Vergleich mit anderen, leichter zugänglichen Sprachen wie beispielsweise Python oder Java ziehen wollen, könnte man sagen: »C++ macht exakt das, was Sie sagen. Andere Sprachen machen das, von dem sie denken, dass Sie es wollen«. Offensichtlich hat beides seine Vor- und Nachteile.

C++ wurde zuerst als »C with Classes« (C mit Klassen) bezeichnet, da die Sprache als Erweiterung der auch heute ebenfalls noch populären Sprache C entstand. Dadurch ermöglicht C++ das Schreiben von objektorientierten Programmen, eine Technik, die im Verlauf des Buchs erklärt wird. C++ findet man in der Programmierung von Spielen, Betriebssystemen, Datenbanken,

1 Stand: April 2021

wissenschaftlichen Simulationen, Anwendungen der künstlichen Intelligenz, Raketen[2]… und sie wird von Start-ups bis hin zu Großkonzernen eingesetzt.

Wenn Sie sich jetzt fragen, warum 40 Jahre später sowohl die Erweiterung C++ als auch die anscheinend erweiterungsbedürftige Sprache C gleichzeitig populär sind: Es wird niemals »die eine Programmiersprache« geben, die sich gegen alle durchsetzt. Vielmehr sind sie alle als Teil eines Werkzeugkastens zu begreifen, aus der sich ein Programmierer für jedes neue Projekt das jeweils passende Hilfsmittel aussucht.

1.3 Einrichten der Programmierumgebung

Da C++ eine so weitverbreitete Sprache ist, gibt es auch eine Menge an unterschiedlichen Werkzeugen zur Entwicklung. Sie benötigen hierzu zwei Dinge: Einen *Texteditor* zum Schreiben des Codes und einen sogenannten *Compiler* zum Übersetzen in Maschinensprache.

Man kann sich diese beiden Programme selbst zusammenstellen, oder aber man greift zu einer *integrierten Entwicklungsumgebung* (geläufige englische Abkürzung: IDE, für Integrated Development Environment), die beides in sich vereint und obendrein noch weitere Werkzeuge zur Verfügung stellt: zum Beispiel eine automatische Codevervollständigung, die oft getippte Zeichenfolgen vorschlägt, oder einen sogenannten *Debugger*, der bei der Fehlersuche hilft (hierzu mehr gegen Ende des Buchs in Kapitel 13, »Fehlersuche leicht gemacht: Debugging«).

In diesem Buch verwenden wir Microsoft Visual Studio in der kostenlosen Community-Edition, welches Sie hier für Windows herunterladen können: *https://visualstudio.microsoft.com/de/vs/community/*. Falls Sie ein anderes Betriebssystem verwenden, benötigen Sie eine andere IDE, die Inhalte diese Buchs bleiben jedoch gültig.

Was, wenn Sie nicht mit Windows arbeiten?

Keine Sorge, Sie können mit diesem Buch trotzdem C++ lernen, nur ist das Einrichten der Programmierumgebung etwas komplizierter. Ich empfehle Ihnen die Installation des kostenlosen Editors *Visual Studio Code*, ebenfalls von Microsoft. Trotz der Namensähnlichkeit zu Visual Studio ist dieses Programm nur ein Texteditor, den man aber mit Erweiterungen zu einer vollwertigen Entwicklungsumge-

2 Raketenfirmen sind entweder älter als C++ oder besonders geheimniskrämerisch. Allerdings haben Mitarbeiter von SpaceX ihre Verwendung von C++ bestätigt.

bung machen kann. Eine englische Installationsanleitung finden Sie hier: *https://code.visualstudio.com/docs/languages/cpp*. Die Benutzeroberfläche ist anders als die Visual-Studio-Screenshots in diesem Buch, Sie sollten sich allerdings dennoch gut zurechtfinden.

Führen Sie folgende Schritte aus, um Visual Studio zu installieren:

1. Laden Sie das Installationspaket von *https://visualstudio.microsoft.com/de/vs/community/* herunter und führen Sie es aus.
2. Visual Studio kann für verschiedene Programmiersprachen benutzt werden, für die man vorher die entsprechenden Module installieren muss. Setzen Sie daher den Haken in der Kachel Desktopentwicklung mit C++ (Abbildung 1.1).

Abb. 1.1: C++-Modul im Installationsdialog aktivieren

3. Wechseln Sie am oberen Fensterrand den Tab zu Sprachpakete.
4. Aktivieren Sie unbedingt das englische Sprachpaket, das wir später aus technischen Gründen brauchen werden. Das deutsche Paket ist hilfreich, weil die Erklärungen und Screenshots in diesem Buch auf Deutsch sind. Sie sollten es daher ebenfalls installieren (Abbildung 1.2).

Abb. 1.2: Das englische und deutsche Sprachpaket im Installationsdialog aktivieren

5. Klicken Sie auf INSTALLIEREN in der rechten unteren Ecke und warten Sie, bis der Vorgang abgeschlossen wurde. Das kann eine Weile dauern.
6. Nun werden Sie gebeten, einen Visual-Studio-Account anzulegen. Sie können diesen Schritt zwar fürs Erste überspringen, nach spätestens 30 Tagen werden Sie jedoch erneut dazu aufgefordert, um das Programm weiterhin kostenlos nutzen zu können (Abbildung 1.3).

Abb. 1.3: Entwicklerkonto einrichten oder überspringen

7. Zuletzt können Sie noch ein Farbschema auswählen und dann über die gleichnamige Schaltfläche Visual Studio starten.

Sie werden mit einem Startfenster begrüßt (Abbildung 1.4), welches Sie im nächsten Kapitel nutzen, um ein neues Projekt zu erstellen.

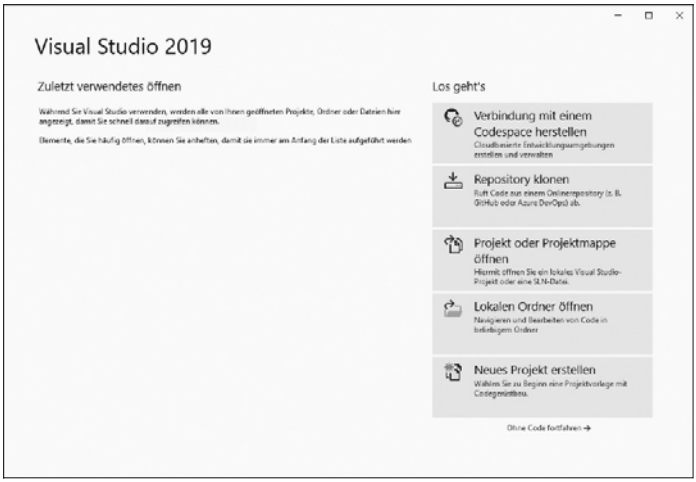

Abb. 1.4: Das Visual-Studio-Startfenster

Das erste Programm: »Hello World«

Es hat sich etabliert, beim Lernen einer neuen Programmiersprache immer mit einem sogenannten »Hello World« zu beginnen. Das ist das einfachste mögliche Programm, welches nur den Text »Hello World« ausgibt und sich dann wieder beendet. Es verdeutlicht kurz und knapp, was mindestens benötigt wird, um mit dem Programmieren loslegen zu können.

Und das hier ist »Hello World« in C++:

```
001 #include <iostream>
002
003 int main()
004 {
005   std::cout << "Hello World" << std::endl;
006   return 0;
007 }
```

Sieht erst mal seltsam aus? Keine Sorge, wir werden diesen sogenannten *Code* in diesem Kapitel Stück für Stück durchgehen. Sie werden das Grundgerüst eines jeden C++-Programms kennenlernen und dann das Konzept von Variablen erlernen. Im Verlauf des Kapitels erweitern wir nach und nach den obigen Code um neu gelernte Konzepte und beenden die Lerneinheit mit ein paar Übungen.

Die Nummern an der linken Seite sind übrigens nicht Teil des Programmcodes – das sind Zeilennummern, damit wir einfacher über den Inhalt sprechen können.

2.1 Anlegen eines neuen Projekts

Jedes neue Programmierprojekt beginnt in seinem eigenen, leeren Ordner auf der Festplatte. Ganz so, als würden Sie ein neues Textdokument beginnen, nur dass es statt einer einzelnen Datei ein ganzer Ordner voller Programmierdateien werden wird. Das Aufsetzen eines Projekts läuft immer ähnlich ab, und so können Sie die folgende Liste als Spickzettel nutzen, wann immer Sie ein neues starten.

Diese Anleitung ist spezifisch für Visual Studio geschrieben und weicht bei anderen Entwicklungsumgebungen leicht ab.

1. Legen Sie einen neuen Ordner `Programmierprojekte` an, zum Beispiel als Unterordner von `Dokumente`. Sie können auch einen beliebigen anderen Ordnernamen wählen.
2. Starten Sie Visual Studio über das Startmenü.
3. Das Startfenster erscheint (Abbildung 1.4). Sollte dies nicht der Fall sein, sondern sich die Programmierumgebung direkt öffnen, können Sie das Startfenster am oberen Bildschirmrand über DATEI|STARTFENSTER aufrufen.

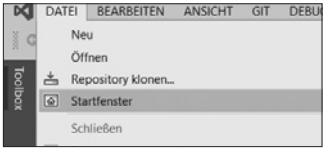

Abb. 2.1: Das Visual-Studio-Startfenster öffnen

4. Klicken Sie im Startfenster auf NEUES PROJEKT ERSTELLEN.

Wählen Sie rechts die Vorlage KONSOLEN-APP aus, und klicken Sie auf WEITER. Die anderen Projektvorlagen werden in diesem Buch nicht verwendet.

Abb. 2.2: Ein Projekt nach der Vorlage »Konsolen-App« erstellen

5. Geben Sie dem Projekt einen Namen, beispielsweise »Hello World«, und wählen Sie den im ersten Schritt erstellten Ordner `Programmierprojekte` als Speicherort aus. Aktivieren Sie den Haken neben Platzieren Sie die Projektmappe und das Projekt im selben Verzeichnis.

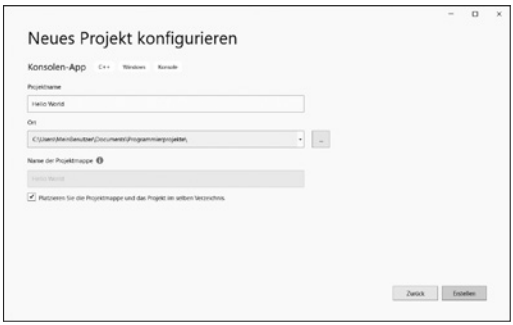

Abb. 2.3: Speicherort des neuen Projekts wählen

6. Nach dem Klick auf Erstellen öffnet sich die Programmierumgebung.

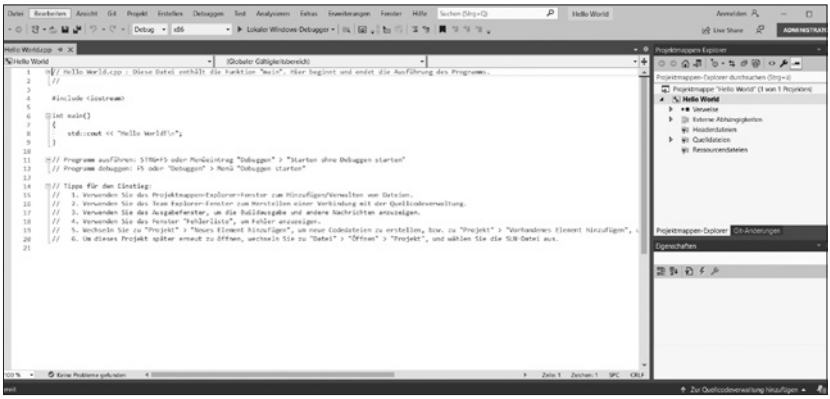

Abb. 2.4: Die Programmierumgebung nach der Erstellung eines neuen Projekts

7. Der Großteil des Fensters besteht aus einem Texteditor, in dem Sie Ihren Programmcode schreiben werden. Der Text wurde schon von Visual Studio vorausgefüllt und zeigt eine leicht andere Variante des Hello Worlds, als ich sie Ihnen weiter oben vorgestellt habe. Markieren Sie den vorhandenen Text, und löschen Sie ihn vollständig.

Nun haben Sie ein neues, leeres Projekt vorbereitet. Übertragen Sie anschließend die Codezeilen des Hello Worlds in den Texteditor.

```
001 #include <iostream>
002
003 int main()
004 {
005   std::cout << "Hello World" << std::endl;
006   return 0;
007 }
```

Sie werden merken, dass Sie beim Programmieren eine Menge an Spezialsymbolen benötigen, die Sie vermutlich noch nie zuvor verwendet haben. Die geschweiften Klammern { und } benötigen zum Beispiel die Taste AltGr, rechts neben der Leertaste, genau wie die spitzen Klammern <<. Die Einrückung in Zeile 5 und 6 besteht übrigens aus zwei Leerzeichen hintereinander, dazu später mehr (Abschnitt 2.5.2). Nach dem Abtippen des Hello World sollte Ihr Editor so wie in Abbildung 2.5 aussehen.

Groß-/Kleinschreibung

In C++ und vielen anderen Programmiersprachen ist die Groß-/Kleinschreibung wichtig! Die Befehle werden nicht mehr erkannt, wenn Sie eine andere Schreibweise wählen.

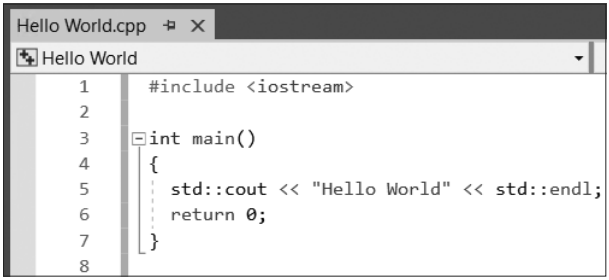

Abb. 2.5: Hello World in der IDE

Über der Texteingabe steht der Name der Datei, in der Sie gerade programmieren, hier `Hello World.cpp`. Sie können diese Datei auch auf Ihrer Festplatte finden, indem Sie entweder

- in Ihren Programmierprojekteordner gehen und dort den Unterordner `Hello World` öffnen oder
- mit der rechten Maustaste auf den gelben Tab klicken und Enthaltenden Ordner öffnen auswählen.

Sie könnten die .cpp-Datei übrigens in einem beliebigen Texteditor öffnen (Rechtsklick auf die Datei → Öffnen mit) und sogar auch damit verändern. Das Programmieren innerhalb von Visual Studio ist aber deutlich angenehmer.

Führen Sie nun das Programm aus. Dazu wählen Sie oben im Fenster Debuggen|Starten ohne Debuggen mit dem unausgefüllten grünen Dreieck (Abbildung 2.6) oder verwenden das Tastenkürzel Strg+F5. Das Kürzel lohnt sich, das wird Ihre häufigste Eingabe werden.

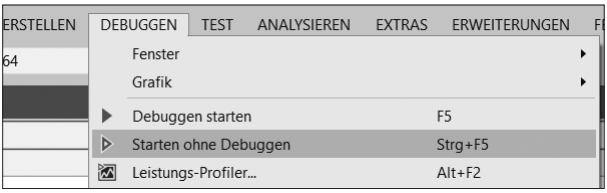

Abb. 2.6: Schaltfläche zum Starten des Programms

Es öffnet sich ein Konsolenfenster mit folgendem Inhalt:

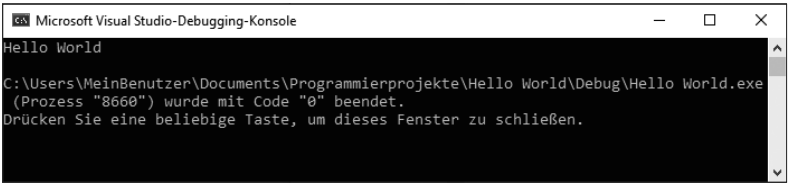

Abb. 2.7: Screenshot des Hello-World-Konsolenfensters

Im weiteren Verlauf dieses Buchs werden die Konsolenausgaben zur besseren Lesbarkeit auf das Wesentliche reduziert und in diesem Format dargestellt:

```
Hello World
```

Ihr erstes Programm wurde erfolgreich kompiliert! Das heißt, der menschenlesbare Programmcode wurde in maschinenlesbare Befehle übersetzt. Danach hat der Computer diese Befehle ausgeführt, und Sie haben das Ergebnis auf dem Bildschirm gesehen. Sie können das Konsolenfenster schließen und das Programm jederzeit erneut über Strg+F5 starten.

Um sich schneller ans Programmieren zu gewöhnen, empfehle ich Ihnen, den Code selbst abzutippen. Als Referenz finden Sie jedoch hier sämtliche Codebeispiele zum Download:
https://cpp.hasper.info/code_im_buch

2.2 Der Ausgangspunkt eines Programms: main()

Nun sehen wir uns den Code unseres Hello-World-Programms noch einmal im Detail an. Beginnen wir mit der dritten Zeile:

```
003 int main()
```

Das ist der *Einsprungspunkt*, an dem Ihr Programm beginnt, jedes Mal, wenn Sie es starten. Was genau passiert, wird mit einer geschweiften Klammer in Zeile 4 begonnen

```
004 {
```

und mit einer geschweiften Klammer in Zeile 7 beendet:

```
007 }
```

Ihr Programm beginnt also direkt nach der geschweiften Klammer und führt von dort Zeile für Zeile nacheinander aus. Diese Klammern definieren einen *Codeblock*, und die IDE erlaubt sogar, ihn ein- und auszuklappen, was besonders hilfreich ist, wenn der Code später komplizierter wird.

Abb. 2.8: Ein- und Ausklappen von Codeblöcken in der IDE mit dem Minus- bzw. Pluszeichen

Wie Sie bestimmt schon erraten haben, sorgt diese Zeile

```
005   std::cout << "Hello World" << std::endl;
```

dafür, dass auf der Konsole der Text `Hello World` ausgegeben wird. Die Anweisung wird durch zwei Leerzeichen am Beginn eingerückt.

`std` steht für die »Standard Library«, also die Standardbibliothek. Eine Bibliothek ist eine Ansammlung von hilfreichen Funktionalitäten, wie zum Beispiel in diesem Fall die Textausgabe. Mit `std::` machen Sie klar, dass Sie aus dieser Bibliothek eine Funktion nutzen möchten. Was genau Bibliotheken für Sie machen und welche weitere es gibt, folgt im Verlauf des Buchs.

`cout` steht für »Console Out« und bedeutet, dass nun Text auf der Konsole ausgegeben werden soll. `cout` ist ein sogenannter *Stream*, also ein Objekt, das nach und nach mit Daten gefüttert werden. In diesem Fall handelt es sich um Textdaten, die sofort ausgegeben werden sollen.

`<<` ist ein sogenannter *Operator*, das sind ein oder mehrere Zeichen mit besonderer Bedeutung. Hier stehen die zwei spitzen Klammern dafür, dass ein Text in `cout` eingegeben werden soll. Es ist kein Zufall, dass der Operator an Pfeile nach links erinnert: Das, was rechts von ihm steht, soll in das, was links von ihm steht, eingefügt werden. Später werden Sie noch anderen, Ihnen schon aus der Mathematik bekannten Operatoren begegnen, wie `+` oder `-`.

`"Hello World"` ist eine Zeichenkette. Damit so ein Text von Befehlen unterschieden werden kann, beginnt und endet er immer mit Anführungszeichen. In C++ ist es übrigens wichtig, dass Sie für Zeichenketten doppelte Anführungszeichen benutzen!

`std::endl` nimmt wieder etwas aus der Standard Library, in dem Fall `endl`, was für »End of Line« steht. Dieses Element fügt einen Zeilenumbruch ein, sodass die nächste Textausgabe in einer eigenen, neuen Zeile beginnt.

`;` Das Semikolon sieht unscheinbar aus, ist aber wichtig. Es beendet die aktuelle Anweisung und trennt sie von der nächsten. Löschen Sie es mal versuchsweise, und probieren Sie, das Programm zu kompilieren. Es wird nicht funktionieren.

Die nächste Zeile

```
006   return 0;
```

beendet das Programm mit dem Wert null. Das bedeutet, dass alles glatt gelaufen ist, sozusagen »null Fehler«. Alle anderen Zahlen bedeuten, dass das Programm mit einem Fehlerzustand beendet werden soll – und Sie als Programmierer dürfen selbst festlegen, welche Zahl für welchen Fehler steht. Zum Beispiel könnte eine Eins für eine volle Festplatte stehen, eine Zwei für fehlerhafte Benutzereingaben etc. In unserem kleinen Programm gibt es eigentlich noch keine Möglichkeit für einen Fehler, aber fügen Sie doch versuchsweise einmal statt einer Null eine Drei ein, starten das Programm (zur Erinnerung: Strg + F5 oder Abbildung 2.6) und sehen sich an, was passiert.

Vielleicht wird Ihnen auch mal eine etwas andere Version des »Hello World« begegnen, nämlich eine, die ohne die `return`-Anweisung auskommt. Das ist eine kleine Abkürzung und besagt, dass dieses Programm immer ohne Fehlernummer beendet werden soll.

Fällt Ihnen noch ein weiterer Unterschied zu unserem Hello World auf? Was es damit auf sich hat, sehen Sie in Abschnitt 4.1.

```
#include <iostream>
void main()
{
  std::cout << "Hello World" << std::endl;
}
```

Nun fehlt noch die allererste Zeile:

```
001 #include <iostream>
```

`#include` ist eine Anweisung an den Compiler, sich zusätzlichen Code aus einer anderen Datei zu holen. Das ist hilfreich, weil man damit in einer Datei auf Funktionalitäten zugreifen kann, die an anderer Stelle programmiert (man sagt auch: »implementiert«) wurden.

`<iostream>` ist eine Datei der Standard Library und beinhaltet den Code für `std::cout`. Ohne diese erste Zeile des Codes könnte der Compiler die Konsolenausgabe also nicht finden. Die spitzen Klammern bedeuten übrigens, dass diese Datei in einem Systemordner zu finden ist und nicht etwa in Ihrem eigenen Projektordner direkt neben der `Hello World.cpp`. Sie werden in einem der fortgeschrittenen Kapitel noch sehen, wie Sie auch Funktionalitäten aus selbstgeschriebenen Dateien einbinden können (Abschnitt 9.5).

Nun haben Sie das Grundgerüst eines jeden Programms kennengelernt. Im Folgenden erweitern wir unser kleines Projekt mit sogenannten *Variablen*.

2.3 Zwischenspeicher: Variablen und ihre Typen

Ein Computerprogramm besteht im Grunde aus zwei Dingen: *Anweisungen*, was das Programm tun soll, und *Daten*, mit denen diese Anweisungen arbeiten. Die einfachste Form, Daten im Programm zu speichern, sind sogenannte *Variablen*. Sie bestimmen einen Speicherplatz für eine Information und haben einen Typ, der festlegt, welche Art von Information in ihnen steckt, sowie einen Namen, um sie an beliebigen Stellen im Programm aufrufen zu können.

2.3.1 Zeichenketten

Das erste Beispiel für eine Variable sieht so aus:

```
std::string text;
```

String heißt auf Deutsch »Zeichenkette«, und Variablen mit diesem Typ können einen beliebigen Text beinhalten. An dem vorangestellten `std::` sehen Sie, dass dieser Variablentyp aus der Standard Library stammt. Auf den Variablentyp folgt der Name, mit dem die Variable im weiteren Verlauf aufgerufen werden kann.

Eine Variable wird also immer nach dem Schema *`Typ Name;`* definiert. Befüllt wird sie mit dem Gleichheitszeichen `=`, auch *Zuweisungsoperator* genannt. Hierfür gibt es zwei Möglichkeiten:

```
std::string text;
text = "Hello World";
```

oder die Kurzschreibweise

```
std::string text = "Hello World";
```

Wandeln Sie also unser erstes Programm dementsprechend ab:

```
001 #include <iostream>
002 #include <string>
003
004 int main()
005 {
006   std::string text = "Hello World";
007   std::cout << text << std::endl;
008   return 0;
009 }
```

Wie Sie sehen können, wird in Zeile 6 die Variable definiert, direkt mit Inhalt gefüllt und in Zeile 7 über ihren Namen aufgerufen und ausgegeben. Damit Sie den Typ `String` verwenden können, müssen Sie auch noch einen weiteren `include`-Befehl hinzufügen, siehe Zeile 2.

Führen Sie das Programm aus, es verhält sich genauso wie vorher, es ist nur etwas strukturierter.

Die Benennung von Variablen

Namen von Variablen sind frei wählbar, zum Beispiel:

```
std::string ofenkartoffel123;
```

allerdings sollten Sie sich immer die Mühe machen, einen sprechenden Namen zu verwenden, das heißt einen, der möglichst gut beschreibt, was in der Variable zu finden ist. Sonst wird das Programm schnell unverständlich. Daher eignet sich die »Ofenkartoffel123« eher nicht als Variablenname – außer vielleicht, wenn Sie ein Rezeptbuch programmieren und 123 verschiedene Kartoffelsorten verwalten. Abgesehen davon sind die technischen Voraussetzungen für einen Variablennamen wie folgt:

- Sie müssen mit einem Buchstaben oder einem Unterstrich _ beginnen.
- Sie dürfen keine Leerzeichen oder Spezialsymbole außer dem Unterstrich beinhalten.
- Ab dem zweiten Zeichen dürfen auch Zahlen verwendet werden.
- Variablennamen dürfen kein von C++ schon reserviertes Schlüsselwort sein (wie zum Beispiel `return`, weitere Schlüsselwörter lernen Sie im Verlaufe des Buchs kennen).

Strukturierter Code ist die eine Sache – aber Variablen können noch mehr! Was genau, hängt von ihrem Typ ab. Strings haben zum Beispiel die Möglichkeit, ihre Länge auszugeben. Ändern Sie die Zeile 7 wie folgt ab, und führen Sie das Programm aus:

```
007 std::cout << text << ". Zeichenanzahl: " << text.size();
```

Die Ausgabe sieht wie folgt aus:

```
Hello World. Zeichenanzahl: 11
```

Variablen haben also zusätzliche Funktionalitäten, die Sie über einen Punkt nach dem Variablennamen nutzen können – in diesem Falle die Länge von Strings, die mit `.size()` abgefragt werden kann.

Bis jetzt ist unser erstes Programm noch sehr vorhersehbar. Das ändert sich nun, denn jetzt nehmen wir Eingaben des Benutzers hinzu, speichern sie in einer Variable und verarbeiten sie. Hierfür empfiehlt es sich, nach der Anleitung in 2.1 ein neues Projekt zu erstellen und ihm beispielsweise den Namen

»Konsoleneingabe« zu geben. Die Datei, in der der neue Code landet, wird dementsprechend dann `Konsoleneingabe.cpp` heißen.

Ich werde in den folgenden Kapiteln die Codebeispiele jeweils mit einem neuen Dateinamen einleiten um anzuzeigen, dass Sie hierfür ein neues Projekt anlegen könnten. Es steht Ihnen natürlich frei, alle Beispiele im Hello-World-Projekt auszuprobieren und den Code immer wieder zu überschreiben, aber dann verlieren Sie natürlich die Möglichkeit, einmal absolvierte Beispiele später noch einmal abzurufen.

Für die Benutzereingabe von Wörtern verwendet man das Gegenstück zu `std::cout`, welches `std::cin` heißt und für »Console In« steht. Der Operator wird dafür umgedreht, aus zwei spitzen Klammern nach links `<<` werden zwei nach rechts `>>`.

Konsoleneingabe.cpp

```
001 #include <iostream>
002 #include <string>
003
004 int main()
005 {
006   std::cout << "Bitte geben Sie ein Wort ein: ";
007   std::string text;
008   std::cin >> text;
009   std::cout << "Sie haben ein Wort mit " << text.size() << " Buchstaben
    eingegeben: " << text << std::endl;
010   return 0;
011 }
```

Die Eingabe eines Wortes wird mit der Enter-Taste bestätigt. Die Ausgabe sieht zum Beispiel so aus:

```
Bitte geben Sie ein Wort ein: Programmieren
Sie haben ein Wort mit 13 Buchstaben eingegeben: Programmieren
```

Wenn Sie ein bisschen herumprobieren, werden Sie vielleicht feststellen, dass `std::cin` leider nur ein einziges Wort entgegennimmt und alles nach dem

ersten Leerzeichen verwirft. Wenn Sie einen Text mit mehreren Wörtern einlesen möchten, müssen Sie die Zeile 8

```
008 std::cin >> text;
```

mit dieser Zeile

```
008 std::getline(std::cin, text);
```

ersetzen. Probieren Sie es aus!

2.3.2 Zahlen

Es gibt noch weitere Standardtypen für Variablen. Der nächste Typ, den wir uns näher anschauen, ist für Ganzzahlen, im Englischen *Integer* genannt.

Integer.cpp

```
001 #include <iostream>
002
003 int main()
004 {
005   std::cout << "Bitte geben Sie eine Zahl ein: ";
006   int input;
007   std::cin >> input;
008   std::cout << "Sie haben die Zahl " << input << " eingegeben. Sie
    liegt zwischen den Zahlen " << input - 1 << " und " << input + 1 <<
    std::endl;
009   return 0;
010 }
```

Die Ausgabe sieht zum Beispiel so aus:

```
Bitte geben Sie eine Zahl ein: 17
Sie haben die Zahl 17 eingegeben. Sie liegt zwischen den Zahlen 16 und
18
```

Der Typ `int` kommt ohne das vorangestellte `std::` aus, denn dieser Typ entstammt nicht der Standardbibliothek, sondern ist ein Teil der Sprache C++.

Man nennt diese Typen auch *primitive Typen*[3]. Mit `- 1` und `+ 1` verringern oder erhöhen Sie den Wert für die Ausgabe. Wichtig hier ist, dass dadurch der Inhalt der Variable `input` nicht geändert, sondern rein für die Ausgabe etwas subtrahiert oder addiert wird.

Wenn Sie das Ergebnis in der Variable abspeichern wollen, können Sie das mit folgenden Befehlen erreichen:

- `input = input + 1;` erhöht den Wert um eins und speichert ihn danach in der Variable. Anders als in der Mathematik handelt es sich hier also nicht um eine unlösbare Gleichung, sondern um eine Zuweisung: »Addiere eine Eins auf den aktuellen Wert von `input` und speichere das Ergebnis wieder in `input`«.
- `input++;` ist die Kurzschreibweise des vorherigen Befehls. Die Ähnlichkeit mit dem Namen C++ ist übrigens beabsichtigt.
- `input += 3;` ist eine weitere Kurzschreibweise, diesmal für den Befehl `input = input + 3;`
- Dieselben Befehle gibt es auch entsprechend mit dem Minuszeichen: `input = input - 1; input--; input -= 3;`
- `input = input * 3;` multipliziert die Zahl mit drei. Die Kurzschreibweise hierfür ist `input *= 3;`
- `input = input / 3;` dividiert die Zahl durch drei. Die Kurzschreibweise hierfür ist `input /= 3;`

Wenn die Variable `input` am Anfang dieser Aufzählung den Wert 5 hatte, welchen Wert hat sie nach all den Berechnungen? Das Ergebnis finden Sie in der Lösung zu Übung 2.6.3. Die Sonderzeichen wie `+`, `++`, `-`, `--` usw. sind übrigens ebenfalls *Operatoren*, genau wie die schon bekannten `>>` oder `<<`.

Sie können auch zwei Ziffern verrechnen oder auch mehrere Variablen:

```
int a = 4 + 3;
int b = a + 3;
int c = a + b;
int d = a + b + c;
```

3 C++ besteht sozusagen aus zwei Teilen: zum einen der Sprache an sich mit ihren Regeln und Schreibweisen und zum anderen einer Handvoll primitiver Typen, mit denen jedes Programm grundsätzlich umsetzbar wäre. Damit aber nicht sämtliche Grundlagen wie zum Beispiel Zeichenketten jedes Mal selbst entwickelt werden müssen, kommt noch die Standardbibliothek hinzu – eine Ansammlung praktischer Hilfsmittel und komplexer Typen.

Beim Gleichheitszeichen gilt immer, dass der Ausdruck rechts davon ausgerechnet und dann in der Variablen links gespeichert wird. Sie können das beispielsweise nutzen, um einen simplen Taschenrechner zu programmieren, bei dem der Benutzer zwei Zahlen eingeben kann, die anschließend addiert werden:

Taschenrechner.cpp

```
001 #include <iostream>
002
003 int main()
004 {
005   std::cout << "Erste Zahl eingeben: ";
006   int a;
007   std::cin >> a;
008   std::cout << std::endl;
009   std::cout << "Zweite Zahl eingeben: ";
010   int b;
011   std::cin >> b;
012   std::cout << "Ihre Summe ergibt: " << a + b << std::endl;
013   return 0;
014 }
```

Experimentieren Sie hier auch mit den anderen Operatoren `-`, `*` und `/`. Es gelten die üblichen Regeln aus der Mathematik wie Punkt-vor-Strich, und Sie haben die Möglichkeit, mit runden Klammern die Berechnung zu gruppieren:

```
int a = (7 + 3) * 5;
```

Die Variable `a` enthält nun den Wert `50`. Allerdings wird Ihnen bei der Division vielleicht etwas Unerwartetes auffallen, zum Beispiel wird `5 / 3` den Wert `1` ergeben. Das liegt daran, dass `int` nur Ganzzahlen beinhalten kann und alles hinter dem Komma nicht rundet, sondern direkt abschneidet. Aus `1.6` wird daher `1`. Für Kommazahlen gibt es den Typen `double`. Beim Programmieren werden die Kommazahlen übrigens mit einem Punkt getrennt, nicht mit einem Komma – wieder ein Beispiel für die Englischprägung der Programmiersprachen.

```
double a = 5.0;
double b = 3.0;
std::cout << a / b;
```

2.3.3 Das Zusammenspiel von Variablen

Primitive Datentypen (Auszug)

- `int`: Für Ganzzahlen
- `double`: Für Kommazahlen
- `float`: Ebenfalls für Kommazahlen, allerdings mit weniger Präzision[4]
- `short`: Für sehr kleine Ganzzahlen
- `long long`: Für sehr große Ganzzahlen
- Sie können jeden Zahlentyp mit einem vorangestellten `unsigned` zu einem Typen machen, der nur positive Werte (und die `0`) annehmen kann, zum Beispiel: `unsigned int a = 4;` Wenn Sie diesem Wert zum Beispiel die `-1` zuweisen, wird Ihr Programm nicht kompilieren und Sie vor einem Fehler warnen.
- `bool`: Eine Variable, die wahr oder falsch sein kann.
  Mehr dazu in Kapitel 3
- `char`: Eine Variable für ein einzelnes Zeichen (englisch: character) in einem String. Mehr dazu ebenfalls in Kapitel 3

Den Plus-Operator können Sie nicht nur zum Verrechnen von Zahlen verwenden, sondern auch, um Zeichenketten aneinanderzuhängen:

StringPlus.cpp

```
001 #include <iostream>
002 #include <string>
003
004 int main()
005 {
006   std::string a = "Hello ";
007   std::string b = "World";
008   std::string c = a + b;
009   std::cout << c << std::endl;
010   return 0;
011 }
```

4 Kommazahlen vom Typ `float` nehmen weniger Speicher als Kommazahlen vom Typ `double` ein, dafür können sie aber nicht so viele Nachkommastellen abbilden. Nutzen Sie im Zweifel immer `double` für Kommazahlen, damit fahren Sie meistens richtig.

Man kann übrigens jede Variable – ob eine primitive wie `int` oder eine komplexere wie `std::string` – mehrfach befüllen, wodurch sich ihr Wert im Programmablauf ändert. Sehen Sie selbst:

Variablen.cpp

```
001 #include <iostream>
002 #include <string>
003
004 int main()
005 {
006   std::string text = "Hello World";
007   std::cout << text << " hat " << text.size() << " Zeichen." <<
    std::endl;
008   text = "Hallo Welt";
009   std::cout << text << " hat " << text.size() << " Zeichen." <<
    std::endl;
010   text = "";
011   std::cout << text << " hat " << text.size() << " Zeichen." <<
    std::endl;
012   return 0;
013 }
```

Ausgabe:

```
Hello World hat 11 Zeichen.
Hallo Welt hat 10 Zeichen.
 hat 0 Zeichen.
```

Es ist allerdings **nicht** möglich, eine Variable mehrmals zu definieren, sodass der folgende Code nicht kompilieren würde:

```
001 int main()
002 {
003   int variable;
004   int variable = 2;
005   return variable;
006 }
```

Der Compiler beschwert sich hier, dass der Variablenname `variable` schon vergeben ist. Das soll Fehler wie den folgenden vermeiden, wo nicht ganz klar wäre, welche der Variablen der Programmierer nutzen möchte:

```
001 #include <iostream>
002 #include <string>
003
004 int main()
005 {
006   int variable = 0;
007   std::string variable = "Null";
008   std::cout << variable << std::endl;    ??
009   return 0;
010 }
```

2.4 Vertippt? Fehler beim Kompilieren beheben

Falls Sie beim Abtippen des Codes im Buch einen Fehler gemacht haben, kann es sein, dass das Programm nicht kompiliert. Sie werden dann vermutlich mit diesem Dialog konfrontiert:

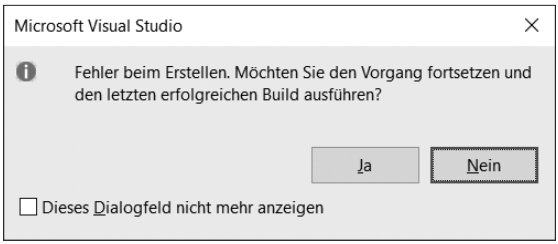

Abb. 2.9: Dialog der IDE bei Compilerfehlern

Sie werden gefragt, ob Sie den letzten erfolgreichen Build ausführen wollen, also den letzten Stand Ihres Programms, der noch fehlerfrei war. In diesem Dialog sollten Sie das Häkchen bei NICHT MEHR ANZEIGEN setzen und ihn dann **unbedingt mit Nein beantworten**. Ansonsten wird es Ihnen passieren, dass Ihr Programm in einem veralteten Stand gestartet wird und Sie einen Programmierfehler nicht sofort bemerken. Oder schlimmer – Sie wollen testen, ob das Programm nach einer Änderung immer noch das tut, was es soll, und da statt des neuen Codes noch der alte läuft, werden Sie fälschlicherweise in Sicherheit gewogen. Falls Ihnen ein gutes Argument einfällt, warum dieser Dialog jemals mit »Ja« beantwortet werden sollte, schreiben Sie mir gerne, mir ist keines bekannt.