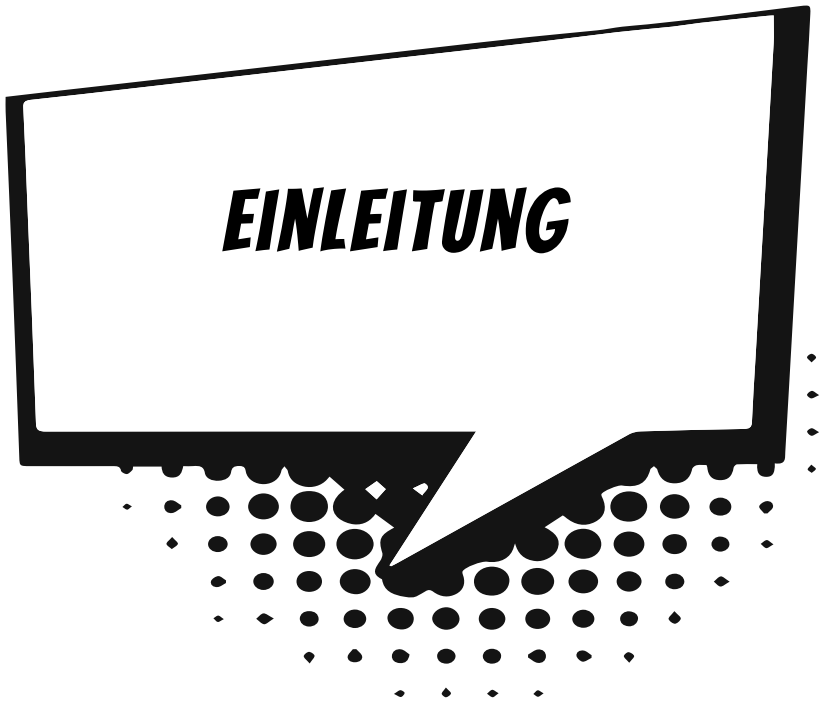
HANS-GEORG SCHUMANN

GANZ EINFACH PROGRAMMIEREN LERNEN UND EIGENE SPIELE ERSTELLEN

mitp

INHALT



EINLEITUNG

Um ein Spiel selbst zu erstellen, muss man vom Programmieren anfangs eigentlich noch gar nichts verstehen. Denn zuallererst braucht man eine Idee und dann einen Plan.

Wovon soll das Spiel handeln? Welche Geschichte soll es erzählen? Personen, Orte und Ereignisse, all das führt zu einem Plan, der umfasst, was zu diesem Spiel gehören soll. Und erst, wenn der Plan »steht«, kann die eigentliche Umsetzung in ein Programmprojekt beginnen. Dann allerdings sollte man schon möglichst gut programmieren können.

Und damit bist du bei diesem Buch genau richtig. Hier bekommst du über neun Kapitel solide und umfassende Grundlagen der Programmiersprache C++ vermittelt. Und damit du von Anfang an in die Spieleprogrammierung hineinschnuppern kannst, sind die Beispiele fast alle auf Spiele bezogen.

Danach geht es dann tatsächlich ans Programmieren eines richtigen Spiels, dabei lernst du, dein Grundwissen in C++ immer professioneller anzuwenden.

WELCHES WERKZEUG BENÖTIGEN WIR?

Zuerst einmal zu den Vorkenntnissen: Es sind keine nötig. Denn du beginnst ja sozusagen bei null. Allerdings solltest du dich mit Windows auskennen, denn zum Programmieren benötigen wir natürlich ein Betriebssystem, in dem unsere Programme laufen.

Was du brauchst, ist eine passende **Entwicklungsumgebung** und schon kanns losgehen. Damit ist ein System aus mehreren Komponenten gemeint:

Um ein Programm zu erstellen, musst du erst mal etwas eintippen. Das ist wie bei einem Brief oder einer Geschichte, die man schreibt. Das Textprogramm dafür kann sehr einfach sein, weil es ja nicht auf eine besondere Schrift oder Darstellung ankommt. So etwas wird **Editor** genannt.

Ist das Programm eingetippt, kann es der Computer nicht einfach lesen und ausführen. Jetzt muss es so übersetzt werden, dass er versteht, was du von ihm willst. Weil er aber eine ganz andere Sprache spricht als du, muss ein Dolmetscher her. Du programmierst in einer Sprache, die **du** verstehst, und der Dolmetscher übersetzt es so, dass es **dem Computer** verständlich wird. So etwas heißt dann **Compiler**.

Schließlich müssen Programme überarbeitet, verbessert, wieder getestet und weiterentwickelt werden. Dazu gibt es noch einige zusätzliche Hilfen wie den **Debugger**. Und aus alledem wird dann ein ganzes System, die **Entwicklungsumgebung**.

Mit der von Microsoft kostenlos zur Verfügung gestellten Software **Visual Studio** hast du eine sehr leistungsfähige Entwicklungsumgebung für die neuesten Windows-Versionen. Die benutzen wir hier.

Weil du nicht programmieren kannst, wie dir der Schnabel gewachsen ist, brauchen wir eine **Programmiersprache**. Die muss so aufgebaut sein, dass möglichst viele Menschen in möglichst vielen Ländern einheitlich damit umgehen können.

Weil in der ganzen Welt Leute zu finden sind, die wenigstens ein paar Brocken Englisch können, besteht auch fast jede Programmiersprache aus englischen Wörtern.

In diesem Buch hast du es mit der Programmiersprache **C++** zu tun. Sie ist seit vielen Jahren sehr weit verbreitet, allerdings nicht leicht zu erlernen. Dafür ist C++ die vielleicht mächtigste Sprache überhaupt. Viele Spiele und Spiele-Tools sind in C++ programmiert. Daneben gibt es noch einige andere Sprachen wie z. B. C, Java und C#, wobei C als die Mutter vieler Sprachen gilt, auch C++ ist von C abgeleitet, ebenso wie C# und Java.

UND WAS BIETET DIESES BUCH?

Vorwiegend geht es bei der Programmierung in C++ auch um Spiele. Du bekommst hier u. a. serviert:

- das Basiswissen von C++: Variablen, Ein- und Ausgabe, Kontrollstrukturen, Funktionen
- Profikenntnisse über Arrays, Zeiger, objektorientierte Programmierung
- Programmierung einer Player-Klasse als Basis für Spiele
- einen Einstieg in ein grafisches System mit dem Ziel, ein komplettes Spiel zu erstellen

Im **Anhang** gibt es dann noch zusätzliche Informationen, u. a. über Installationen und den Umgang mit Fehlern.

WIE ARBEITEST DU MIT DIESEM BUCH?

Um dir den Weg vom ersten Projekt bis zu einem fertigen Game einfacher zu machen, gibt es einige zusätzliche Symbole, die ich dir hier gern erklären möchte:

ARBEITSSCHRITTE

≫ Wenn du dieses Zeichen siehst, heißt das: Es gibt etwas zu tun. Damit kommen wir beim Entwickeln Schritt für Schritt einem neuen Ziel immer näher.

Grundsätzlich lernt man besser, wenn man Quelltexte selber eintippt oder ändert. Aber nicht immer hat man große Lust dazu. Weil es alle Projekte im Buch auch zum Download gibt, findest du am Ende des einen oder anderen Abschnitts auch den jeweiligen Dateinamen (z. B. → GAME). Wenn du also das Projekt nicht selbst erstellen willst, kannst du es stattdessen auch aus dem Internet laden – zu finden unter

```
www.mitp.de/0688
```

FRAGEN UND AUFGABEN

Am Ende eines Kapitels gibt es jeweils einige Fragen und Aufgaben. Diese Übungen sind nicht immer ganz einfach, aber sie helfen dir, dein Programmierwissen noch mehr zu vertiefen. Lösungen zu den Aufgaben findest du ebenfalls auf der **mitp**-Homepage. Du kannst sie dir alle im Editor von Windows oder auch in deinem Textverarbeitungsprogramm ansehen. Oder du lässt sie dir ausdrucken und hast sie dann schwarz auf weiß, um sie neben deinen Computer zu legen.

NOTFÄLLE

Vielleicht hast du irgendetwas falsch gemacht oder etwas vergessen. Oder es wird gerade knifflig. Dann fragst du dich, was du nun tun sollst. Bei diesem Symbol findest du eine Lösungsmöglichkeit. Auch ganz hinten im Anhang B findest du ein paar Hinweise zur Pannenhilfe.

WICHTIGE STELLEN IM BUCH

Hin und wieder findest du ein solch dickes Ausrufezeichen im Buch. Dann ist das eine Stelle, an der etwas besonders Wichtiges steht.

Wenn du ein solches »Wow« siehst, geht es um ausführlichere Informationen zu einem Thema.

WAS BRAUCHST DU FÜR DIESES BUCH?

Du findest Visual Studio als komplette Entwicklungsumgebung zum Download auf dieser Homepage:

```
https://www.visualstudio.com/de/downloads/
```

Visual Studio ist in der **Community**-Version kostenlos, sie reicht komplett aus, um mit C++ auch umfangreiche Programme zu erstellen. Nach dem Download wird alles mit dem **Setup**-Programm in ein Verzeichnis deiner Wahl installiert, z. B. c:\Programme\MICROSOFT Visual Studio.

Die Beispielprojekte in diesem Buch findest du ebenso wie die Lösungen zu den Aufgaben auf der Homepage des Verlags:

```
www.mitp.de/0688
```

BETRIEBSSYSTEM

Die meisten Computer arbeiten heute mit dem Betriebssystem Windows. Empfehlenswert ist eine der Versionen ab 10. C++ als universelle Sprache funktioniert auch unter anderen Betriebssystemen (wie z. B. Linux oder macOS), worauf ich hier aber nicht weiter eingehe.

SPEICHERMEDIEN

Auf jeden Fall benötigst du etwas wie einen USB-Stick oder eine SD-Card, auch wenn du deine Programme auf die Festplatte speichern willst. Auf einem externen Speicher sind deine Arbeiten auf jeden Fall zusätzlich sicher aufgehoben.

Gegebenenfalls bitte deine Eltern oder Lehrer um Hilfe.

HINWEISE FÜR LEHRER

Dieses Buch lässt sich selbstverständlich auch für den Informatik-Unterricht verwenden. Dort setzt natürlich jeder Lehrer seine eigenen Schwerpunkte. Aber wenn es z. B. um eine Programmier-AG oder einen Informatikkurs mit Schwerpunkt Spieleprogrammierung geht, lässt sich dieses Buch in Ergänzung zu Ihrem Lehrmaterial gut einsetzen. Und mit Visual Studio steht Ihnen und Ihren Schülern ein mächtiges Entwicklungswerkzeug zur Verfügung.

Was die Programmierung angeht, so wird hier mit **C++** keine unbedingt einfache Programmiersprache benutzt, aber vielleicht die mächtigste überhaupt. Zahlreiche Fragen und Aufgaben mit Lösungen helfen, Gelerntes zu festigen und zu vertiefen.

AUF DIE DATEIEN ZUM BUCH VERZICHTEN?

Vielleicht ist es Ihnen lieber, wenn Ihre Schüler die Projekte alle selbst erstellen. Dann lassen Sie die Download-Dateien einfach (erst einmal) weg.

ÜBUNGSMEDIEN

Für den Informatik-Unterricht sollte jeder Schüler ein eigenes externes Speichermedium haben, um darauf seine Versuche zu sichern. So wird verhindert, dass sich auf der Festplatte des Schulcomputers mit der Zeit allerlei »Datenmüll« ansammelt. Außerdem dient der eigene Datenträger dem Datenschutz: Nur der betreffende Schüler kann seine Daten manipulieren.

REGELMÄßIG SICHERN

Es kann nicht schaden, die Programmdateien, an denen gerade gearbeitet wird, etwa alle **zehn** Minuten zu speichern. Denn Computer pflegen gern gerade dann »abzustürzen«, wenn man seine Arbeit längere Zeit nicht gespeichert hat.

1 ERSTE SCHRITTE MIT C++

Am besten legen wir gleich los mit dem Programmieren. Nach dem Start des Computers und dem Auftauchen von Windows können wir uns schon dem ersten C++-Projekt widmen. Es wird natürlich noch kein Computerspiel sein, aber es gibt schon einiges zum Herumspielen.

In diesem Kapitel lernst du

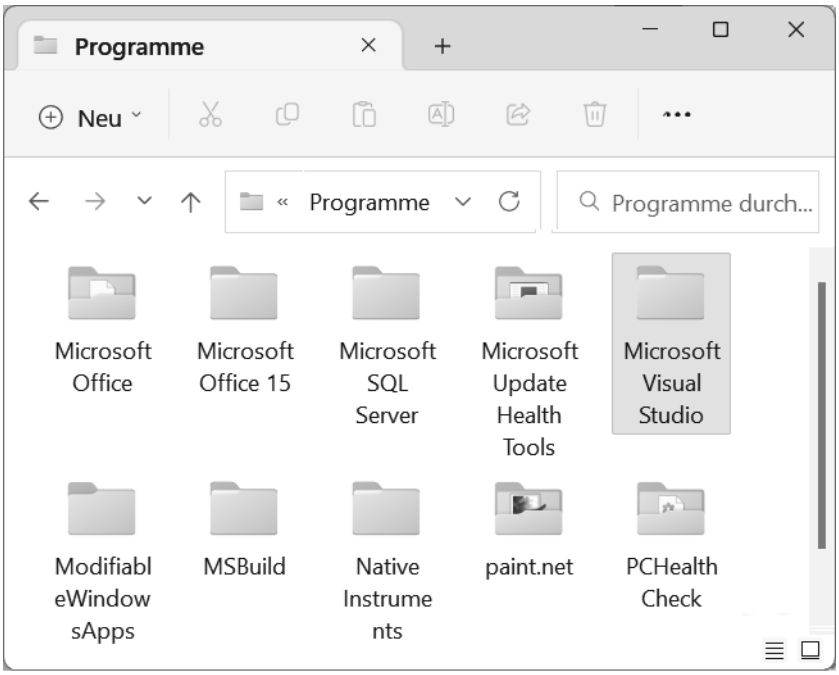
- wie man Visual Studio startet
- wie man ein Projekt erstellt und ausführt
- wozu `#include` gut ist
- was `cout` und `cin` bedeuten
- etwas über `int`, `char` und `string`
- wie man ein Projekt speichert
- wie man Visual Studio beendet

VISUAL STUDIO STARTEN

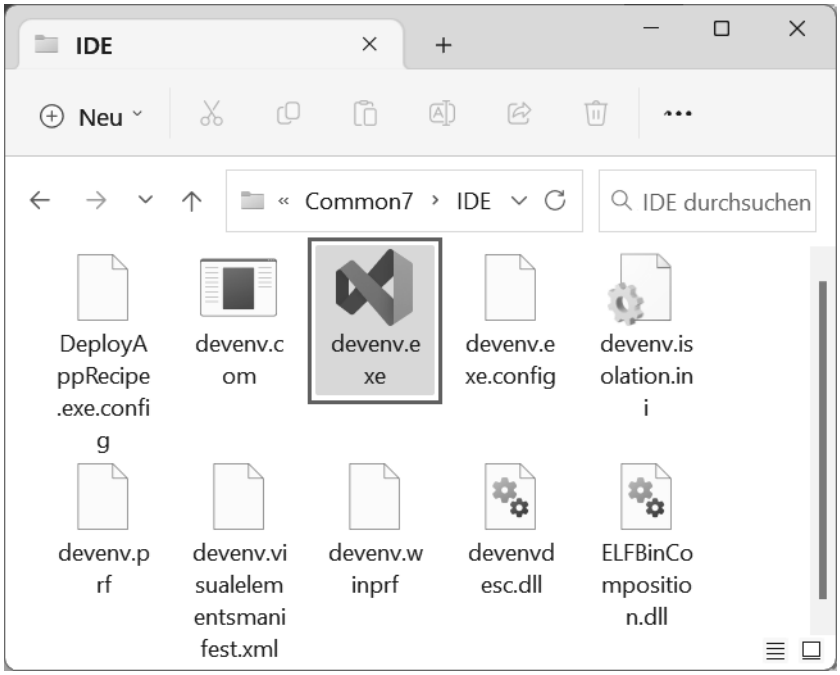
Bevor wir mit dem Programmieren anfangen können, muss **Visual Studio** erst installiert werden. Die Installation übernimmt ein Programm namens SETUP. Genaues erfährst du in **Anhang A**. Hier musst du dir von jemandem helfen lassen, wenn du dir die Installation nicht allein zutraust.

Eine Möglichkeit, Visual Studio zu starten, ist diese:

- Öffne den Ordner, in dem du Visual Studio untergebracht hast (z. B. C:\PROGRAMME\MICROSOFT VISUAL STUDIO).



- Dort musst du nun weiter in einige Unterordner mit den Namen COMMUNITY\COMMON7\IDE wechseln:
- Hier suchst du unter den zahlreichen Symbolen eines derjenigen heraus, bei denen etwas aussieht wie eine gekippte lila 8, und zwar das mit dem Namen DEVENV.EXE.



≫ Nun kannst du das Programm mit einem Doppelklick auf das Symbol starten:

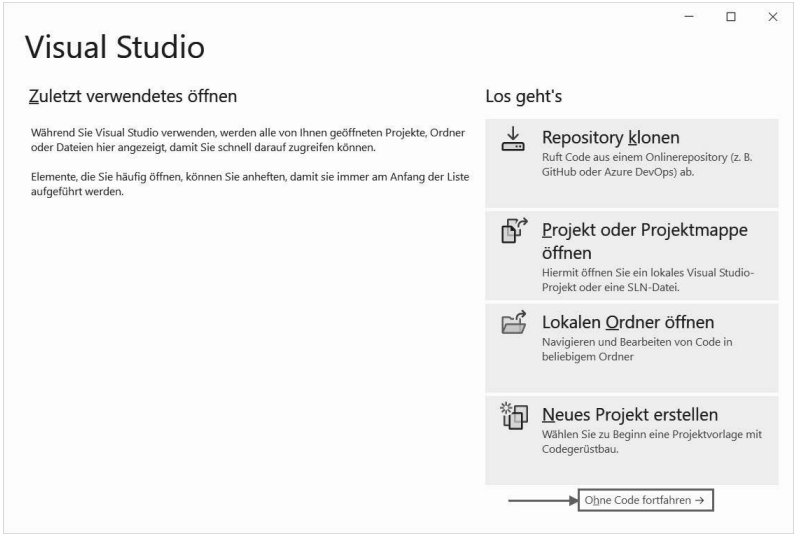
Ich empfehle dir, eine **Verknüpfung** auf dem Desktop anzulegen:

- Dazu klickst du mit der rechten Maustaste auf das Symbol für Visual Studio (DEVENV.EXE). Im Kontextmenü wählst du KOPIEREN.
- Dann klicke auf eine freie Stelle auf dem Desktop, ebenfalls mit der rechten Maustaste. Im Kontextmenü wählst du VERKNÜPFUNG EINFÜGEN.
- Es ist sinnvoll, für das neue Symbol auf dem Desktop den Text DEVENV.EXE – VERKNÜPFUNG durch VISUAL STUDIO zu ersetzen.

Von nun an kannst du auf das neue Symbol **doppelklicken** und damit Visual Studio starten.

KLEINE SPRITZTOUR DURCHS STUDIO

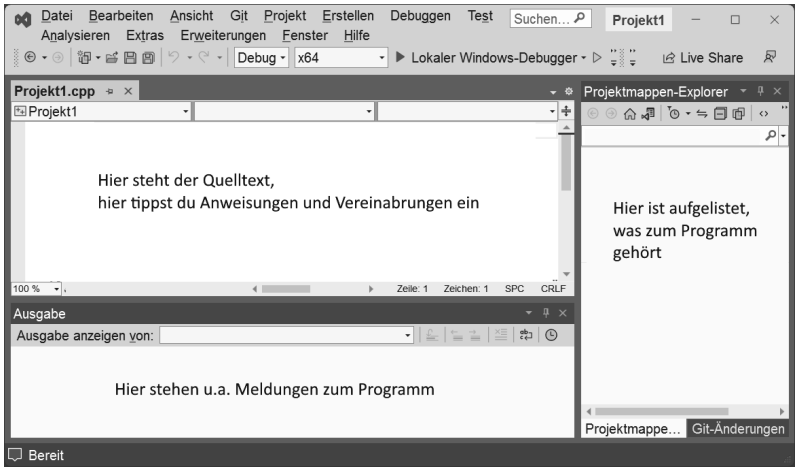
Je nach Computer kann es eine Weile dauern, bis Visual Studio geladen ist. Was dich schließlich erwartet, könnte ungefähr so aussehen:



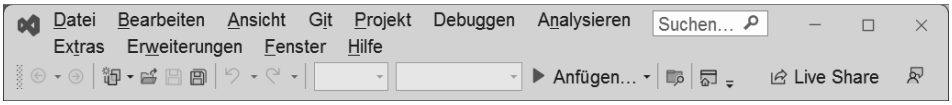
Wenn du hier auf NEUES PROJEKT ERSTELLEN klickst, beginnt dein erstes Projekt. Du kannst also gleich loslegen, wenn du willst. Oder:

≫ Du klickst erst einmal auf OHNE CODE FORTFAHREN, um im Hauptfenster von Visual Studio zu landen.

Dort sehen wir uns jetzt ein bisschen um. Über die Bedeutung der einzelnen Fenster (es gibt noch mehr davon) erfährst du nach und nach Genaueres. Hier nur ein kurzer Überblick:



Nun soll uns nur die Menüleiste interessieren – ganz oben:



Links darunter befinden sich jede Menge Symbole, die man mit der Maus anklicken kann. Zu einigen davon kommen wir im Laufe der folgenden Kapitel noch.

Diese Menüs von Visual Studio wirst du wahrscheinlich am meisten benutzen:

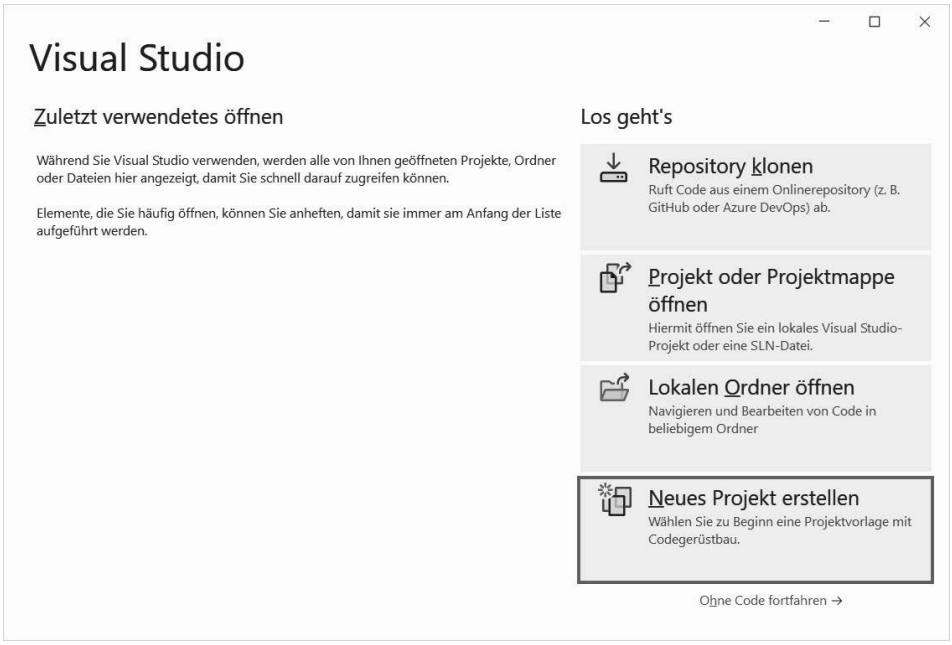
- Über das DATEI-Menü kannst du Dateien speichern, laden (öffnen), ausdrucken, neu erstellen oder Visual Studio beenden.
- Das BEARBEITEN-Menü hilft dir bei der Bearbeitung deines Programmtextes, aber auch bei anderen Programmelementen. Außerdem kannst du dort bestimmte Arbeitsschritte rückgängig machen oder wiederherstellen.
- Im ANSICHT-Menü hast du unter anderem die Möglichkeit, zusätzliche Hilfsfenster und Boxen ein- oder auszublenden.
- Über das DEBUGGEN-Menü sorgst du dafür, dass dein Programmprojekt ausgeführt wird.
- Und das HILFE-Menü bietet dir vielfältige Hilfe-Informationen an.

Einige wichtige Menüeinträge sind in einem sogenannten **Popup**-Menü zusammengefasst. Das heißt so, weil es dort aufklappt, wohin du gerade mit der **rechten** Maustaste klickst.

Ein Editorfenster, wie du es vielleicht von einem Editor oder Textverarbeitungsprogramm her kennst, ist gerade nicht in Sicht. Aber das lässt sich ändern: Ziel ist es ja, ein neues Projekt – unser Erstlingswerk – zu erstellen. Also los!

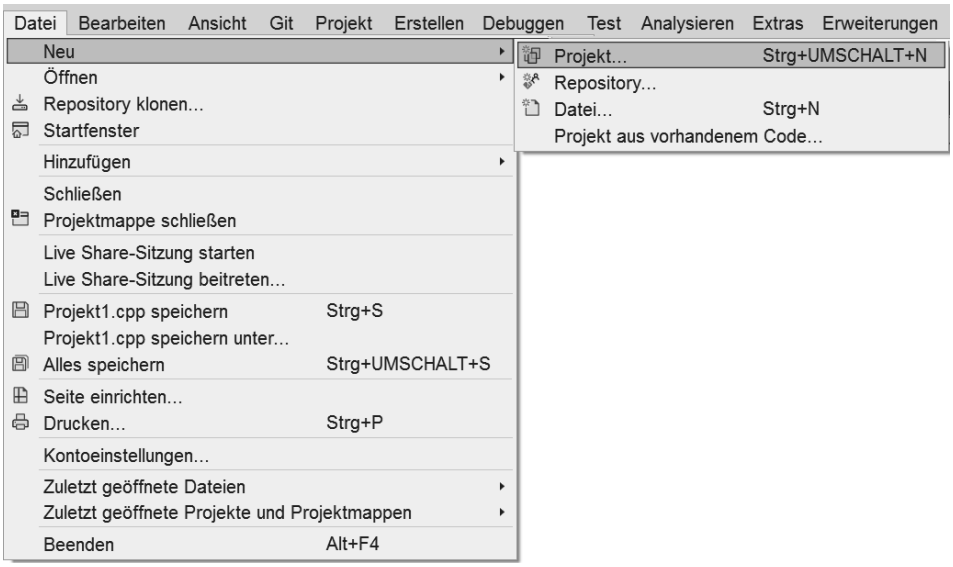
DAS ERSTE PROGRAMM

- Es gibt nun diese zwei Wege. Entweder du schließt das Fenster von Visual Studio (Klick auf das X rechts oben) und startest es neu – womit du in diesem Fenster landest:

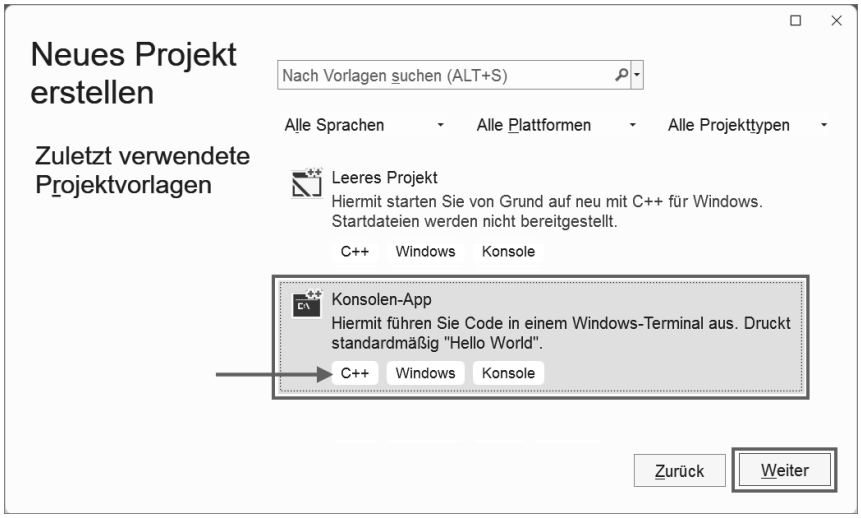


≫ Dann klickst du jetzt auf NEUES PROJEKT ERSTELLEN.

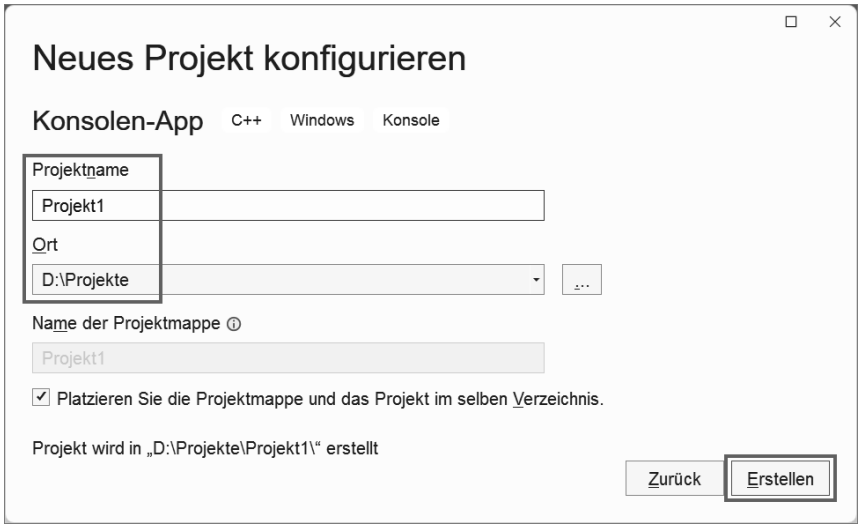
≫ Oder du hast dich entschieden, bei der Menüleiste von Visual Studio zu bleiben. Dann klickst du dort auf DATEI und im sich öffnenden Menü auf NEU und dann auf PROJEKT.



Es erscheint ein Dialogfeld, in dem es offenbar mehrere Möglichkeiten gibt, wie du dein Projekt erstellst:



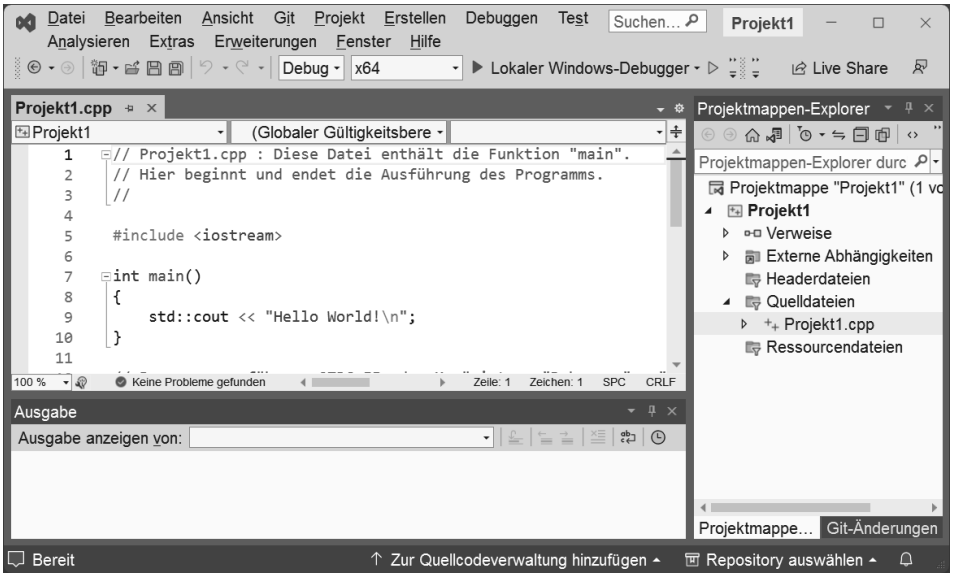
≫ Du musst dich erst ein wenig durch das Angebot nach unten blättern. Sorge dafür, dass links der Eintrag KONSOLEN APP für **C++** markiert ist. (Es gibt da mehrere Möglichkeiten, aber die anderen sind für andere Sprachen.)



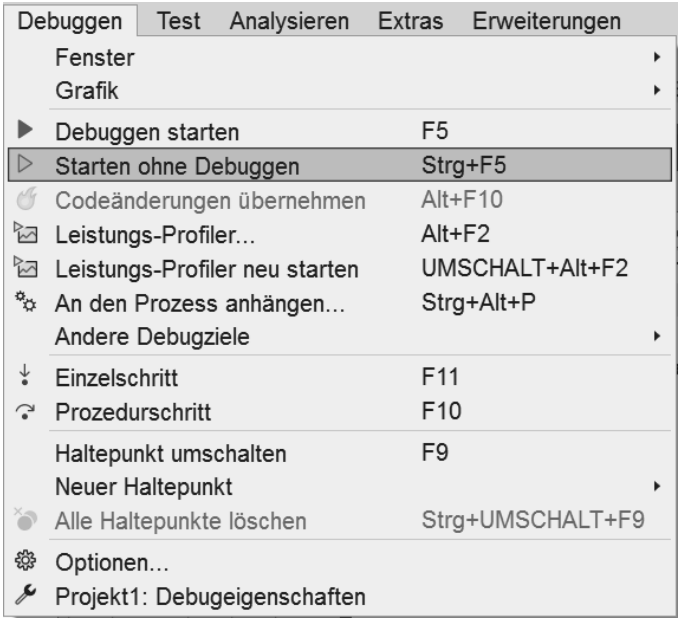
≫ Gib dem Projekt einen Namen und sorge dafür, dass hinter ORT der Pfad steht, wo du dein Projekt unterbringen willst. Dann klicke abschließend auf ERSTELLEN.

Ich habe das Ganze einfallslos erst mal PROJEKT1 genannt. Du kannst den von Visual Studio vorgegebenen Speicherort übernehmen, ich empfehle dir aber, besser einen eigenen Ordner zu erzeugen und den dort anzugeben. Bei mir ist das der Ordner PROJEKTE auf Laufwerk D:.

Und nicht lange darauf hast du dein erstes Mini-Projekt in C++. Visual Studio bietet also schon ein Gerüst-Programm, das du natürlich auch starten kannst, allerdings passiert (noch) nichts Aufregendes.

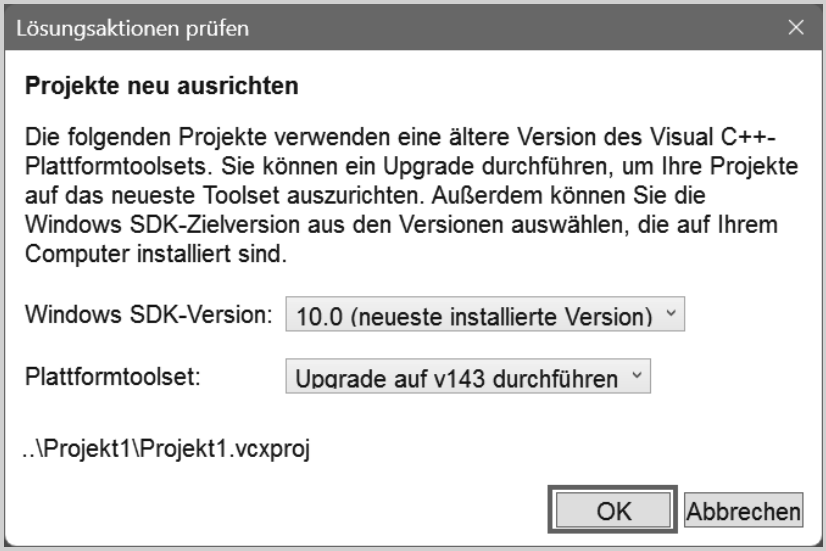


≫ Probier ruhig mal aus, was sich tut, wenn du in der Menüleiste auf DEBUGGEN und STARTEN OHNE DEBUGGING klickst. Oder du drückst die Tastenkombination Strg + F5.



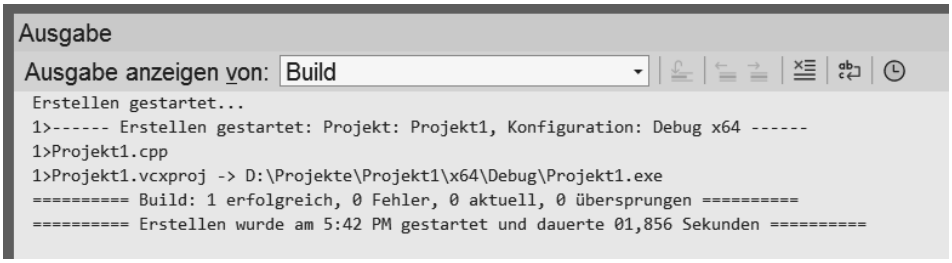
Alternativ funktioniert natürlich auch die Option DEBUGGEN STARTEN bzw. F5. Dann kann es ein bisschen länger dauern, bis das Programm startet.

Bei älteren Projekten kann beim ersten Start ein solches Meldefenster erscheinen:



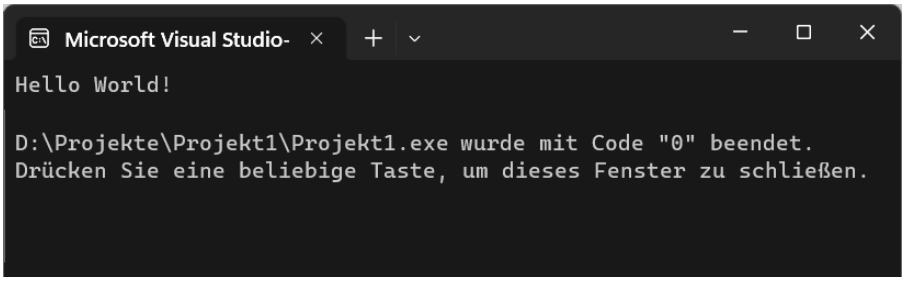
Klick einfach auf OK. Und das Projekt wird auf den neuesten Stand gebracht.

Vielleicht tut sich nach dem Start des Programms unten im Ausgabe-Fenster etwas, allerdings scheint das für uns nicht mehr als Kauderwelsch zu sein.



Aber da ist auch noch ein Fenster mit schwarzem Hintergrund, das sich auftut. Wenn du es nicht siehst, liegt es vielleicht **hinter** dem Fenster von Visual Studio. (Dann sorge dafür, dass dieses Fenster in den Vordergrund kommt.)

In diesem Fenster steht erst ein kurzer (englischer) Text. Weiter wichtig ist dann das, was ganz unten steht: die Aufforderung, eine beliebige Taste zu drücken.



≫ Folge dieser Aufforderung (oder Bitte), um das Fenster wieder zu schließen.

DER QUELLTEXT

Na ja, überwältigend war deine erste Begegnung mit einem C++-Programm nicht, aber es liegt an dir, daraus etwas zu machen. Zuerst aber sehen wir uns jetzt das näher an, was da links oben im Editorfenster steht. Dabei schäle ich mal das heraus, was für uns wichtig ist.

```
// Projekt1.cpp:
// Diese Datei enthält die Funktion "main".
// Hier beginnt und endet die Ausführung des Programms.
//
#include <iostream>

int main()
{
    std::cout << "Hello World!\n";
}
```

Den nachfolgenden Text lasse ich weg, er enthält nur Hinweise, die du auch hier im Buch erfährst.

Was du da siehst, wird **Quelltext** genannt. Die ersten Zeilen mit den beiden Schrägstrichen (//) am Anfang sind **Kommentare**. Da könnte man also auch so etwas hinschreiben:

```
// Mein erstes Projekt
```

Oder man lässt die beiden Kommentarzeilen einfach ganz weg. Das eigentliche Programm sieht in seiner einfachsten Form so aus:

```
int main()
{
}
```

Damit hat es einen Kopf und einen Rumpf: Der **Programmkopf** besteht aus der Zeile `int main()`, der **Programmrumpf** bisher nur aus zwei geschweiften Klammern. Man spricht hier auch von Hauptprogramm oder von Hauptfunktion. Daher der Name `main`.

Die Klammern (`{ }`) sind sehr wichtig. Sie bilden die Anfangs- und die Ende-Marke. Dazwischen stehen die Anweisungen, die dem Programm erst richtig zum Leben verhelfen. Und die stammen größtenteils von uns – noch nicht, aber wir sind ja erst am Anfang.

Zu **jeder** öffnenden Klammer muss es auch eine schließende Klammer geben! Wo genau du hier die Klammern hinsetzt, ist Geschmackssache. Der obige Programmtext könnte also auch so aussehen:

```
int main() {
}
```

Oder gar so:

```
int main() { }
```

Wo wir schon beim Thema Klammern sind: Was bedeuten eigentlich die runden Klammern hinter `main`? Das gehört zum Konzept von C++. Im Allgemeinen sind Anweisungen, die etwas ausführen sollen, Funktionen.

Und tatsächlich fasst C++ das ganze Programm als eine Funktion auf, nämlich der Hauptfunktion (englisch: »main function«). Du hast den Begriff schon weiter oben kennengelernt.

Den Begriff der **Funktion** solltest du aus dem Mathematik-Unterricht kennen. Du erinnerst dich nicht mehr? Da war doch zum Beispiel bei f(x) »f« der Name der Funktion und »x« der Name des Arguments, des Wertes also, der an die Funktion übergeben wird. Und etwas Vergleichbares gibt es auch in C++. Eine Funktion übernimmt in Klammern ein Argument, auch **Parameter** genannt.

In unserem Fall gibt es keinen Parameter, deshalb sind hier die Klammern hinter `main` leer. Aber auch so etwas wäre möglich:


```
int main(int x)
{
  return x;
}
```

Dann gäbe es einen Parameter namens `x`. Seinen Wert kann man innerhalb des Programms bearbeiten und dann über `return` zurückgeben.

Um eine Erläuterung von `int` drücke ich mich jetzt noch, doch ich komme bestimmt darauf zurück.

HALLO

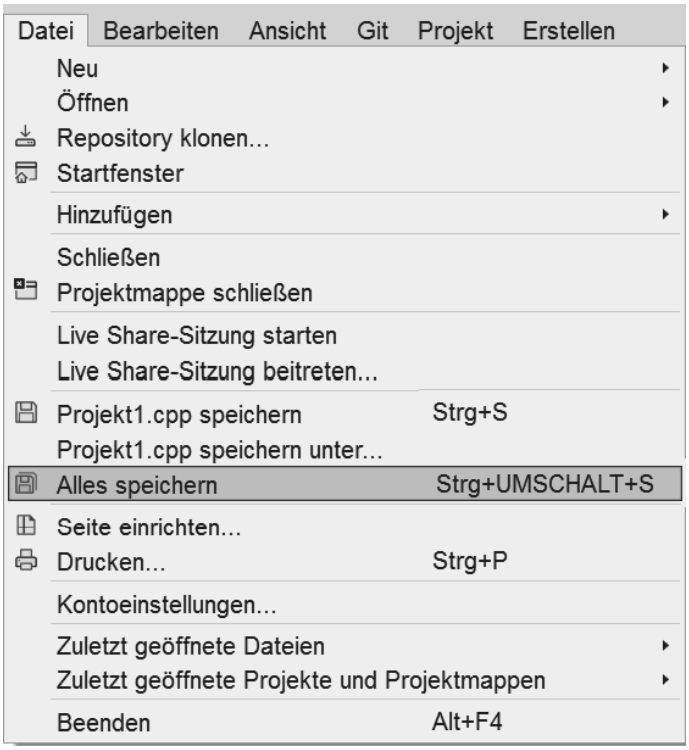
Und nun wird es Zeit für eine Programmerweiterung, damit auch wir etwas zu sehen bekommen.

≫ Lösche erst einmal den gesamten Text unterhalb der letzten geschweiften Klammer.

≫ Dann sieh dir den folgenden Quelltext an und pass ihn im Editor genauso an:

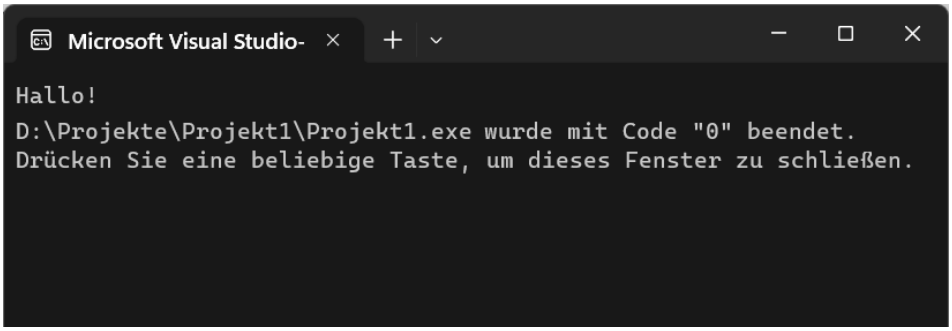
```
// Mein erstes Projekt
#include <iostream>

int main()
{
  std::cout << "Hallo!";
}
```

≫ Speichere dein Projekt über DATEI und ALLES SPEICHERN. Oder mit der Tastenkombination Strg + Shift + S.



≫ Dann starte das Programm mit Strg + F5 oder über DEBUGGEN und STARTEN OHNE DEBUGGING.

Und wieder öffnet sich ein schwarzes Fenster. Tatsächlich steht dort auch der Gruß »Hallo!« – und darunter wieder das schon bekannte Kauderwelsch, an das du dich aktuell gewöhnen musst.



Nun hat der Computer genau das getan, was im Quelltext als Anweisung stand:

```
std::cout << "Hallo!";
```

Was bedeutet: Gib über die Konsole den Text »Hallo!« aus. Das Objekt `cout` setzt sich aus »c« für »console« und »out« für »output« zusammen. Mit Konsole (oder Console) ist eben das schwarze Fenster gemeint, das gerade geöffnet ist.

Genauer gesagt besteht die **Konsole** aus einem Fenster auf dem Bildschirm (oder dem kompletten Bildschirm) und aus der Tastatur.

Dass das vorgesetzte `std` eine Abkürzung für »Standard« ist, kannst du dir vielleicht denken, ich komme später darauf zurück.

Nach Druck auf eine (beliebige) Taste wird das Programm beendet und das Konsolenfenster geschlossen.

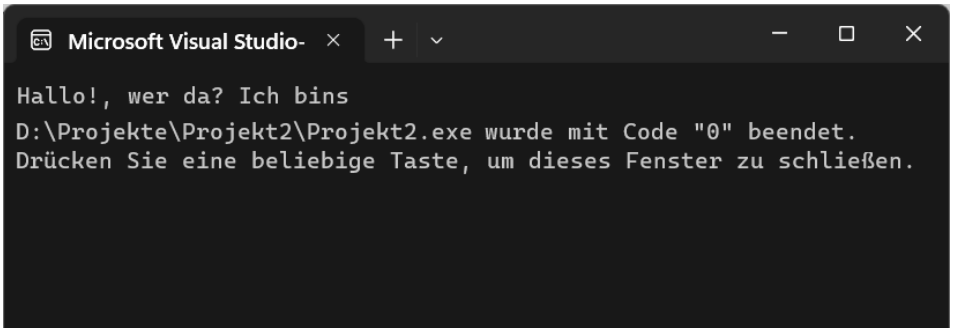
Ist dir aufgefallen, wie jede der beiden Zeilen innerhalb der geschweiften Klammern beendet wird? Dieses unscheinbar aussehende Semikolon (;) schließt **jede** Anweisung ab, darf also nicht vergessen werden!

COUT UND CIN

Bevor ich zu weiteren Erklärungen komme, möchte ich das Programm gleich noch etwas erweitern. Dann gibt es für dich auch schon etwas zu tun:

```
// Mein erstes Projekt
#include <iostream>

int main()
{
  int Name;
  std::cout << "Hallo, wer da? ";
  std::cin >> Name;
}
```

≫ Ändere deinen Quelltext entsprechend, speichere alles und starte dann das Programm.



≫ Was du nach der Frage »Hallo, wer da?« eintippst, ist egal. Vergiss nicht, das Ganze mit der Eingabe-Taste abzuschließen. (Und natürlich zum Schluss noch mal eine Taste zu drücken.)

Und nun bin ich dir einige Erklärungen schuldig. Beginnen wir mit dem Pärchen `cout` und `cin`, das ich in einer kleinen Tabelle zusammenfassen möchte:

`cout`	Zuständig für die Datenanzeige auf dem Display oder Monitor (= console output)
`cin`	Zuständig für die Datenübernahme von der Tastatur (= console input)

Beides sind C++-Elemente, die aber nicht zum Basis-Wortschatz von C++ gehören. Dafür brauchen wir ein Extra-Modul, das IOSTREAM heißt. Eingebunden wird die neue Datei mit

```
#include <iostream>
```

Dieses Modul gehört zu den Standard-Bibliotheken von C++. Deshalb verwenden wir hier spitze Klammern (<>).

iostream ist die Abkürzung für »input-output-stream«, was auf Deutsch so viel wie »Eingabe-Ausgabe-Strom« bedeutet. Somit sind cout und cin auch sogenannte Datenstrom-Objekte.

Mithilfe von cin und cout lassen sich Daten wie Zahlen und Zeichen übertragen, in diesem Fall über die Standard-Geräte Tastatur und Monitor. Womit wir bei dem Kürzel std wären, das für »standard« steht. Die beiden Doppelpunkte (::) verbinden dann das Ganze:

```
std::cout << "Hallo!";
std::cin  >> Name;
```

Wie du sehen kannst, lässt sich cin ähnlich wie cout auch mit einem Operator einsetzen:

»Strom«	Operator	Datenelement	Datentransfer
cout	<<	"Hallo"	Text → Ausgabe
cin	>>	Name	Eingabe → Variable

In der ersten Zeile wird der Text "Hallo" an den Ausgabestrom geschickt, der dann dafür sorgt, dass dieser Text sichtbar wird. In der zweiten Zeile wartet der Eingabestrom auf Zeichen von der Tastatur. Der Druck auf die Eingabe-Taste besagt, dass die Eingabe beendet ist.

Weil der "Hallo"-Text fest vorgegeben ist, weiß cout genau, was zu tun ist. Bei cin ist das anders, denn zunächst ist ja nicht bekannt, was derjenige, der das Programm benutzt, eingeben wird.

Deshalb benötigt cin eine **Variable**, in der die eingetippten Werte gespeichert werden können. Die habe ich hier Name genannt, man hätte sie aber auch x nennen können, oder wasweissich.

Variablen? Vielleicht kennst du den Begriff aus dem Mathe-Unterricht. Man kann dafür auch Platzhalter sagen. Die werden meist mit Buchstaben wie x oder y bezeichnet. Und weil solche Platzhalter in jeder Aufgabe einen anderen Wert annehmen können, nennt man so etwas Variablen (das Fremdwort »variabel« heißt ja auch so viel wie »veränderlich«).

Im Gegensatz dazu gibt es natürlich in C++ auch **Konstanten**. Die haben dann einen festgelegten Wert, der sich während des Programmlaufs **nicht** verändert. Und auch bei jedem neuen Programmstart behält eine Konstante ihren Wert.

Ein Beispiel ist der Text »Hallo, wer da?« Aber auch Zahlen wie z. B. 0, 1, -1, 3.14 lassen sich als Konstanten einsetzen (wie du noch sehen wirst).

Der Name einer Variablen darf übrigens nicht mit einer Ziffer beginnen. Ansonsten kannst du Buchstaben und Ziffern mischen, auch der Unterstrich (_) ist zulässig.

Zum Beispiel sind `Name1`, `A1b2C3`, `Mein_Name` erlaubte Namen für Variablen. Wichtig: C++ unterscheidet zwischen **Groß- und Kleinschreibung**, `Name` und `NAME` ist also nicht dasselbe!

Keinesfalls als Variablennamen benutzt werden dürfen **Schlüsselwörter**. Das sind Wörter, die nur für den Gebrauch in einer Programmiersprache reserviert sind. Dazu gehören z. B. `main` und `int`. Viele weitere wirst du noch im Laufe dieses Buches kennenlernen.

DATENTYPEN

Eine Variable muss dem Computer erst bekannt gemacht werden, bevor er sie benutzen kann. Mit unbekanntem Zeug schlägt sich der Computer nämlich gar nicht erst herum. Man nennt das auch **Vereinbarung**, womit wir zu dieser Zeile kommen:

```
int Name;
```

Was bedeutet denn nun das vorgestellte `int`? Jede Variable muss mit einem bestimmten **Typ** vereinbart werden. Damit der C++-Compiler aus dem Quelltext ein für den Computer ausführbares Programm machen kann, muss er wissen, womit er es zu tun hat: mit einer Zahl, einer Zeichenkette oder irgendetwas anderem. Denn mit Zahlen z. B. kann der Computer rechnen, mit Zeichen nicht. Das `int` ist eine Abkürzung für »Integer«, was hier ganze Zahl bedeutet.

Natürlich hast du recht, wenn du jetzt als aufmerksamer Leser sagst: Der Name ist doch keine Zahl. Eigentlich müssten wir die Variable `Name` also anders vereinbaren, oder? Es handelt sich bei einem Namen ja um einen Text bzw. eine **Zeichenfolge** oder **Zeichenkette**, und das muss der Compiler wissen.

Zeichen werden in C++ `char` genannt. Das ist eine Abkürzung von »character«. Vereinbaren wir also durch

```
char Name[20];
```

gleich eine Kette von Zeichen, sagen wir 20. (Wobei das letzte Zeichen intern reserviert ist. Solltest du längere Namen eingeben wollen, musst du diese Zahl unbedingt höher ansetzen.)

Damit sähe unser Programm-Listing jetzt so aus:

```
// Mein erstes Projekt
#include <iostream>

int main()
{
  char Name[20];
  std::cout << "Hallo, wer da? ";
  std::cin >> Name;
}
```

Wenn du dieses Programm startest, wirst du keinen Unterschied feststellen. Warum? Weil das Programm gleich nach der Eingabe beendet ist. Man bekommt gar nicht mit, was in der Variablen `Name` gespeichert wird. Dazu müsste man eine weitere Anweisung mit `cout` geben – was wir aber jetzt noch nicht tun werden.

Gibt es Probleme, die Tastenkombinationen für die passenden Klammern zu finden? Dann hilft dir sicher diese kleine Tabelle weiter:

Runde Klammern ()	Shift + 8	Shift + 9
Eckige Klammern []	AltGr + 8	AltGr + 9
Geschweifte Klammern {}	AltGr + 7	AltGr + 0

Du wirst künftig noch reichlich Gelegenheit haben, diese Tasten zu benutzen.

Ich möchte erst einmal eine kleine Änderung vornehmen. Außer dem Typ `char` gibt es in C++ auch noch den Typ `string`, was hier so viel wie »Zeichenkette« bedeutet. Dieser Datentyp ist deutlich komfortabler als `char`, u. a. weil `string` eine flexible Länge erlaubt. Dazu brauchen wir allerdings ein Extra-Modul, das wir gleich mit `#include` einbinden.

Und bei der Gelegenheit fügen wir natürlich noch eine `cout`-Zeile hinzu, in der der eingegebene Name auch angezeigt wird. Hier also die elegantere Variante unseres Programms:

```
// Mein erstes Projekt
#include <iostream>
#include <string>

int main()
{
  std::string Name;
  std::cout << "Hallo!, wer da? ";
  std::cin >> Name;
  std::cout << "Du bist also " << Name << "\n";
}
```

Wie angekündigt wird zuerst die neue Bibliothek für Strings eingebunden:

```
#include <string>
```

Dann kommt die Vereinbarung der Variablen `Name` als `string`:

```
std::string Name;
```

Und nach der Eingabe über `cin` soll dann der Name auch noch mal ausgegeben werden:

```
std::cout << "Du bist also " << Name << "\n";
```

Das `"\n"` am Ende ist ein **Steuerzeichen** und bedeutet, dass anschließend eine neue Zeile angefangen werden soll (= newline), damit der folgende Text nicht mehr hinten dranklebt. Wie du siehst, muss man hier den Operator << mehrmals benutzen.

Statt `"\n"` kann man auch `std::endl` benutzen, was »end of line« bedeutet. Auch das bewirkt, dass anschließend eine neue Zeile angefangen wird.

Bevor du diesen Quelltext eingibst, wollen wir ihn erst noch etwas schlanker machen. Mich stört dieses ständige `std`, aber es gibt einen Weg, um das Problem zu lösen. Lass einfach das folgende Listing auf dich wirken (→ HALLO1):

```
// Mein erstes Projekt
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string Name;
  cout << "Hallo!, wer da? ";
  cin >> Name;
  cout << "Du bist also " << Name << "\n";
}
```

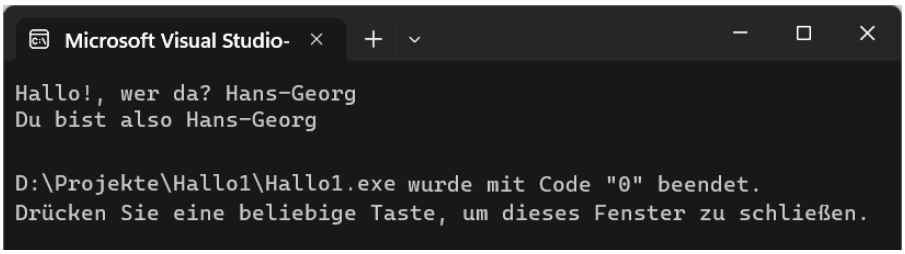
Sieht doch gleich viel aufgeräumter aus, oder? Dafür sorgt diese Zeile:

```
using namespace std;
```

Das heißt frei übersetzt so viel wie »Benutzen wir mal den Namensraum `std`«. Damit erlaubt der Compiler, dass alle Wörter, die zum Standard-Bereich bzw. zum Namensraum `std` gehören, nun ohne Zusatz benutzt werden dürfen.

Wie du in den Zeilen darunter siehst, wirkt der gesamte Quelltext jetzt deutlich aufgeräumter. Stell dir vor, man würde die Anzahl der `cout-cin`-Zeilen noch erhöhen, dann erspart man sich die ganze Zeit den Zusatz `std::`.

≫ Und nun pass deinen Quelltext an, speichere ihn und lass das Programm laufen.

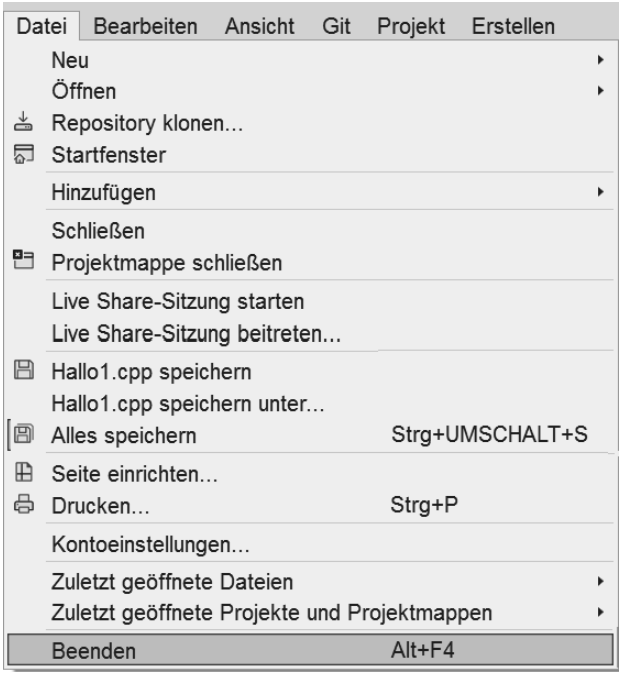


≫ Gib deinen Namen ein und prüfe, ob der Computer ihn sich gemerkt hat. Dann drück irgendeine Taste, um das Programm zu beenden.

VISUAL STUDIO BEENDEN

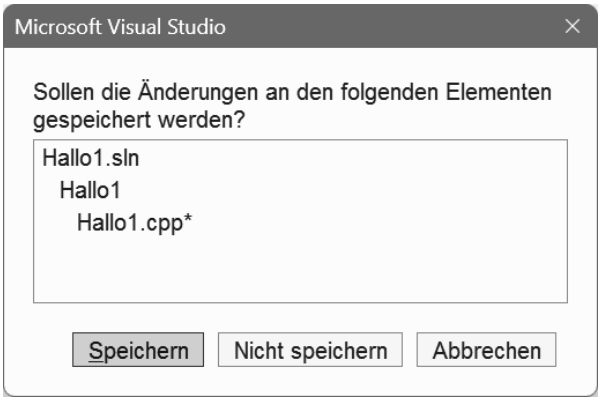
Dein allererstes Projekt liegt nun sicher auf einem Datenträger. Und damit wird es Zeit für eine kleine Pause.

≫ Um Visual Studio zu verlassen, klicke auf DATEI und dann auf BEENDEN.



≫ Oder du drückst die Tastenkombination Alt + F4. Du kannst auch im Hauptfenster ganz oben rechts auf das kleine X klicken. Irgendwie kommst du also immer »nach Hause«.

Sollte sich vor dem letzten Speichern noch etwas verändert haben, bekommst du dieses Meldefenster zu sehen:



Dort werden alle Dateien aufgelistet, an denen eine Änderung vorgenommen wurde. In unserem Fall ist das nur eine Datei.

≫ Klicke auf SPEICHERN, um deine letzte Version sicher unterzubringen.

Wie du siehst, hat die Datei die Kennung CPP, was für »C-plus-plus« steht.

ZUSAMMENFASSUNG

Mit deinem ersten Projekt gehörst du zwar noch nicht zur Zunft der C++-Programmierer, aber die Anfangsschritte hast du hinter dir. Mal sehen, was du von diesem Kapitel behalten hast. Da wären zuerst mal ein paar Optionen im Umgang mit Visual Studio:

Visual Studio starten	Doppelklick auf das Visual-Studio-Symbol
Ganzes Projekt speichern	Klick auf DATEI/ALLES SPEICHERN.
Programmprojekt starten	Klick auf DEBUGGEN/STARTEN OHNE DEBUGGING oder drück Strg + F5.
Alternative	Klick auf DEBUGGEN/DEBUGGEN STARTEN oder drück F5.

Programm in Konsole beenden	Drück eine beliebige Taste oder klick auf das **X** am Konsolenfenster.
Visual Studio beenden	Klick auf DATEI/BEENDEN.

Dein C++-Wortschatz ist zwar noch nicht allzu groß, aber einiges an Wörtern und Zeichen kennst du schon:

`#include`	Ein Modul einbinden
`std`	Namensraum für Standard-Module
`using namespace`	Namensraum festlegen
`char, char[]`	Einzelzeichen oder Zeichenkette
`int`	Ganze Zahl
`string`	Zeichenkette
`main()`	Hauptfunktion, Hauptprogramm
`return`	Wert zurückgeben
`cout`	Objekt für die Ausgabe, »Datenstrom«
`cin`	Objekt für die Eingabe, »Datenstrom«
`<<`	Operator für »Verschiebung« nach links (hier Ausgabestrom)
`>>`	Operator für »Verschiebung« nach rechts (hier Eingabestrom)
`{ }`	Klammern für Anfang-Ende-Marken
`( )`	Klammern für Parameter
`< >`	Klammern für Standard-Modul
`//`	Kommentarzeichen
`;`	Abschluss einer Anweisung

Und du hast auch schon ein paar Module eingebunden, die C++ zusätzlich zum Programm benötigt:

`iostream`	Modul für Eingabe/Ausgabe-Operationen
`string`	Modul für den Einsatz von Zeichenketten

EIN PAAR FRAGEN ...

1. Wie setzt man in einem C++-Programm Marken für Anfang und Ende?
2. Was genau ist der Unterschied zwischen `cin` und `cout`?
3. Wie bindet man zusätzliche Module in ein Programm ein?
4. Was passiert, wenn man als Typ für die Variable `Name` statt `string` wieder `int` benutzt?

... UND EINE AUFGABE

1. Setze das »Gespräch« mit `cout` und `cin` fort. Wenn du willst, kannst du die Variable `Name` weiter benutzen, ihr Inhalt wird allerdings mit jeder neuen Eingabe überschrieben. Wenn du das nicht willst, vereinbare neue Variablen.

STICHWORTVERZEICHNIS

E

F

G

H

I