HANS-GEORG SCHUMANN

GRUNDLAGEN FÜR SPIELEPROGRAMMIERER

mitp

Hinweis des Verlages zum Urheberrecht und Digitalen Rechtemanagement (DRM)

Der Verlag räumt Ihnen mit dem Kauf des ebooks das Recht ein, die Inhalte im Rahmen des geltenden Urheberrechts zu nutzen. Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und Einspeicherung und Verarbeitung in elektronischen Systemen.

Der Verlag schützt seine ebooks vor Missbrauch des Urheberrechts durch ein digitales Rechtemanagement. Bei Kauf im Webshop des Verlages werden die ebooks mit einem nicht sichtbaren digitalen Wasserzeichen individuell pro Nutzer signiert.

Bei Kauf in anderen ebook-Webshops erfolgt die Signatur durch die Shopbetreiber. Angaben zu diesem DRM finden Sie auf den Seiten der jeweiligen Anbieter.

FÜR
JANNE, JULIA, DANIEL UND KATRIN

Hans-Georg Schumann

C++ FÜR KIDS

GRUNDLAGEN FÜR SPIELEPROGRAMMIERER

mitp

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

ISBN 978-3-95845-713-3
1. Auflage 2018

www.mitp.de
E-Mail: mitp-verlag@sigloch.de
Telefon: +49 7953 / 7189 - 079
Telefax: +49 7953 / 7189 - 082

© 2018 mitp-Verlags GmbH & Co. KG, Frechen
Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.
Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Lektorat: Katja Völpel
Sprachkorrektorat: Petra Heubach-Erdmann
Covergestaltung: Sandrina Dralle, Christian Kalkert
Coverbild: © yayasya @ fotolia.com
Satz: III-satz, Husby, www.drei-satz.de

INHALT

EINLEITUNG 11

Welches Werkzeug benötigen wir? 12
Und was bietet dieses Buch? 13
Wie arbeite ich mit diesem Buch? 13
Was brauchst du für dieses Buch? 14
Hinweise für Lehrer 15

ERSTE SCHRITTE MIT C++ 17 **1**

Visual Studio starten 18
Kleine Spritztour durchs Studio 19
Das erste Programm 20
Der Quelltext 25
Der Aufbau eines Programms 27
Hallo 28
cout und cin 30
Datentypen 33
Visual Studio beenden 36
Zusammenfassung 37
Ein paar Fragen 38
... und eine Aufgabe 38

TYPEN UND OPERATOREN 39 **2**

Variablen und Werte 40
Typenvielfalt 43
ä-ö-ü-ß 45
Rechenspiele 47
Operationen 50
Ausgabe mit Format 53
Mathe mit Strings? 55
Konstanten 56

Zusammenfassung 58
Ein paar Fragen 58
... und zwei Aufgaben 59

3 KONTROLLE UND AUSWAHL 61
Ein Projekt öffnen 62
Die if-Struktur 65
if und else 68
Vergleichsoperatoren 71
Verknüpfungen 73
Von Fall zu Fall 75
Zusammenfassung 79
Ein paar Fragen 80
... und ein paar Aufgaben 80

4 WIEDERHOLUNGEN 81
Zufallszahlen 82
Es darf geraten werden 84
while oder do-while? 87
Wie oft? 88
Zählschleifen 90
break oder continue? 92
Verschachtelungen 95
Zusammenfassung 98
Ein paar Fragen 99
... und ein paar Aufgaben 99

5 FUNKTIONEN 101
C++ ist lernfähig 102
Init, Play, Evaluate 105
Lokal oder global? 106
Parameter 109
Wertverlust? 111
bool und return 112
Wert oder Referenz 116
Prototypen 118
Zusammenfassung 122
Ein paar Fragen 122
... und ein paar Aufgaben 122

ARRAYS, STRUKTUREN, ZEIGER ..... 123 — 6
Variablenfelder 124
Dimensionen 128
Die Sache mit struct 132
Adressen 136
... und Zeiger 140
Zusammenfassung 145
Ein paar Fragen 145
... und zwei Aufgaben 145

KLASSEN UND MODULE ..... 147 — 7
Spieler-Struktur 148
Alles unter einem Hut? 150
Es geht nicht ohne public 153
Keine Klasse ohne Konstruktor 156
Privatsphäre 159
Neue Dateien 161
Projekt-Module 166
Zusammenfassung 169
Keine Fragen 169
... aber ein paar Aufgaben 169

VERERBUNG UND POLYMORPHIE ..... 171 — 8
Erbschaften 172
Noch mehr Konstruktoren? 175
Überladen von Funktionen 178
Überschreiben von Funktionen 180
Polymorphie 183
Zeiger auf Objekte 186
Destruktionen? 189
Objekt-Felder 191
Zusammenfassung 193
Ein paar Fragen 194
... und zwei Aufgaben 194

CONTAINER UND DATENSTRÖME ..... 195 — 9
Dynamische Arrays 196
Suchen und Finden 199

Verkettete Listen ... 201
Die Sache mit dem Iterator ... 204
Datenverkehr ... 207
Flexible Mengen ... 210
Zusammenfassung ... 213
Ein paar Fragen 214
... und zwei Aufgaben ... 214

10 BUNTES ALLERLEI ... 215
Standardparameter ... 216
Statische Elemente ... 218
String-Schnipsel ... 222
Die Sache mit char ... 225
Operatoren überladen ... 227
Funktions-Templates ... 230
try und catch ... 232
Zusammenfassung ... 236
Ein paar Fragen 236
... aber keine Aufgabe ... 236

11 KLEINER KRABBELKURS ... 237
SDL oder SFML? ... 238
Dynamisch und statisch ... 242
Ein Grafikfenster ... 245
Insekt als Player ... 249
Tastensteuerung ... 253
Grenzkontrollen ... 257
Zusammenfassung ... 260
Ein paar Fragen 261
... und eine Aufgabe ... 261

12 WANZENJAGD ... 263
Die Sache mit der Maus ... 264
Ohne Mathe geht es nicht ... 266
Freilauf ... 270
Klick und Platt ... 274
Zeit und Punkte ... 279

Zusammenfassung 286
Ein paar Fragen 286
... und ein paar Aufgaben 287

GAME UND PLAYER ..... 289 — 13
Alles auf neu 290
Die Game-Klasse 292
Die Player-Klasse 296
Play the Game 300
Wanzen-Sammlung 304
Zusammenfassung 308
Ein paar Fragen 308
... und eine Aufgabe 308

DODGE ODER HIT ..... 309 — 14
Ein neuer Player 310
Standing, Ducking, Jumping 313
Die Ball-Klasse 317
Kollisionskontrollen 321
Das Hauptprogramm 325
Zusammenfassung 328
Keine Fragen 328
... aber ein Paar Aufgaben 328

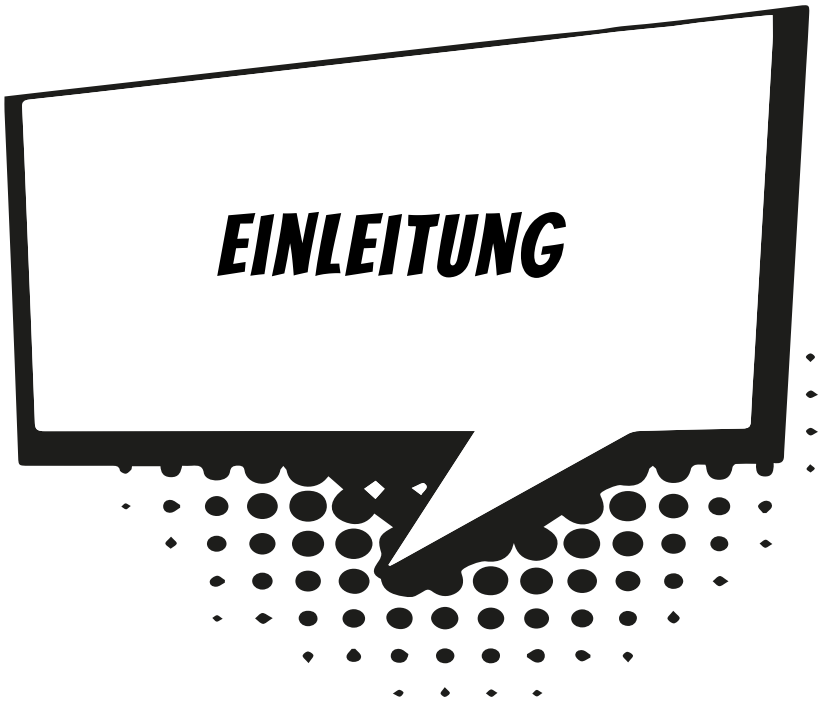
JUMP & RUN ..... 329 — 15
Figuren-Auflauf 330
Ein neuer Player 333
Testlauf 336
Animation 339
Szenenwechsel 343
Himmel und Boden 346
Farbgebung 348
Laufen und Springen 350
Zusammenfassung 352
Keine Fragen 352
... und nur eine Aufgabe 352

16 GIZMO ODER WAS? 353
Die Gizmo-Klasse 354
Bunte Scheiben 357
Auf dem Weg zum Regenbogen 360
Hindernislauf 364
Allerlei Dinge 367
Sprung-Technik 369
Leben und Tod 374
Zusammenfassung und Schluss 377
Keine Fragen 377
... aber eine Aufgabe 377

A ANHANG A 379
Visual Studio installieren 379
SFML herunterladen und vorbereiten 385
Einsatz der Buch-Dateien 388

B ANHANG B 389
Kleine Checkliste 389
Dem Fehler auf der Spur 390

STICHWORTVERZEICHNIS 393



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 im ersten Teil über zehn 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.

Im zweiten Teil geht es dann tatsächlich ans Programmieren von drei richtigen Spielen, 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 brauchen wir natürlich ein Betriebssystem, in dem unsere Programme laufen.

Was du brauchst, ist eine passende **Entwicklungsumgebung** und schon kann's 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. Daraus 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.

UND WAS BIETET DIESES BUCH?

Vorwiegend geht es um die Programmiersprache C++ und natürlich um Spiele. Du bekommst hier u.a. serviert:

- das Basiswissen von C++: Variablen, Ein- und Ausgabe, Kontrollstrukturen, Funktionen
- Profikenntnisse über Arrays, Zeiger, Grafik, objektorientierte Programmierung
- wie man eigene `Game`- und `Player`-Klassen programmiert
- Profikenntnisse über Sprites, Bewegungssteuerung, Kollisionen

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

WIE ARBEITE ICH 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. → *MyGame*). Wenn du also das Projekt nicht selbst erstellen willst, kannst du es stattdessen auch aus dem Internet laden – zu finden unter

www.mitp.de/712

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 anschauen. 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.

HINTERGRUNDWISSEN

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.

Für die Programmierung der Spiele benötigst du noch einige Zusatzmodule, die dann in Visual Studio eingebunden werden. Die findest du unter:

https://www.sfml-dev.org/download.php

SFML ist die Abkürzung für »Simple and Fast Multimedia Library« und ist ein Paket, das zahlreiche Klassen und Funktionen speziell für Grafik und Spiele enthält.

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

http://www.mitp.de/712

Alle Projekt-Ordner mit Spielen enthalten **nur** die Beispiel-Programme, **nicht** aber die SFML-Dateien – aus Copyright-Gründen. Du musst sie also selbst in den jeweiligen Projekt-Ordner kopieren.

BETRIEBSSYSTEM

Die meisten Computer arbeiten heute mit dem Betriebssystem Windows. Davon brauchst du eine der Versionen ab 7. C++ als universelle Sprache funktioniert auch unter anderen Betriebssystemen (wie z.B. Linux oder Mac OS), worauf wir hier aber nicht weiter eingehen.

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.

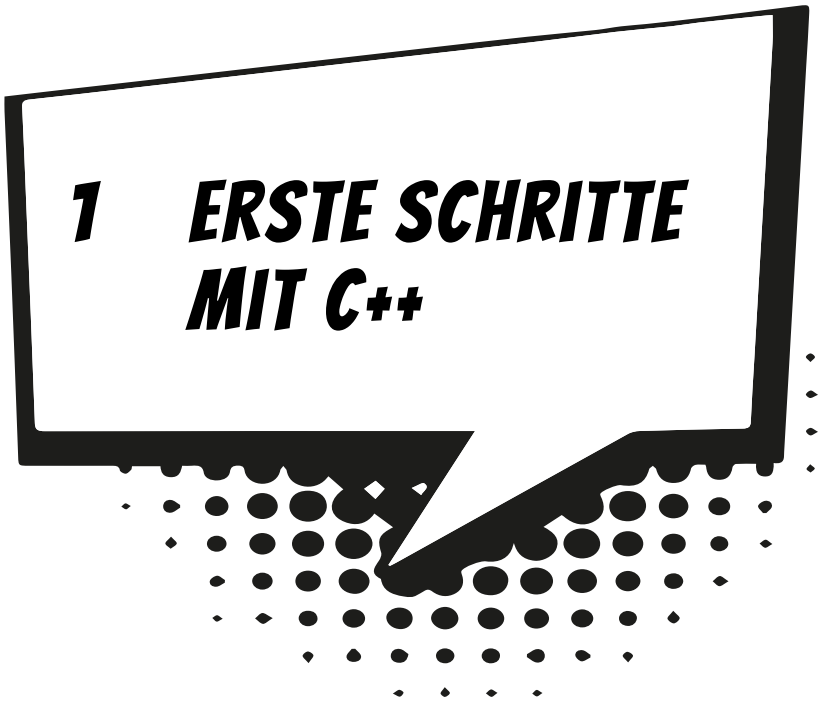
ÜBUNGSMEDIEN

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

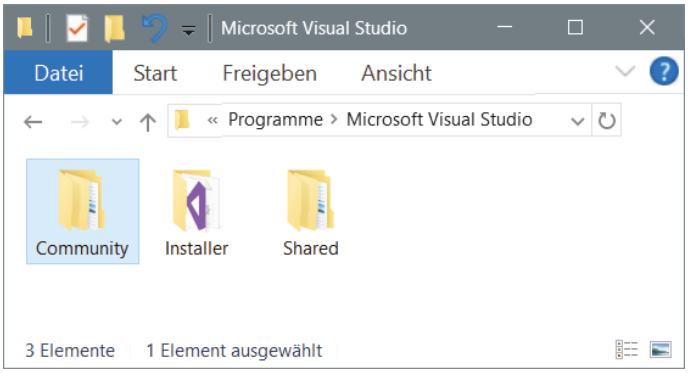
- wie man Visual Studio startet
- wie man ein Projekt erstellt und ausführt
- wozu `#include` gut ist
- was `cout` und `cin` bedeuten
- etwas über `int`, `char` und `string`
- wie man ein Projekt speichert
- wie man Visual Studio beendet

VISUAL STUDIO STARTEN

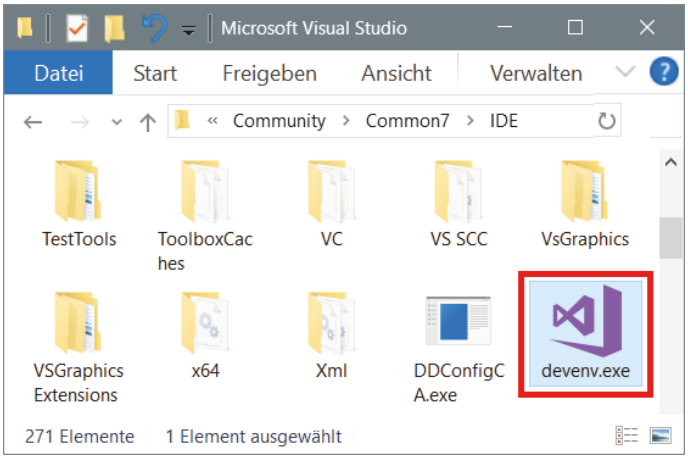
Bevor wir mit dem Programmieren anfangen können, muss **Visual Studio** erst installiert werden. Die Installation übernimmt ein Programm namens *Setup*. Genaues erfährst du im **Anhang A**. Hier musst du dir von jemandem helfen lassen, wenn du dir die Installation nicht allein zutraust.

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*).

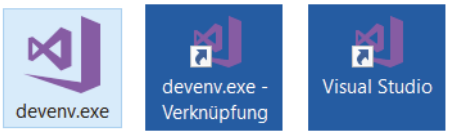


≫ Dort musst du nun weiter in einige Unterordner mit den Namen *Community \Common7\IDE* wechseln:



≫ 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:



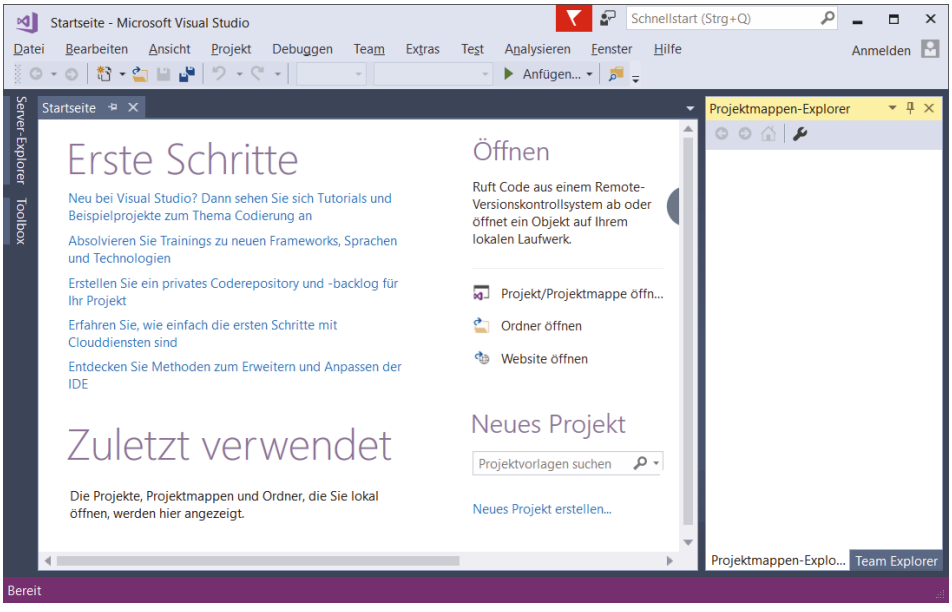
Ich empfehle dir, eine **Verknüpfung** auf dem Desktop anzulegen:

- 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.

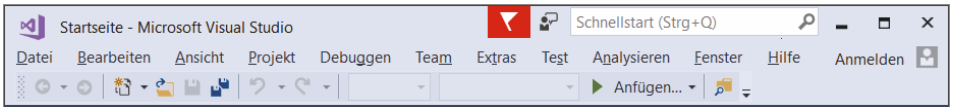
Von nun an kannst du auf das neue Symbol **doppelklicken** und damit Visual Studio starten.

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 – wobei das aktuelle Bild unter anderem davon abhängt, ob du eine Verbindung zum Internet hast oder nicht:



Für den ersten Augenblick ist das vielleicht ein bisschen verwirrend. Da tummelt sich ja einiges auf dem Bildschirm.



Ganz oben kann man die Menüleiste erkennen. Links darunter befinden sich jede Menge Symbole, die man mit der Maus anklicken kann.

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

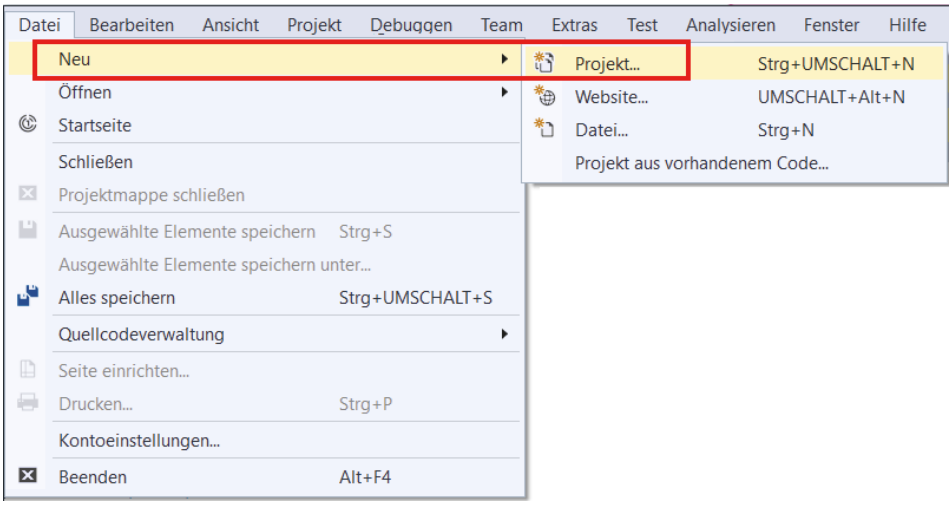
- Über das DATEI-Menü kannst du Dateien speichern, laden (öffnen), ausdrucken, neu erstellen oder Visual Studio beenden.
- Das BEARBEITEN-Menü hilft dir bei der Bearbeitung deines Programmtextes, aber auch bei anderen Programmelementen. Außerdem kannst du dort bestimmte Arbeitsschritte rückgängig machen 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, wo du gerade mit der **rechten** Maustaste hinklickst.

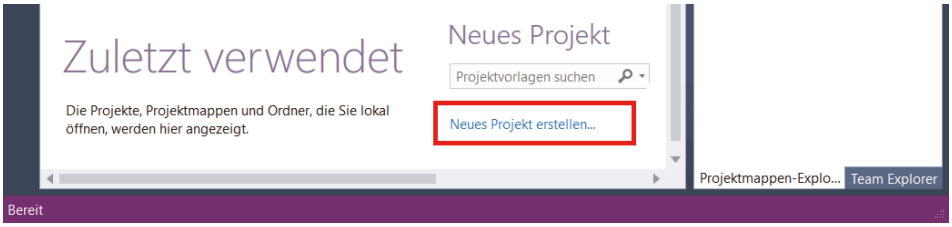
Ein Editorfenster, wie du es vielleicht von einem Editor oder Textverarbeitungsprogramm her kennst, ist gerade nicht in Sicht. Was tun? Unser Ziel ist es, ein neues Projekt – unser Erstlingswerk – zu erstellen. Also los!

DAS ERSTE PROGRAMM

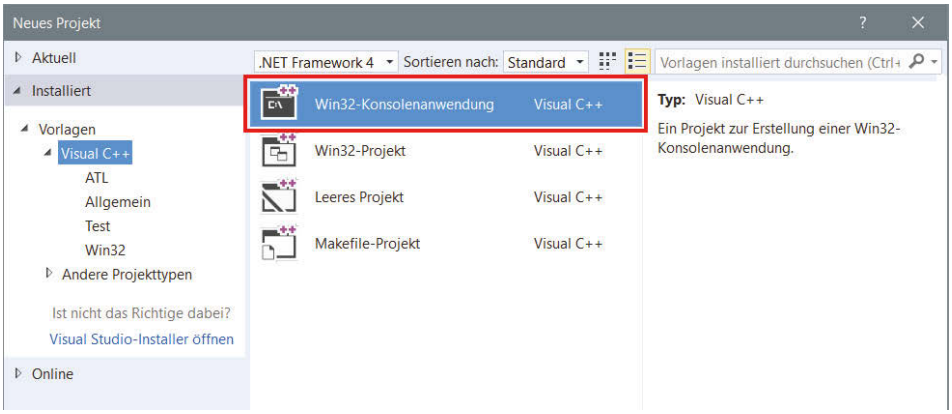
≫ Klicke in der Menüleiste auf DATEI und im sich öffnenden Menü auf NEU und dann auf PROJEKT.



≫ Oder du suchst im Fenster von Visual Studio weiter unten den Eintrag NEUES PROJEKT ERSTELLEN und klickst darauf.



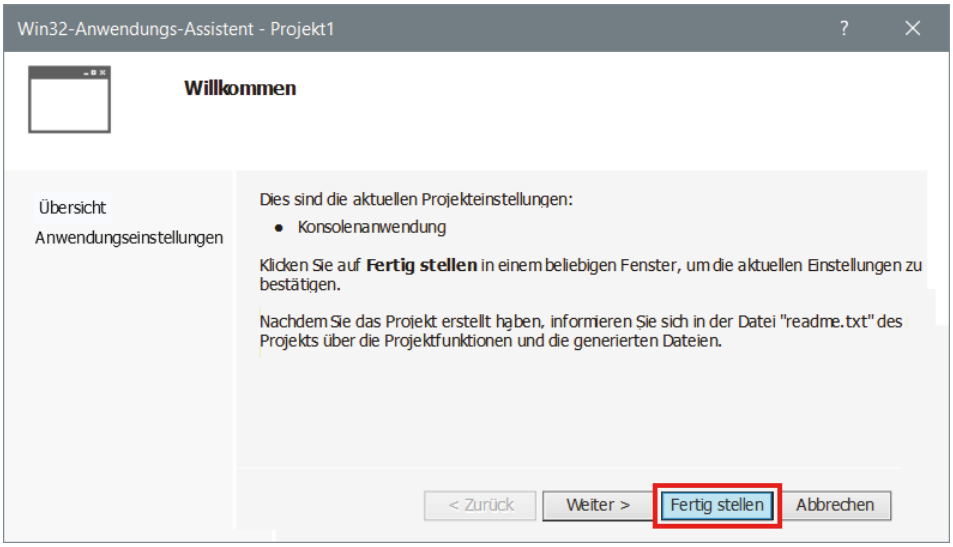
Es erscheint ein Dialogfeld, in dem es offenbar mehrere Möglichkeiten gibt, wie du dein Projekt erstellst:



≫ Sorge dafür, dass links der Eintrag VISUAL C++ markiert ist. Danach wählst du rechts WIN32-KONSOLENANWENDUNG.

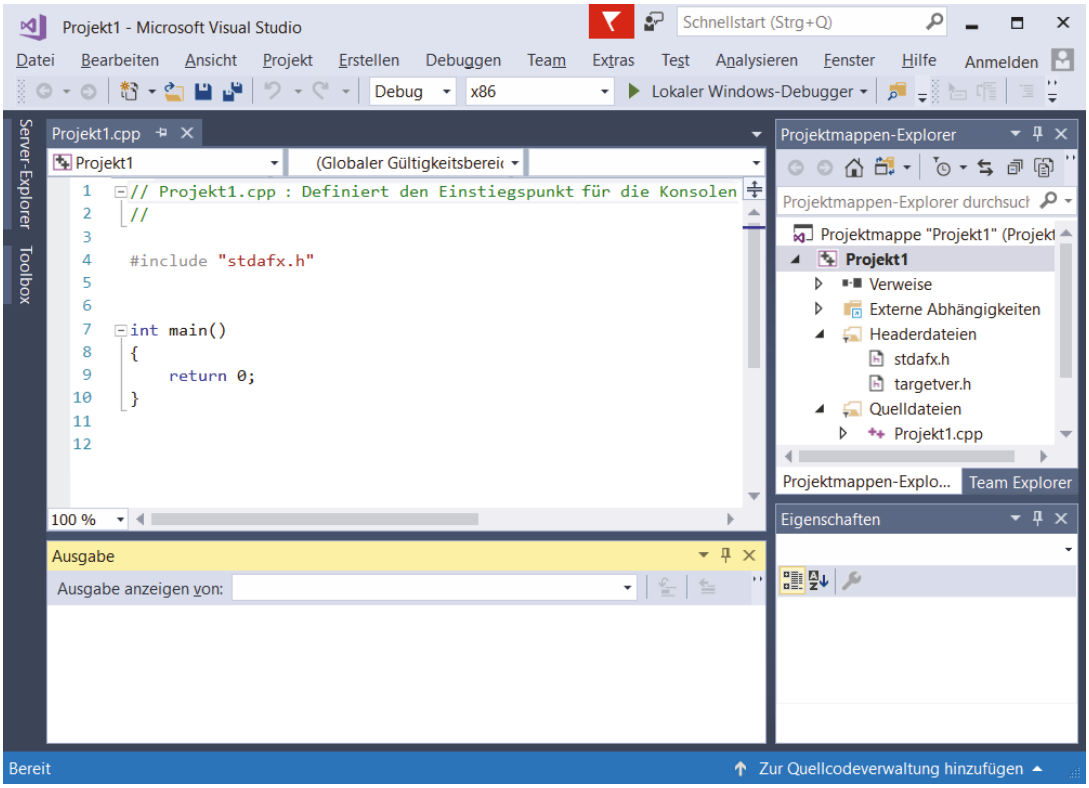
≫ Gib dem Projekt einen Namen und sorge dafür, dass hinter SPEICHERORT der Pfad steht, wo du dein Projekt unterbringen willst. Dann klicke abschließend auf OK.

Ich habe das Ganze einfallslos erst mal PROJEKT1 genannt. Du kannst den 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:*.

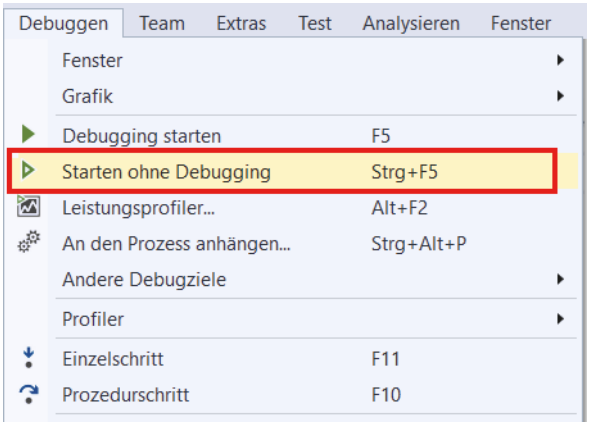


≫ Mit einem anschließenden Klick auf FERTIG STELLEN hast du dein erstes Mini-Projekt in C++.

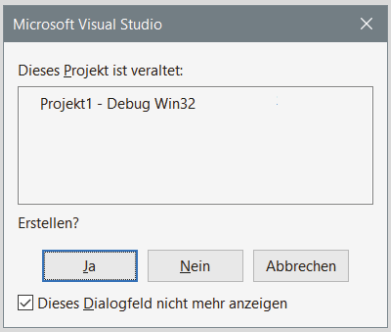
Visual Studio bietet also schon ein Gerüst-Programm, das du natürlich auch starten kannst, allerdings passiert (noch) nichts Aufregendes.



≫ Probiere ruhig mal aus, was sich tut, wenn du in der Menüleiste auf DEBUGGEN und STARTEN OHNE DEBUGGING klickst. (Alternativ kannst du auch die Tastenkombination Strg + F5 drücken.)

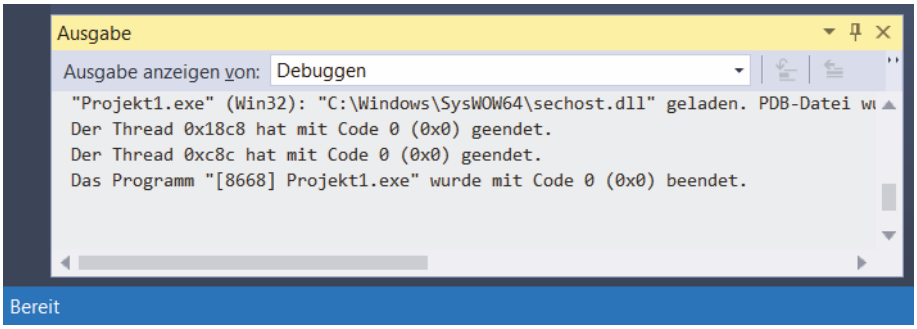


Möglicherweise erscheint beim ersten Start ein solches Meldefenster:

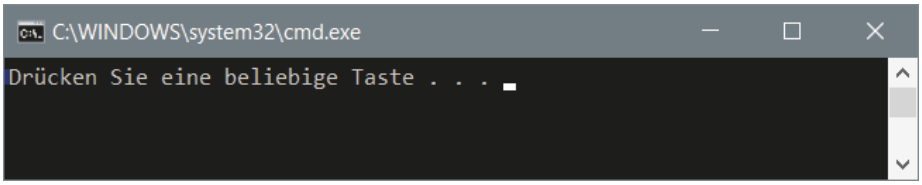


Sorge dafür, dass vor dem Eintrag DIESES DIALOGFELD NICHT MEHR ANZEIGEN ein Häkchen steht. Dann klicke auf JA.

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



Aber da ist auch noch ein Fenster mit schwarzem Hintergrund, das sich vor Visual Studio schiebt. In dem steht aber nichts weiter als 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 schauen wir uns jetzt das näher an, was da links oben im Editorfenster steht:

```
// Projekt1.cpp : Definiert den Einstiegspunkt für die
Konsolenanwendung.
//
#include "stdafx.h"

int main()
{
  return 0;
}
```

Das, was du da siehst, wird **Quelltext** genannt. Die ersten beiden 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. Für ein funktionierendes Programm sind nur diese Zeilen nötig:

```
#include "stdafx.h"
int main()
{
  return 0;
}
```

Die erste wichtige Zeile ist diese:

```
#include "stdafx.h"
```

Mit `#include` sorgen wir dafür, dass etwas in unser Programm eingebunden wird. Aber wozu braucht dieses kleine Programm irgendetwas von außen? Das führt uns zur Frage: Was passiert da eigentlich, wenn du auf den »Startknopf« drückst?

So, wie der Quelltext da steht, kann ein Computer damit nichts anfangen. Denn der hat seine eigene Sprache, **Maschinensprache** genannt, die nur er versteht (und noch ein paar menschliche Experten). Die Aufgabe von Visual Studio ist es, den für uns lesbaren Quelltext in etwas zu übersetzen, das der Computer »lesen« kann. Das nennt man **Kompilieren** und der Teil von Visual Studio, der das übernimmt, heißt **Compiler**.

Damit ein Programm richtig funktioniert, sind mehr Zutaten nötig als unser Quelltext. Und damit diese Zutaten mit eingebunden werden, benötigt der Compiler einen Hinweis. Und das ist das Wörtchen `#include`. Das sorgt dafür, dass das Modul mit dem Namen *stdafx.h* ins fertige Programm eingefügt wird, damit es nachher auch richtig funktioniert.

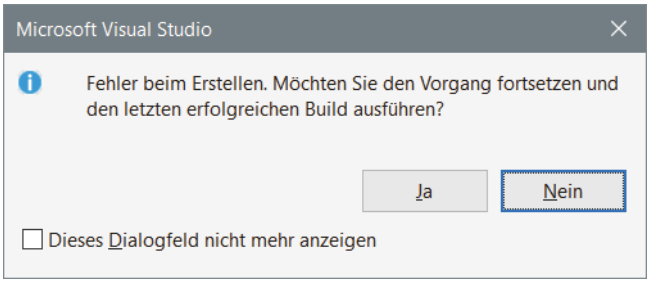
≫ Machen wir einen Test; dazu musst du den Quelltext so ändern:

```
// #include "stdafx.h"
int main()
{
  return 0;
}
```

Nun wird die `include`-Zeile als Kommentar behandelt, der Compiler überspringt jede Kommentarzeile, als wäre sie gar nicht da.

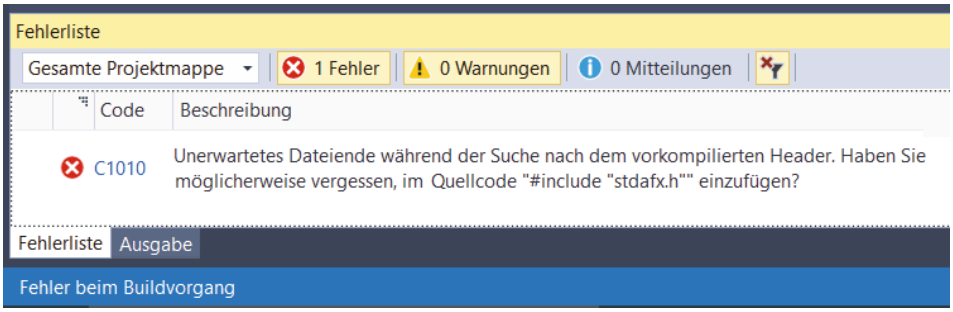
≫ Starte das Programm über DEBUGGEN und STARTEN OHNE DEBUGGING oder mit Strg + F5.

Als Erstes bekommst du dieses Meldefenster zu sehen:



≫ Klicke auf NEIN.

Und nun schauen wir mal nach unten ins Ausgabe-Fenster und suchen dort nach einer Fehlerliste.



Dort finden wir den Hinweis, dass offenbar eine Datei namens *stdafx.h* nötig ist, um das Programm ordnungsgemäß ausführen zu können.

Genau genommen gilt das nur für Visual C++, also wenn du mit Visual Studio arbeitest. Bei anderen Programmsystemen mit C++ gibt es oft gar kein *stdafx.h*, es wird also nicht benötigt. Warum bei Visual C++? In der Fehlermeldung steht etwas von »vorkompilierten Headern«. Dazu jetzt nur so viel: Die machen das Starten des Programms um einiges schneller.

Schon bald wirst du sehen, dass wir noch mehr Zutaten brauchen, wenn wir unser Programm erweitern.

DER AUFBAU EINES PROGRAMMS

Schauen wir jetzt mal weiter. 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`.

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

Zu klären wäre aber, was das `return` bedeutet. Damit gibt die Hauptfunktion am Ende einen Wert zurück. In diesem Fall kann der etwas darüber aussagen, wie gut der Programmlauf geklappt hat, z.B.:

```
return 0;
```

Mit dieser Zeile wird eine 0 ausgegeben, was in der Regel so viel heißt wie »alles o.k.«.

HALLO

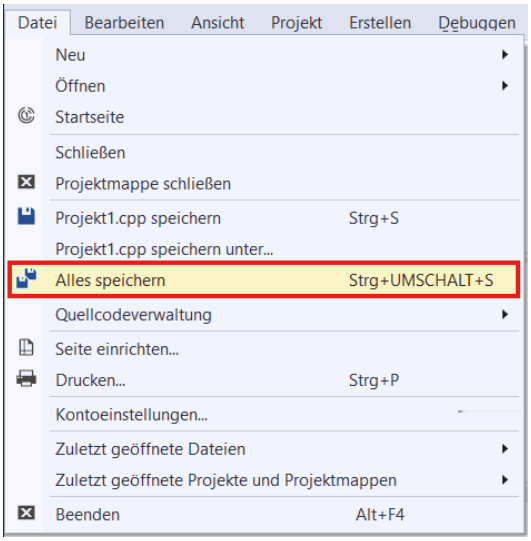
Und nun wird es Zeit für eine Programmerweiterung, damit auch wir etwas zu sehen bekommen.

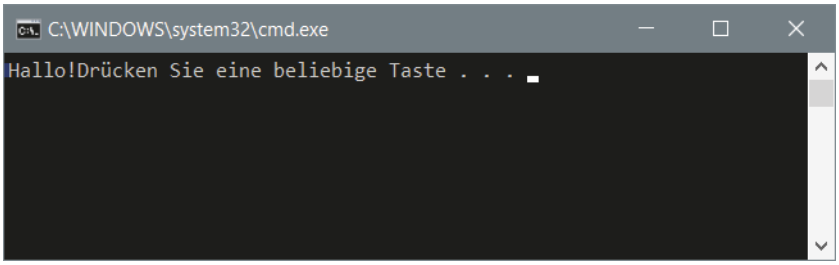
≫ Schaue dir den folgenden Quelltext an und tippe ihn genauso ein:

```
// Mein erstes Projekt
#include "stdafx.h"
#include <iostream>

int main()
{
  std::cout << "Hallo!";
  return 0;
}
```

≫ 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!«, aber leider wirkt die Aufforderung »Drücken Sie eine beliebige Taste« wie drangeklebt.