2 Grundkurs: Das erste Programm

Nun ist es endlich so weit! Wir werden unser erstes C++-Programm erstellen.

2.1 Hallo Welt! – das Programmgerüst

Es gibt eine Reihe von typischen Programmelementen, die man in so gut wie jedem C++-Programm wiederfindet. Diese Elemente werden wir uns jetzt einmal näher anschauen.

Bevor ich Ihnen die Programmelemente im Einzelnen vorstelle, sollten wir jedoch einen Blick auf den vollständigen Quelltext des Programms werfen. Wenn Sie bereits über etwas Programmiererfahrung verfügen, werden Sie das Programm womöglich sogar wiedererkennen: Es ist eine Adaption des klassischen „Hello World“-Programms aus der C-Bibel von Kernighan und Ritchie.

Listing 2.1 Das erste Programm (aus HalloWelt.cpp)

```
/*****************************************************
 * Hallo Welt-Programm
 *
 * gibt einen Gruss auf den Bildschirm aus
 */

#include <iostream>
using namespace std;

int main()
{
   cout << "Hallo Welt!" << endl;

   return 0;
}
```

Packen Sie jetzt bitte Ihr Sezierbesteck aus und schärfen Sie Ihren Verstand. Wir beginnen mit der Analyse.

Achtung!

C++ unterscheidet streng zwischen Groß- und Kleinschreibung. Beachten Sie dies, wenn Sie in Abschnitt 2.2 den Quelltext in Ihren Editor eingeben.

2.1.1 Typischer Programmaufbau

Die landläufige Vorstellung von einem Programm ist gemeinhin eine Folge von Anweisungen, die vom Rechner nacheinander ausgeführt werden. Blickt man aber in die Quelltextdatei eines beliebigen C++-Programms, offenbart sich ein ganz anderes Bild.

Tatsächlich bestehen C++-Programme aus:

- Kommentaren
- Präprozessordirektiven
- Namensräumen
- Deklarationen und Definitionen
- einer Eintrittsfunktion namens `main()`

Natürlich gibt es auch Anweisungen, doch existieren diese nur als untergeordnete Elemente in den Definitionen der Funktionen!

Listing 2.2 Die typischen Elemente eines C++-Programms

```
/*****************************************************
 * Hallo Welt-Programm           // mehrzeiliger Kommentar
 *
 * gibt einen Gruss auf den Bildschirm aus
 */

#include <iostream>              // Präprozessor-Direktive
using namespace std;             // Namensraum-Einbindung

int main()                       // Definition der Eintrittsfunktion
{
   cout << "Hallo Welt!" << endl;

   return 0;
}
```

Das Verhältnis aus Anweisungen (in Listing 2.1 fett hervorgehoben) und Elementen, die vornehmlich der Organisation des Quelltextes dienen (Präprozessordirektiven, Definitionen etc.), ist nicht immer so drastisch wie in diesem HalloWelt-Programm. Doch eines können und sollten Sie aus diesem Beispiel bereits ablesen: Programmierung hat auch viel mit Codeorganisation zu tun!

Merksatz

Bei der C++-Programmierung – wie im Übrigen bei der Programmierung mit jeder modernen Programmiersprache – genügt es nicht, sich zu überlegen, welche Anweisungen in welcher Reihenfolge zur effizienten Erledigung einer Aufgabe benötigt werden (Algorithmus). Sie müssen sich auch Gedanken darüber machen, wie Sie Ihren Quelltext organisieren.

Fürs Erste werden wir die Codeorganisation so einfach wie möglich halten. Konkret bedeutet dies, dass wir während unserer ersten Gehversuche mit C++ einfach unsere gesamten Anweisungen in die `main()`-Funktion schreiben werden.

Wo aber kommt diese `main()`-Funktion her? Und welche Bedeutung haben die anderen Elemente des Grundgerüsts aus Listing 2.1?

2.1.2 Die Eintrittsfunktion main()

Wenn Sie ein C++-Programm aufrufen, wird der Code des Programms in den Arbeitsspeicher geladen und vom Prozessor ausgeführt. Doch mit welchem Code beginnt die Ausführung des Programms?

Per Konvention beginnen C++-Programme immer mit einer Funktion namens `main()`. Wenn Sie einen Quelltext zu einer *.exe*-Datei kompilieren lassen, generiert der Compiler automatisch Startcode, der dafür sorgt, dass die Programmausführung mit der ersten Anweisung in `main()` beginnt.

Ihre Aufgabe ist es daher, in Ihrem Programmquelltext eine passende `main()`-Eintrittsfunktion zu definieren:

```
int main()
{
   // hier können Sie eigenen Code einfügen

   return 0;
}
```

Was diese Definition im Einzelnen zu bedeuten hat, werden Sie erst in Kapitel 6 erfahren, wenn wir uns intensiver mit der Definition von Funktionen beschäftigen. Bis dahin ist nur eines wichtig: Sie dürfen den Definitionscode nicht verändern, da die Funktion sonst nicht mehr vom Compiler als Eintrittsfunktion erkannt wird.

Wenn Sie also das `int` vergessen oder den `return`-Befehl falsch schreiben oder versuchen, die Funktion von `main()` in `start()` umzutaufen, so werden Sie dafür bei der Programmerstellung entsprechende Fehlermeldungen ernten. Und achten Sie auch auf die Groß- und Kleinschreibung. Für C++ sind `main` und `Main` nicht zwei Schreibweisen eines Namens, sondern ganz klar zwei verschiedene Namen!

Merksatz

C++ unterscheidet strikt zwischen Groß- und Kleinschreibung!

Ich sollte allerdings noch erwähnen, dass es eine zweite Variante für die Definition der Eintrittsfunktion `main()` gibt:

```
int main(int argc, char *argv[])
{
   // hier können Sie eigenen Code einfügen

   return 0;
}
```

Ja, manche Compiler erlauben sogar noch weitere Varianten. Grundsätzlich sollten Sie sich aber auf die beiden obigen Varianten beschränken, da nur so sichergestellt ist, dass sich Ihr Programm mit jedem ANSI-kompatiblen Compiler übersetzen lässt.

Wenn Sie den Compiler anweisen, aus einem Quelltext, der keine korrekt definierte `main()`-Eintrittsfunktion enthält, ein *.exe*-Programm zu erzeugen, werden Sie am Ende des Erstellungsprozesses vom Linker eine Fehlermeldung erhalten, dass die im Startcode referenzierte `main()`-Funktion nicht gefunden werden konnte.

2.1.3 Die Anweisungen

Innerhalb der geschweiften Klammern unserer `main()`-Funktion können wir nun endlich die Anweisungen aufsetzen, die bei Start des Programms ausgeführt werden sollen. Im Falle unseres ersten Beispielprogramms bescheiden wir uns mit einer einzigen Zeile, die den Text „Hallo Welt!“ ausgeben soll.

```
int main()
{
   cout << "Hallo Welt!" << endl;

   return 0;
}
```

Was bewirkt die obige Anweisung? Zunächst muss man wissen, dass `cout` ein vordefiniertes Objekt ist, welches die Konsole repräsentiert.

Die Konsole ist ein spezielles Programm des Betriebssystems (siehe Kasten), über das der Anwender Befehle ans Betriebssystem schicken kann. Für uns als Programmierer ist sie interessant, weil wir sie zum Datenaustausch zwischen unseren Programmen und den Anwendern nutzen können. Wir ersparen uns also den Aufbau einer eigenen Benutzeroberfläche und können uns ganz auf den reinen C++-Code konzentrieren.

Die Konsole

Die meisten PC-Benutzer, vor allem Windows- oder KDE-Anwender, sind daran gewöhnt, dass die Programme als Fenster auf dem Bildschirm erscheinen. Dies erfordert aber, dass das Programm mit dem Fenstermanager des Betriebssystems kommuniziert und spezielle Optionen und Funktionen des Betriebssystems nutzt. Programme, die ohne fensterbasierte, grafische Benutzeroberfläche (GUI = graphical user interface) auskommen, können hierauf jedoch verzichten und stattdessen die Konsole zum Datenaustausch mit dem Benutzer verwenden.

Die Konsole ist eine spezielle Umgebung, die dem Programm vorgaukelt, es lebe in der guten alten Zeit, als es noch keine Window-Systeme gab und immer nur ein Programm zurzeit ausgeführt werden konnte. Dieses Programm konnte dann uneingeschränkt über alle Ressourcen des Rechners verfügen – beispielsweise die Tastatur, das wichtigste Eingabegerät, oder auch den Bildschirm, das wichtigste Ausgabegerät. Der Bildschirm war in der Regel in den Textmodus geschaltet, wurde also nicht aus Pixelreihen, sondern aus Textzeilen aufgebaut.

Unter Windows heißt die Konsole MS-DOS-Eingabeaufforderung oder auch nur Eingabeaufforderung und kann je nach Betriebssystem über **Start/Programme** oder **Start/Programme/Zubehör** aufgerufen werden.

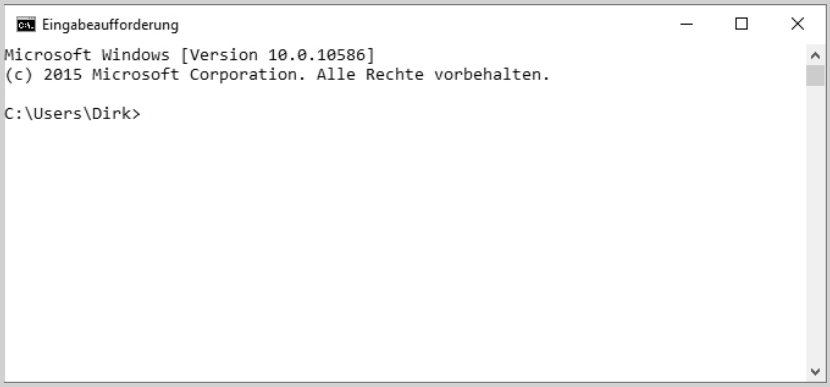

Bild 2.1 Die Konsole von Windows 8 (mit invertiertem Hintergrund)

Damit wir innerhalb eines Programms auf die Konsole zugreifen können, muss es im Programmcode aber ein Element geben, welches die Konsole repräsentiert. Dieses Element ist wie gesagt `cout`.

Allerdings handelt es sich bei `cout` nicht um ein Element, das fest in die Sprache integriert es. Vielmehr verbirgt sich hinter `cout` ein Objekt, das im Code der C++-Standardbibliothek definiert ist. Gleiches gilt im Übrigen auch für den Operator `<<`, der Ausgaben an `cout` schickt, sowie `endl`, das in der Ausgabe einen Zeilenumbruch (**end**-of-**l**ine = Zeilenende) erzeugt.

Unsere Anweisung schickt also zuerst mit Hilfe des `<<`-Operators den Text `Hallo Welt!` und dann noch einen Zeilenumbruch (`endl`) zur Konsole (`cout`):

```
cout << "Hallo Welt!" << endl;
```

Zwei Punkte an diesem Code bedürfen noch einer besonderen Erwähnung:

Merksatz

Texte, die ein Programm verarbeitet, werden auch als *Strings* bezeichnet und stehen in Anführungszeichen `""`, damit der Compiler sie vom Programmcode unterscheiden kann.

Merksatz

Anweisungen dürfen in C++ nur innerhalb von Funktionen stehen und müssen mit einem Semikolon abgeschlossen werden.

2.1.4 Headerdateien

Erinnern Sie sich, was in Kapitel 1 über die Verwendung von Elementen gesagt wurde, die nicht in die Sprache integriert sind: Sie müssen im Programmcode einmal definiert und in jeder Quelltextdatei, in der sie verwendet werden, deklariert werden.

Wie sieht es also mit der Definition aus? Die Definition von `cout`, `<<` und `endl` findet sich im Code der C++-Standardbibliothek. Dieser Code wird bei der Programmerstellung vom Linker automatisch mit Ihrem Code zur ausführbaren *.exe*-Datei verbunden. Wir müssen uns um diesen Teil nicht weiter kümmern. (Außer der Compiler wäre nicht korrekt konfiguriert und findet die Bibliotheksdateien nicht. Diese stehen übrigens meist in einem Verzeichnis *lib* und der Pfad zu diesem Verzeichnis kann über die Compiler-Optionen eingestellt werden.)

Bleibt noch die Deklaration. Da wir `cout`, `<<` und `endl` in unserer Quelltextdatei *HalloWelt.cpp* verwenden, müssen wir die Elemente dem Compiler auch in dieser Datei bekannt machen. Wir könnten dazu so vorgehen, dass wir in der Fachliteratur, der Bibliotheksdokumentation oder – soweit vorhanden – gar direkt im Quelltext der Bibliothek nachschlagen, wie die betreffenden Bibliothekselemente definiert sind, und uns daraus die Deklarationen ableiten, die wir dann über `main()` in den Quelltext einfügen.

Sie werden mir allerdings sicher zustimmen, dass diese Verfahrensweise recht mühselig, kompliziert und fehleranfällig wäre. Die C++-Standardbibliothek stellt daher für jeden Themenbereich, den die Bibliothek abdeckt, eine passende Headerdatei zur Verfügung, in der die benötigten Deklarationen schon gesammelt sind. Die Headerdatei für alle Bibliothekselemente, die mit der Ein- und Ausgabe zu tun haben, heißt *iostream* und kann mit der Präprozessor-Direktive `#include` einkopiert werden

```
#include <iostream>

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

Auch diese Technik ist Ihnen – in der Theorie – bereits in Kapitel 1.3.3 vorgestellt worden. Die wichtigsten Fakten möchte ich aber trotzdem hier noch einmal zusammenfassen.

Die `#include`-Direktive sucht nach der angegebenen Datei. Da der Dateiname in eckigen Klammern steht, wird die Datei im Include-Pfad des Compilers gesucht. (Dieser ist nach der Installation üblicherweise automatisch so eingestellt, dass er auf das Verzeichnis mit den Headerdateien der C++-Standardbibliothek verweist. Sie müssen sich in der Regel also nicht weiter um diese Einstellung kümmern.)

Der Inhalt der Datei wird dann an der Stelle der Direktive in die Quelltextdatei einkopiert.

Faktisch fügt die obige `#include`-Direktive also die Deklarationen aller IO-Elemente der C++-Standardbibliothek ein, sodass wir diese Elemente (darunter eben auch `cout`, `<<` und `endl`) verwenden können. Die Einbindung des Namensraums `std` dient dann nur noch der Bequemlichkeit, damit wir die Bibliothekselemente allein mit ihrem Namen ansprechen können (also `cout` statt `std::cout`, siehe Kapitel 1.3.4).

Das engl. Akronym IO steht für Input/Output, zu deutsch also Ein- und Ausgabe.

2.1.5 Kommentare

Wir haben nun fast alle Bestandteile des Quelltextes analysiert. Übrig geblieben sind allein die ersten einleitenden Zeilen:

```
/*****************************************************
 * Hallo Welt-Programm
 *
 *  gibt einen Gruss auf den Bildschirm aus
 */

#include <iostream>
...
```

Bei diesen Zeilen handelt es sich um einen Kommentar. Kommentare dienen dazu, erklärenden Text direkt in den Quellcode einzufügen – quasi als Erklärung oder Gedankenstütze für den Programmierer.

C++ kennt zwei Formen des Kommentars:

- Will man eine einzelne Zeile oder den Rest einer Zeile als Kommentar kennzeichnen, verwendet man die Zeichenfolge //. Alles, was hinter der Zeichenfolge // bis zum Ende der Quelltextzeile steht, wird vom Compiler als Kommentar angesehen und ignoriert.

```
int main()       // Kommentar
```

- Mehrzeilige Kommentare beginnt man dagegen mit /* und schließt sie mit */ ab. Oder Sie müssen jede Zeile mit // beginnen.

```
/* Kommentar
   über mehrere
   Zeilen */
```

Kommentare werden vom Compiler ignoriert, d. h., er löscht sie, bevor er den Quelltext in Maschinencode umwandelt. Sparsam veranlagte Leser brauchen sich also keine Sorgen darüber zu machen, dass eine ausführliche Kommentierung die Größe der ausführbaren Programmdatei aufplustern könnte. ■

Sinnvolles Kommentieren

So einfache Programme, wie wir sie am Anfang dieses Buchs erstellen, bedürfen im Grunde keiner Kommentierung. Kommentare sind nicht dazu gedacht, einem Programmieranfänger C++ zu erklären. Kommentare sollen gestandenen C++-Programmierern helfen, sich in einen Quelltext einzudenken und diesen zu erklären. Kommentare sollten daher eher kurz und informativ sein. Kommentieren Sie beispielsweise die Verwendung wichtiger Variablen (siehe nachfolgendes Kapitel) sowie die Aufgabe größerer Anweisungsabschnitte. Einfache Anweisungen oder leicht zu verstehende Konstruktionen sollten nicht kommentiert werden.

2.2 Programmerstellung

Um aus dem Programmquelltext *HalloWelt.cpp* ein ausführbares Programm zu erzeugen, müssen wir den Quelltext mit Hilfe des C++-Compilers in Maschinencode übersetzen.

Wie Sie dabei vorgehen, hängt davon ab, welche Entwicklungsumgebung Sie verwenden. Zwei Entwicklungsumgebungen möchte ich Ihnen im Folgenden vorstellen: die Visual-Studio-Community-Edition für Windows -Desktop und den GNU-Compiler für Linux.

2.2.1 Programmerstellung mit Visual Studio

Wenn Sie mit Visual Studio arbeiten, steht Ihnen eine komplette, leistungsfähige Entwicklungsumgebung zur Verfügung. Viele Leser werden die Arbeit mit der grafischen Benutzeroberfläche von Visual Studio als angenehmer empfinden als die Arbeit mit einem Compiler, der von der Konsole aus bedient wird.

Allerdings fällt bei der Arbeit mit einer Entwicklungsumgebung etwas mehr Verwaltungsarbeit an. Zum Beispiel verwaltet Visual Studio alle Dateien und Daten, die zu einem Pro-

gramm gehören, in Form eines Projekts. Der erste Schritt bei der Programmentwicklung mit Visual Studio besteht daher darin, ein passendes Projekt anzulegen.

Projekt anlegen

1. Rufen Sie Visual Studio auf. Sie können das Programm z. B. über die Programmgruppe auswählen (unter Windows 7 zu finden im Programme-Ordner des Start-Menüs, bei Windows 10 als Kachel auf der Startseite oder in der App-Ansicht) oder suchen Sie mit der Systemsuche (Suchfeld im Start-Menü für Windows 7, Suchfeld in der Taskleiste für Windows 10) nach **Visual Studio**.

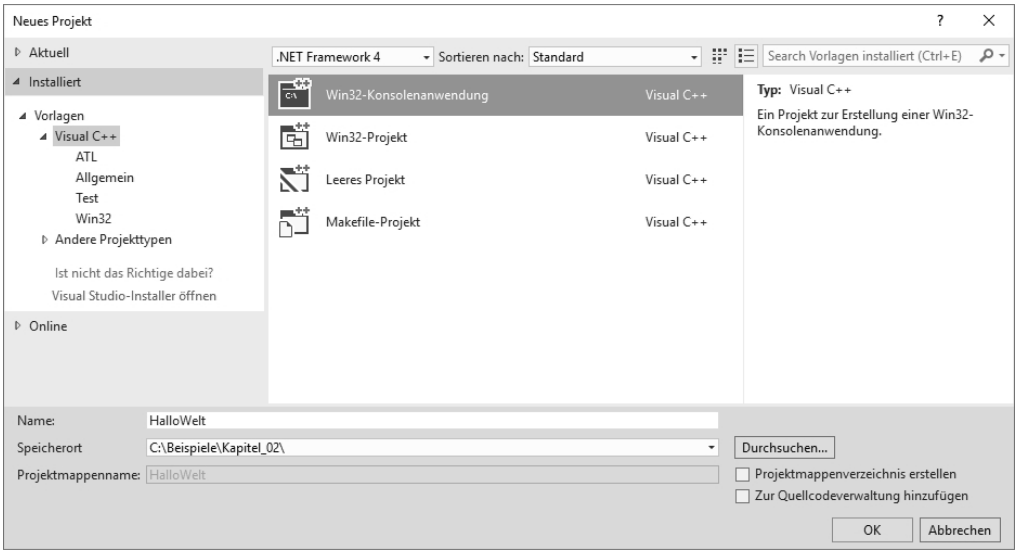

Bild 2.2 Anlegen eines neuen C++-Projekts mit Visual Studio

2. Legen Sie ein neues Projekt an. Rufen Sie dazu den Befehl **Datei/Neu/Projekt** auf.
   - Achten Sie darauf, dass links im Dialogfeld **Neues Projekt** die Vorlagenkategorie **Visual C++** ausgewählt ist. Falls nicht, klicken Sie einfach im linken Teilfenster auf den gleichnamigen Eintrag. Wählen Sie dann im mittleren Fenster die Vorlage **Win32-Konsolenanwendung** aus.
   - Geben Sie einen **Namen** für das Projekt ein, beispielsweise **HalloWelt**, und wählen Sie im Feld **SpeicherOrt** ein übergeordnetes Verzeichnis für das Projekt aus. (Visual Studio wird unter diesem Verzeichnis ein Unterverzeichnis für das Projekt anlegen, das den gleichen Namen wie das Projekt trägt.)
   - Achten Sie darauf, dass die Option **Projektmappenverzeichnis erstellen** deaktiviert ist.
   - Drücken Sie zuletzt auf **OK**.
3. Klicken Sie auf der ersten Seite des aufspringenden Assistenten auf **Weiter**.
4. Deaktivieren Sie auf der zweiten Seite die Option **Vorkompilierter Header** und aktivieren Sie dafür die Option **Leeres Projekt**. Klicken Sie auf **Fertig stellen**.

Wenn Sie die Option **Leeres Projekt** nicht aktivieren, legt Visual Studio für Sie eine *.cpp*-Quelltextdatei mit einem einfachen Programmgerüst an. Wir verzichten allerdings auf dieses Programmgerüst, da es a) nur wenig Arbeitserleichterung bringt und b) kein standardisiertes C++ verwendet.

Vorkompilierte Header dienen dazu, die Programmerstellung zu beschleunigen. Sie werden bei der ersten Kompilierung erstellt und können nachfolgende Kompilierungen beschleunigen. Wenn Sie an größeren Projekten arbeiten, ist dies eine recht nützliche Option. Für unsere kleinen Beispielprogramme können wir allerdings auf den „Header", der viel Speicherplatz belegt, verzichten.

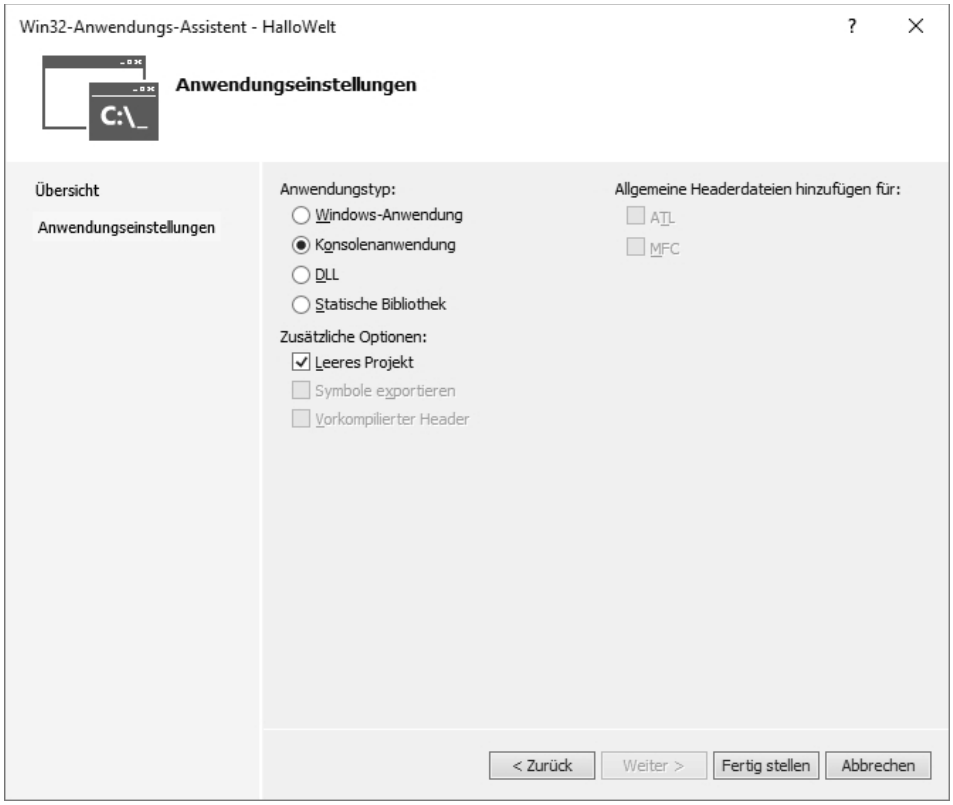

Bild 2.3 Beginnen Sie mit einem leeren Projekt.

Projektmappen

Visual Studio bettet das neue Projekt automatisch in eine Projektmappe ein. Wenn Sie möchten, können Sie über den Befehl **Datei/Neu/Projekt** weitere Projekte in die aktuelle Projektmappe aufnehmen. Sie müssen dann nur im Dialogfenster **neues Projekt** im Listenfeld **Projektmappe** die Option **Hinzufügen** auswählen. Für den Einstieg ist es aber sinnvoller, für jedes neue Programm ein neues Projekt in einer eigenen Projektmappe anzulegen.

Im Projektmappen-Explorer (Aufruf über den gleichnamigen Befehl im Menü **Ansicht**), der standardmäßig links im Visual-Studio-Fenster angezeigt wird, werden alle Projekte der aktuellen Projektmappe zusammen mit den zu den Projekten gehörenden Dateien aufgeführt.

Quelltextdatei hinzufügen

Da wir unsere Arbeit mit einem leeren Projekt begonnen haben, besteht der nächste Schritt darin, dem Projekt eine Quelltextdatei hinzuzufügen.

5. Fügen Sie dem Projekt eine Quelltextdatei hinzu. Klicken Sie dazu mit der rechten Maustaste im Projektmappen-Explorer auf den Projektknoten (in unserem Beispielprojekt ist dies der Knoten mit dem fett dargestellten Namen *HalloWelt*) und rufen Sie im Kontextmenü den Befehl **Hinzufügen/Neues Element** auf.

Falls Sie das Fenster des Projektmappen-Explorers nicht sehen, können Sie es über den Menübefehl **Ansicht/Projektmappen-Explorer** einblenden lassen.

6. Wählen Sie im erscheinenden Dialogfeld die Vorlage *C++-Datei (.cpp)* aus, geben Sie einen Namen für die Datei an und klicken Sie auf **Hinzufügen**.

Die Beispielprogramme der ersten Teile bestehen meist nur aus einer Quelltextdatei, die dann der Einfachheit halber und zur leichteren Identifizierung den Namen des Projekts trägt.

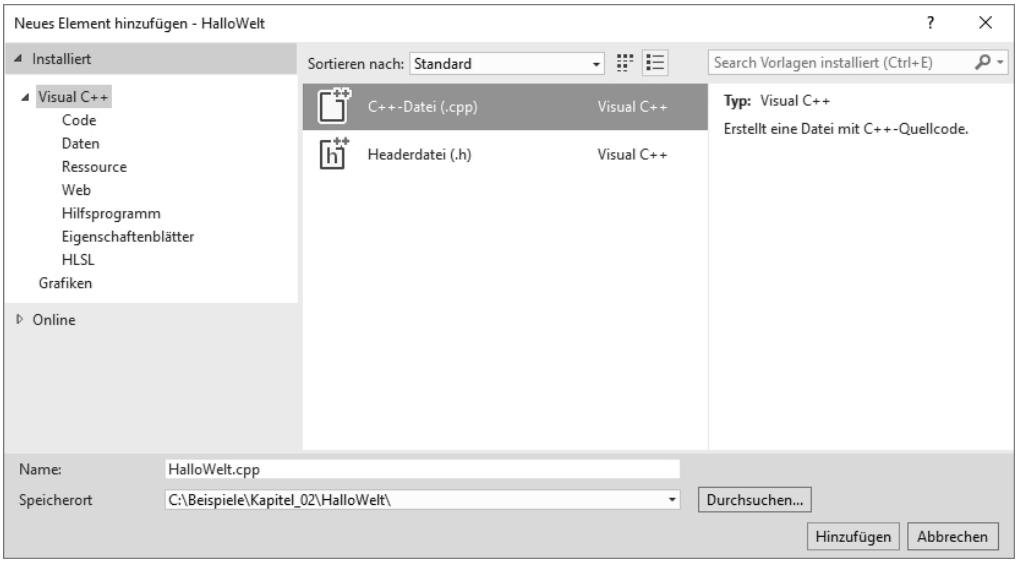

Bild 2.4 Dem Projekt eine Quelltextdatei hinzufügen

Visual Studio legt die neue Quelltextdatei an und lädt sie automatisch in den Editor. Im Projektmappen-Explorer wird die Datei unter dem Knoten **Quelldateien** aufgelistet.

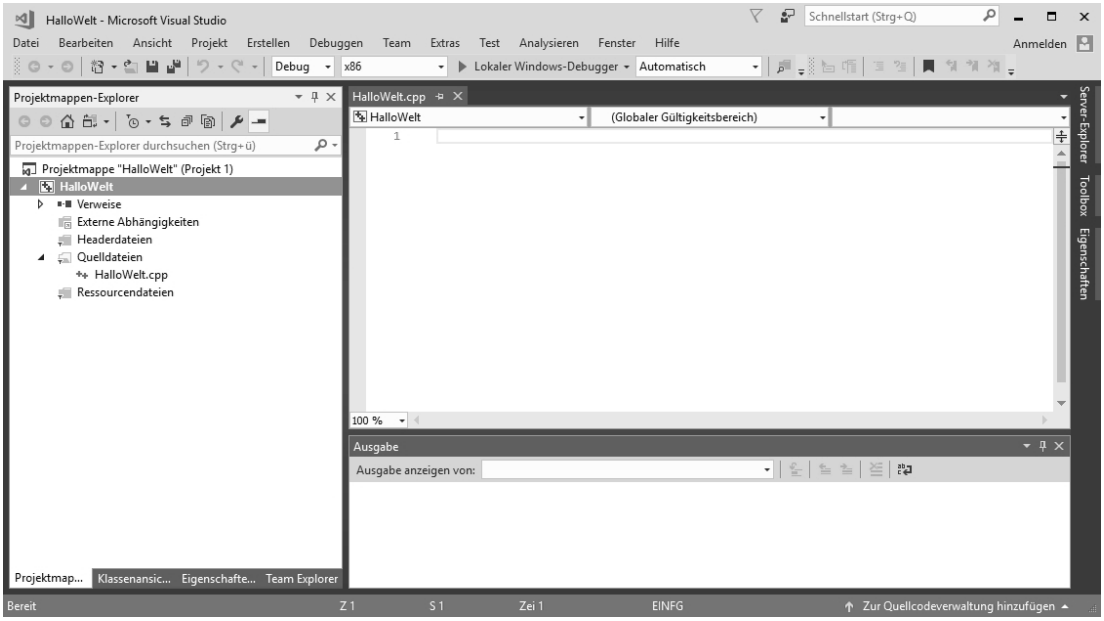

Bild 2.5 Das neue Projekt mit geöffneter, aber noch leerer Quelltextdatei

Wenn Sie im Projektmappen-Explorer auf den Knoten einer Datei doppelklicken, wird die Datei zur Bearbeitung in den Editor geladen.

Quelltext aufsetzen

7. Tippen Sie jetzt den Quelltext aus Listing 2.1 in die Quelltextdatei ein.
8. Speichern Sie zur Sicherheit das Projekt (Befehl **Datei/Alles speichern**).

Projekt kompilieren

Jetzt kommt der große Augenblick. Wir lassen den Quelltext des Programms kompilieren.

9. Um das Programm zu kompilieren und zu erstellen, rufen Sie den Menübefehl **Erstellen/Projektmappe Erstellen** auf.

Im Ausgabefenster, das in den unteren Rand des IDE-Rahmenfensters integriert ist, wird der Fortgang des Erstellungsprozesses angezeigt. Zum Schluss sollte im Ausgabefenster eine Erfolgsmeldung der Form *Erstellen: 1 erfolgreich* und in der Statusleiste die Meldung *Erstellen erfolgreich* zu lesen sein.

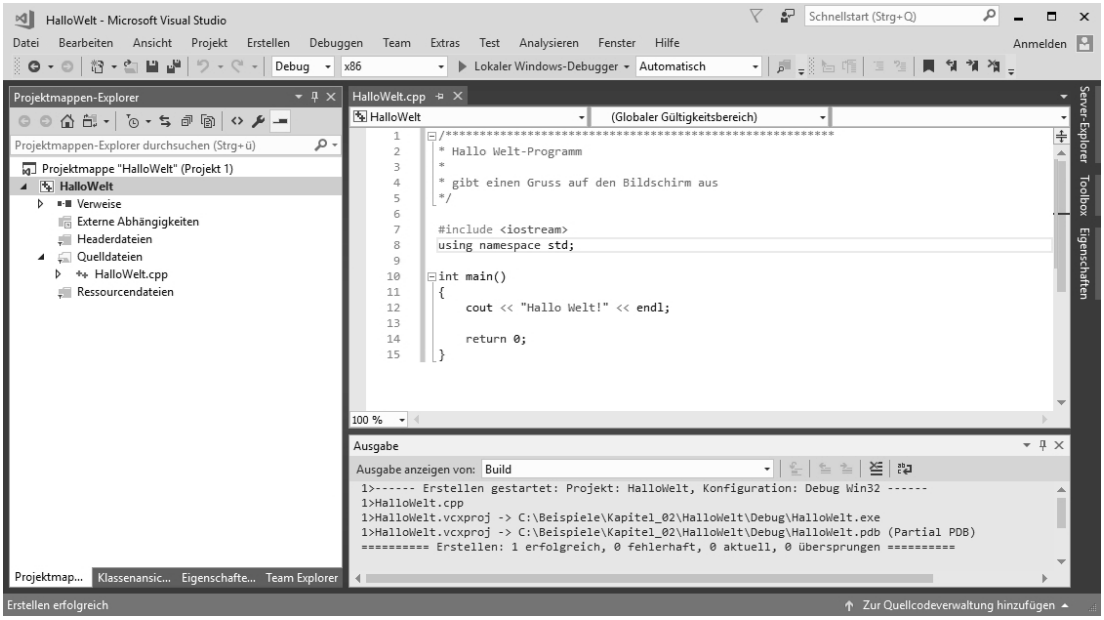

Bild 2.6 Das erfolgreich kompilierte Programm

Sollten bei der Kompilierung Fehler auftreten, gehen Sie so vor, dass Sie die Fehlerliste einblenden lassen (Menübefehl **Ansicht/Fehlerliste**) und dort auf den Reiter **Fehlerliste** klicken. Lesen Sie den Text zu dem ersten Fehler und doppelklicken Sie dann auf die Fehlermeldung, damit Visual Studio die betreffende Stelle im Quelltext markiert. Versuchen Sie, den Fehler zu korrigieren, und erstellen Sie dann das Projekt erneut. Wiederholen Sie diesen Vorgang, bis das Programm erfolgreich erstellt wird.

Manche Fehler erzeugen Folgefehler. Eine fehlende `#include`-Anweisung kann z. B. schnell ein Dutzend oder mehr Fehlermeldungen wegen nicht deklarierter Elemente provozieren.

Kompilieren und Erstellen

Unter *Kompilierung* im engeren Sinne versteht man allein die Übersetzung des Quelltextes in eine Objektdatei. *Erstellen* (engl. build) bedeutet dagegen, dass der Quelltext übersetzt und anschließend vom Linker auch noch mit dem Code weiterer benötigter Objektmodule (Objektdateien anderer Quelltextdateien, Code der Bibliotheken) zu einem ausführbaren Programm zusammengebunden wird. Im allgemeinen Sprachgebrauch wird Kompilieren allerdings oft auch synonym zu Erstellen verwendet.

Programm von Visual Studio aus starten

10. Sie können das Programm direkt von Visual Studio aus starten. Rufen Sie dazu einfach den Befehl **Debuggen/Starten ohne debugging** auf oder drücken und merken Sie sich gleich das Tastaturkürzel **(Strg)+(F5)**.

Bild 2.7 Ausführung eines Konsolenprogramms aus der Visual-Studio-IDE heraus

Visual Studio öffnet automatisch ein Konsolenfenster für das Programm und das Programm schickt seine Ausgaben zu diesem Fenster. Als Ergebnis sehen Sie im Konsolenfenster den Gruß `Hallo Welt!`.

Die Meldung „Drücken Sie eine beliebige Taste ...“ wird von der Visual-Studio-Entwicklungsumgebung hinzugefügt. Sie ist nicht Teil der Anwendung, sondern ein Service der Visual-Studio-IDE, der verhindert, dass das Konsolenfenster gleich wieder verschwindet. Zum Schließen des Fensters drücken Sie eine beliebige Taste.

Achtung!

Wenn Sie zum Ausführen den Befehl **Debuggen/debugging Starten** wählen, schließt sich das Konsolenfenster automatisch nach Beendigung der Programmausführung.

2.2.2 Programmerstellung mit GNU-Compiler

Der GNU-Compiler g++ bzw. gcc[1] ist auf vielen Linux-Systemen standardmäßig installiert.

Im Anhang finden Sie Hinweise, wie Sie testen können, ob der GNU-Compiler auf Ihrem System installiert ist und von wo Sie bei Bedarf eine aktuelle Version herunterladen können.

Quelltexte bearbeiten

1. Legen Sie z. B. mit dem Editor *vi* eine neue Datei namens *HalloWelt.cpp* an, tippen Sie den Quelltext ein und speichern Sie die Datei.

Statt des *vi* können Sie auch jeden beliebigen anderen reinen Texteditor verwenden, beispielsweise den *emacs*, *KEdit* oder *KWrite* unter KDE.

Kompilieren

2. Öffnen Sie ein Konsolenfenster. Wie Ihr Konsolenfenster aussieht und mit welchem Befehl es aufgerufen wird, hängt von Ihrer Linux-Version und dem verwendeten Window-Manager ab. Unter KDE können Sie Konsolenfenster in der Regel über die KDE-Taskleiste aufrufen.
3. Wechseln Sie in der Konsole mit Hilfe des *cd*-Befehls in das Verzeichnis, in dem der Programmquelltext steht.
4. Rufen Sie von der Konsole aus den GNU-Compiler auf. Übergeben Sie dem Compiler in der Kommandozeile den Namen der zu kompilierenden Datei sowie den Schalter *-o* mit dem gewünschten Namen für die ausführbare Datei.

```
g++ HalloWelt.cpp -o HalloWelt
```

oder auch

```
gcc HalloWelt.cpp -o HalloWelt
```

[1] Je nach verwendeter Compilerversion könnte der Compiler auch anders heißen (beispielsweise *egcs*).

Bild 2.8 Kompilation und Erstellung einer ausführbaren Datei

Programm ausführen

5. Tippen Sie in der Konsole den Namen des Programms ein und schicken Sie ab.

Unter Umständen müssen Sie angeben, dass das Programm im aktuellen Verzeichnis zu finden ist. Stellen Sie dazu dem Programmnamen den Punkt als Stellvertreter für das aktuelle Verzeichnis voran: `./HalloWelt`.

Bild 2.9 Ausführung von Konsole

2.2.3 Programmausführung

Unabhängig davon, auf welchem Weg Sie Ihr Programm erstellt haben, können Sie die vom C++-Compiler erzeugte ausführbare Programmdatei danach jederzeit aufrufen und ausführen.

Wo befindet sich die ausführbare Programmdatei?

Wenn Sie das Programm mit Visual Studio erstellen, legt die IDE die *.exe*-Datei unter dem Projektverzeichnis in einem Verzeichnis */Debug* ab.

Wenn Sie das Programm von der Konsole aus mit einem Konsolen-Compiler erstellen, wird die Programmdatei üblicherweise im aktuellen Verzeichnis angelegt.

Sie können Ihren Compiler aber auch so konfigurieren, dass er die Programmdatei in ein anderes Ausgabeverzeichnis schreibt.

Es gibt verschiedene Wege, ein Programm auszuführen, und alle führen letzten Endes über den Aufruf der ausführbaren Programmdatei (unter Windows ist dies die *.exe*-Datei). Dieser kann beispielsweise durch Doppelklick in einem Dateimanager, über Doppelklick auf eine zuvor angelegte Desktop-Verknüpfung oder aber auch aus einem Konsolenfenster heraus erfolgen.

Start aus Konsolenfenster

Für Konsolenanwendungen ist der Start aus einem zuvor explizit geöffneten Konsolenfenster nahezu ideal, da die Anwendung dann genau dieses Fenster für seine Ausgaben benutzt.

1. Öffnen Sie ein Konsolenfenster.

   Unter Windows heißt die Konsole Eingabeaufforderung oder auch MS-DOS-Eingabeaufforderung und kann je nach Betriebssystem über die Suche, das **(WIN)+(X)**-Menü, **Start/Alle Programme/Zubehör** oder **Start/Programme** aufgerufen werden.

2. Wechseln Sie in der Konsole in das Verzeichnis mit der ausführbaren Programmdatei.

   Zum Wechseln des Verzeichnisses gibt es die Befehle *cd Verzeichnis* und *..* (übergeordnetes Verzeichnis). Den Inhalt des aktuellen Verzeichnisses können Sie zur Kontrolle mit *dir* (Windows) oder *ls* (Linux) auflisten lassen. Das Laufwerk können Sie durch Eingabe des Laufwerksnamens wechseln (beispielsweise *c:*). Ein kleines Tutorium zur Bedienung der Windows-Konsole finden Sie im Übrigen unter *www.carpelibrum.de*.

3. Starten Sie das Programm, indem Sie den Programmnamen eintippen und abschicken (Enter).

Bild 2.10 Start eines Programms von der Windows-8-Konsole

Start aus Dateimanager oder über Verknüpfung

Sie können Konsolenanwendungen aber auch per Doppelklick aus einem Dateimanager (z. B. Windows Explorer) heraus aufrufen. Allerdings werden Sie dann unter Umständen nicht allzu viel von Ihrem Programm sehen. Dies liegt daran, dass das Programm seine Ausgabe in ein Konsolenfenster schreiben möchte. Wird das Programm nicht aus einem Konsolenfenster heraus aufgerufen, wird die Konsole für die Ausgabe automatisch vom Betriebssystem bereitgestellt. Sie wird aber auch automatisch geschlossen, wenn das Programm beendet ist. Da unser kleines Programm direkt nach der Ausgabe des „Hallo Welt"-Grußes schon beendet ist, sieht man das Konsolenfenster nur kurz aufflackern.

Damit das Programm auch sinnvoll aus einem Dateimanager oder über eine Verknüpfung aufgerufen werden kann, müssen Sie am Ende des Programms, jedoch vor der abschließenden `return`-Anweisung, die Zeile `cin.get();` einfügen:

```
/*****************************************************
 * Hallo Welt-Programm
 *
 * gibt einen Gruss auf den Bildschirm aus
 */

#include <iostream>
using namespace std;

int main()
{
   cout << "Hallo Welt!" << endl;

   cin.get();

   return 0;
}
```

Das Programm wartet dann, bis der Anwender die **(Enter)**-Taste drückt – und mit ihm wartet das vom Betriebssystem bereitgestellte Konsolenfenster.

2.3 Stil

Zum Abschluss noch ein Wort über guten und schlechten Stil.

C++ besitzt eine ziemlich kryptische Syntax, die daraus resultiert, dass es viele bedeutungstragende Syntaxelemente gibt, die durch einzelne Zeichen dargestellt werden ({, (, ++, %, . etc.), und dass sich die einzelnen Syntaxelemente zu den merkwürdigsten Konstrukten verbinden lassen.

Zwar kann man den sich daraus ergebenden Quelltexten eine gewisse asketische Eleganz kaum absprechen, doch trägt dies weder zur Lesbarkeit noch zur Wartbarkeit der Programme bei – was umso schwerer wiegt, als ein Tippfehler in nur einem Zeichen in C++ schnell zu einem katastrophalen Fehler führen kann.

Tun Sie also Ihr Bestes, um den Quelltext übersichtlich und gut lesbar zu gestalten:

- Schreiben Sie möglichst nur eine Anweisung in eine Zeile.
- Rücken Sie Anweisungsblöcke ein.
- Trennen Sie die einzelnen Elemente durch Leerzeichen.
- Kommentieren Sie Ihren Code.

Zwischen Namen und Schlüsselwörtern müssen Leerzeichen stehen. Sie können also nicht `intmain()` schreiben. Zwischen Namen und Operatoren müssen keine Leerzeichen stehen.

Mit einem Wort: Achten Sie also darauf, dass Ihre Quelltexte wie in Listing 2.1 aussehen und nicht etwa wie folgt:

```
#include <iostream>
using namespace std;
int main(){cout<<"Hallo Welt!"<<endl;cin.get();return 0;}
```

2.4 Übungen

1. Formulieren Sie das Hallo-Welt-Programm ohne die `using`-Anweisung zur Einbindung des `std`-Namensraums. (**Tipp:** Lesen Sie vielleicht noch einmal die Erläuterungen zu den Namensräumen in den Kapiteln 2.1.4 und 1.3.4.)
2. Schreiben Sie das Hallo-Welt-Programm so um, dass es Sie mit Ihrem Namen begrüßt.
3. Räumen Sie Ihre Projektverzeichnisse auf. Compiler legen meist mehr oder weniger umfangreiche Zwischendateien an, die nach Abschluss der Programmentwicklung nicht mehr benötigt werden und möglicherweise nicht automatisch gelöscht werden. Hierzu gehören grundsätzlich die *.obj*-Dateien und speziell für die Visual-Studio-Community-Edition auch die besonders umfangreichen *.idb-*, *.ilk-* und *.pdb*-Dateien (Letztere finden Sie im Verzeichnis der *.exe*-Datei). Zum Löschen dieser Dateien können Sie den Menübefehl **Erstellen/Projektmappe bereinigen** aufrufen. Werfen Sie danach auch noch einen Blick in das Projektverzeichnis. Gibt es dort eine *.sdf*-Datei und ein *ipch*-Verzeichnis? Dann löschen Sie diese.