Der (zumeist harmlose) Einstieg

1

In diesem Kapitel

- Eine extrem kurze und banale Geschichte über C
- Wie aus einer süßen kleinen Textdatei ein Programm wird
- Das erste C-Programm – eine alte Tradition
- Die notwendigen Künste von Editieren und Kompilieren
- Mit Fehlern auf Du und Du
- Wie die Sprache C aussieht
- Eingaben und Ausgaben (die hier mal was Gutes sind)
- Erst das Chaos, dann die Ordnung
- Wichtige C-Regeln, an die Sie nie denken werden

Willkommen bei den ersten Schritten. In diesem Kapitel finden Sie die Hintergrundinformationen, um mit der Programmierung in C beginnen zu können. Es ist nun Zeit für uns, den ersten mutigen Schritt zu wagen, an die Kante der Klippe der Angst zu treten und in die Fallen der Programmierung einzutauchen.

Bevor wir loslegen, würde ich Ihnen gerne noch versichern – vor allem wenn Sie noch nie Computer programmiert haben –, dass es Ihnen viel Spaß machen wird, dem Ding zu sagen, was es tun soll. Ich verspreche es. Dieses Buch wird Sie niemals überfordern, weil ich nur ein gewöhnlicher Sterblicher bin und kein C-Gott.

Eine extrem kurze und banale Geschichte über C

Am Anfang war die Sprache B. Danach gab es die Sprache C.

Nein, ich bin nicht ausgeflippt. C wurde bei den AT&T Bell Labs in den frühen Siebzigern entwickelt. Zu dieser Zeit gab es in den Bell Labs eine Programmiersprache mit dem Namen B – B für Bell. Als die nächste Sprache entwickelt wurde, ging man einen Schritt weiter: C kommt nach B.

Der Typ, der C entwickelte, heißt Dennis Ritchie. Ich erwähne das nur für den Fall, dass Sie mal durch die Straßen laufen und plötzlich in Herrn Ritchie reinlaufen. In diesem Fall können Sie sagen: »He, sind Sie nicht Dennis Ritchie? Der Mann, der C erfunden hat?« Und er wird sagen: »Warum – äh – ja, das stimmt.« Und Sie können sagen: »Cool.«

Sachen, die Sie nicht über Hochsprachen wissen müssen

Programmiersprachen haben verschiedene Stufen. Diese Stufen beschreiben, wie schwierig oder wie einfach die Sprache zu erlernen ist. Am schwierigsten sind die Low-Level-Programmiersprachen – Hochsprachen sind einfacher und erinnern ein bisschen an Englisch. C ist eine Sprache in der Mitte, irgendwo zwischen exakter englischer Schreibweise und Grunzen und Gestikulieren.

Die niedrigste der Low-Level-Programmiersprachen ist die *Maschinensprache*. Das sind das primitive Grunzen und das Gestikulieren des Mikroprozessors. Maschinensprache besteht hauptsächlich aus Zahlen und Symbolen, die der Mikroprozessor versteht und ausführt. Der Vorteil von einem Programm in Maschinensprache ist, dass es schneller läuft als jedes andere Programm, das in einer höheren Sprache geschrieben ist. Und diese Programme sind winzig klein.

Assembler ist der Cousin der Maschinensprache. Sie sind weitgehend identisch, nur verwendet man in Assembler abgekürzte englische Begriffe statt Zahlen. Dies wird immer noch als Low-Level-Sprache bezeichnet und kann von Menschen nur schwer erlernt und beherrscht werden. Und obwohl Assembler-Programme schneller sind als alles andere, dauert es doch Ewigkeiten, bis die Programme fertig sind. Na ja, zumindest eine kleine Ewigkeit.

C ist eine *Mid-Level-Sprache*. Teile davon erinnern an Assembler – viele C-Compiler erlauben sogar, dass man mitten im C-Programm Assembler-Anweisungen einfügt. Andere Teile von C sind eher Hochsprache, praktisch unmittelbar verständlich. C hat den Vorteil, dass es sowohl schnell ist, aber auch zu großen Teilen noch ein bisschen an Englisch erinnert.

Hochsprachen umfassen sowohl die populären Sprachen BASIC oder Java als auch andere Sprachen, die zum Teil heute nicht mehr so verbreitet sind. BASIC liest sich fast wie Englisch, alle Kommandos und Anweisungen sind englische Wörter – oder zumindest englische Wörter, in denen einige Vokale fehlen oder die ein bisschen durch die Schreibweise verunstaltet wurden.

Obwohl Hochsprachen sehr einfach zu lesen und von Menschen zu verstehen sind, laufen die Programme damit langsamer als ihre Gegenstücke in C oder Assembler. Außerdem neigen die Programme dazu, größer zu sein.

Wie aus einer süßen kleinen Textdatei ein Programm wird

Wenn Sie ein Programm schreiben, werden Sie zum Programmierer. Ich weiß, der Begriff ist neu für Sie. Ihre Verwandten oder Bekannten werden Sie weiterhin als »Computernutzer« einstufen, obwohl Ihre Bindung an die Maschine so eng wie nur möglich wurde. Also hängen Sie sich den Titel um den Hals und stolzieren Sie mit geschwellter Brust umher.

Ahh …

Nachdem Sie nun Programmierer sind, worin besteht Ihr Job? Ganz offensichtlich: Programme schreiben. Aber Programme sollten einen Zweck haben. Der Zweck ist, dass der Computer etwas tut.

Das Ziel der Programmierung ist, dass etwas »geschieht«. Die Sprache C ist nur ein Werkzeug zur Kommunikation mit dem PC.

Es ist Ihre Aufgabe als Programmierer, die Wünsche des Nutzers in etwas zu übersetzen, das der Computer versteht und dem Nutzer das liefert, was er will. Und falls Sie dazu nicht in der Lage sind, sollte es doch so nahe dran sein, dass sich die Leute nicht ständig beschweren und –schlimmer – ihr Geld zurückverlangen.

- ✔ Ich könnte eine Menge philosophischer Ratschläge geben, wie man benutzerfreundliche Programme schreibt und daran denkt, dass der Nutzer nicht so besonders clever ist. Aber was soll's.
- ✔ Denken Sie daran, wer das Programm nutzt, und stellen Sie sicher, dass deren Bedürfnisse getroffen werden. Die Erfüllung von Bedürfnissen dagegen ist was für Fortgeschrittene.

Der Entwicklungsprozess in C

Hier nun, wie man in neun Schritten ein C-Programm schreibt – dies nennt sich *Entwicklungsprozess*:

1. Lassen Sie sich eine Idee für ein Programm einfallen.
2. Nehmen Sie den Editor und schreiben Sie den Quelltext.
3. Kompilieren Sie das Programm mit einem C-Compiler.
4. Weinen Sie bitterlich über Fehler (kann auch entfallen).
5. Linken Sie das Programm mit einem Linker.
6. Weinen Sie bitterlich über Fehler (kann auch entfallen).
7. Starten Sie das Programm und beginnen Sie mit Tests.
8. Raufen Sie sich die Haare, wenn schwere Fehler auftreten (kann auch entfallen).
9. Beginnen Sie von vorn (immer notwendig).

Es ist nicht erforderlich, dass Sie das auswendig können. Das ist wie mit der Anleitung auf einer Flasche mit Haarshampoo (die gibt es dort wirklich – wenn Ihnen mal langweilig ist wie mir neulich, dann können Sie mangels anderen Lesestoffs in der Badewanne die Anleitung lesen). Keiner liest so was, weil wir alle auch so wissen, wie man mit Shampoo umgeht. Sie werden bald ganz oft diese Schritte durchlaufen, sodass Sie sich das nicht merken müssen.

(Dennoch ist Wissen über den »Entwicklungsprozess« nützlich. Sie können bei Ihren Freunden lamentieren, dass Sie mitten in einem Entwicklungsprozess eines C-Programms stecken.

Schauen Sie mal, wie sich deren Augen weiten, wenn sie mit Ihrem Wissen über Programmierung konfrontiert werden!)

- ✔ Schritt 1 ist am härtesten. Der Rest folgt dann von selbst.
- ✔ Dies wird in ein paar Kapiteln wirklich so einfach wie Haare waschen werden.

Der Quelltext (oder auch Sourcecode)

Wenn Sie ein Programm erstellen, sagen Sie dem Computer, was zu tun ist. Da der Computer Sprache nicht verstehen kann und ihm Schläge – egal wie gut Sie sich dabei fühlen – nicht viel ausmachen, kommunizieren Sie letztendlich mit dem Computer, indem Sie eine Mitteilung schreiben – eine Datei auf Festplatte.

Um die Mitteilung zu schreiben, benutzen Sie ein Programm namens *Texteditor*. Das ist die primitive Version einer Textverarbeitung, ohne all die Formatierungen und Ausdruckvarianten. Der Texteditor lässt Sie Text eintippen – das ist schon alles.

Wenn Sie Ihren Texteditor benutzen, erzeugen Sie eine Datei, die man *Quelltext* oder auch *Sourcecode* nennt. Das Besondere an dieser Datei ist, dass diese Datei Instruktionen für den Computer enthält, die diesem bestimmte Aufträge erteilen. Und obwohl es angenehm wäre, wenn man Programme in der Form »mach mal 'n lustiges Geräusch« schreiben könnte, so muss man doch eine Formulierung benutzen, die der Computer versteht. In unserem Fall werden die Anweisungen in der Sprache C geschrieben.

Wenn Sie mit dem Schreiben der Anweisungen fertig sind, speichern Sie diese in einer Datei. Benennen Sie den ersten Teil des Dateinamens mit dem Namen, den das fertige Programm später bekommen soll. Wenn Sie zum Beispiel ein Spiel mit dem Namen »UFOs« entwickeln, sollte die Datei mit dem Quelltext im ersten Teil vor dem Punkt den Namen `Ufos` erhalten.

Der zweite Teil des Dateinamens, die Extension oder Dateinamenerweiterung, muss `c` sein, für die Sprache C. Das ist wichtig! Die meisten Textdateien enden auf `txt` oder manchmal `doc`. Für die Sprache C müssen die Dateien mit `c` enden, hier also `Ufos.c`.

- ✔ Der Quelltext ist eine Textdatei auf Festplatte. Er enthält Anweisungen für den Computer, die in der Sprache C geschrieben sind.
- ✔ Sie benutzen einen Texteditor, um den Quelltext zu erzeugen. Die meisten C-Compiler kommen mit ihren eigenen Texteditoren daher. Falls dies für Ihren nicht zutrifft, können Sie einen anderen Texteditor verwenden (manche Programmierer tun dies ohnehin viel lieber).

Es ist möglich, eine Textverarbeitung für die Erstellung von Programmen zu verwenden. Wichtig ist, dass Sie die Datei als »Nur Text« oder »DOS-Text« oder »ASCII« oder »unformatierten Text« speichern (eine Textverarbeitung für Quelltexterstellung zu benutzen, ist ein bisschen wie mit der 747 zur Arbeit zu fliegen – viel Leistung, aber schlecht zu parken).

- ✔ Die Datei mit dem Quelltext endet mit einem `c` als Dateinamenerweiterung.

- ✔ Der erste Teil des Dateinamens sollte der Name des Programms sein.
- ✔ Seien Sie ein bisschen clever beim Benennen Ihrer Dateien.

Der Compiler

Nachdem der Quelltext getippt und gespeichert wurde, muss er in eine Sprache übersetzt werden, die der Computer verstehen kann. Dieser Job wird vom Compiler erledigt.

Der *Compiler* ist ein spezielles Programm, das die Anweisungen in der Quelltextdatei liest. Der Compiler nimmt sich jede Anweisung vor und übersetzt sie in einen unverständlichen Code, der nur vom Mikroprozessor verstanden wird.

Falls alles glattgeht und der Compiler mit Ihrem Quelltext zufrieden ist, erzeugt er eine *Objektcode*-Datei, eine weitere Datei, die gespeichert wird. Die Objektdatei hat den gleichen Namen wie die Quelltextdatei, endet aber auf `.obj`. Für das UFO-Spiel wäre dies also `Ufos.obj`.

Versteht der Compiler irgendetwas nicht, zeigt er eine Fehlermeldung auf dem Bildschirm an. An diesem Punkt können Sie mit den Zähnen knirschen und wütend aufspringen. Gehen Sie zurück zum Texteditor und ändern Sie den Quelltext so, dass der angemahnte Fehler beseitigt ist (das ist leichter, als es klingt). Danach versuchen Sie erneut, den Quelltext zu *kompilieren*.

Nachdem der Compiler seine Arbeit getan hat, ist das Programm fertig. Nun wird ein dritter Schritt benötigt, das Linken. Dieses Thema wird gleich im Anschluss im Abschnitt »Der Linker« behandelt.

- ✔ Der Compiler nimmt die Informationen im Quelltext und übersetzt sie in Anweisungen, die der Computer versteht. Das Resultat ist eine neue Datei – die Objektdatei.
- ✔ Die Objektdatei endet mit `obj`. Der erste Teil des Namens ist genauso wie der Name der Datei mit dem Quelltext.
- ✔ Übrigens: Alle diese Dateien werden im gleichen Verzeichnis gespeichert. Wie man hier ein wenig aufräumt, dazu kommen wir weiter hinten in diesem Kapitel im Abschnitt »Erst das Chaos, dann die Ordnung«.
- ✔ Fehler passieren. Wenn der Compiler etwas findet, was er nicht versteht, beendet er die Kompilierung und erzeugt eine Fehlermeldung auf dem Bildschirm. Obwohl das bedrohlich klingt, sind diese Fehler in der Regel einfach zu beheben.
- ✔ Einige der C-Größen nennen den Inhalt einer Objektdatei auch *Objektcode*.

Neugierig? Nicht notwendige Informationen über Objektdateien

Sind Objektdateien notwendig? Nein, nicht wirklich. Sie sind ein Zwischenschritt. Von Ihrem Quelltext erzeugt der Compiler Anweisungen in Maschinensprache, die für den Mikroprozessor direkt verständlich sind. Eigentlich könnte an dieser Stelle das endgültige Programm erzeugt werden, aber stattdessen wird eine Objektdatei erzeugt.

Der Grund für die Objektdateien liegt im Verwalten von großen Programmierprojekten. In diesen Fällen arbeitet man an vielen Quelltextdateien. Warum? Zunächst mal, damit das Programm handhabbar bleibt. Zum Beispiel kann Ihre Textverarbeitung die 500 Seiten eines Romans als eine einzige Datei verwalten, aber es ist einfacher, wenn man auf der Basis von Kapiteln arbeitet. Für große Programme macht es Sinn, diese in kleinere Portionen oder auch *Module* aufzuteilen. Jedes Modul wird separat kompiliert und verschiedene Objektdateien werden erzeugt. Es ist dann Aufgabe des Linkers, diese separaten `obj`-Dateien in eine einzige ausführbare `exe`-Datei zusammenzubinden.

Zum jetzigen Zeitpunkt Ihrer C-Karriere sind die Programme zu klein, um sich Gedanken über mehrere Objektdateien zu machen. Aber der Linker ist dennoch notwendig, um die eine Objektdatei in das endgültige Programm zu verwandeln.

Der Linker

Der *Linker* ist ein Programm, genau wie der Compiler. Seine Aufgabe besteht in der Erzeugung fertiger Programme.

Der Linker tut Folgendes: Er nimmt die vom Compiler erzeugte `obj`-Datei, macht sie ein wenig hübsch und erzeugt die fertige Programmdatei. Diese Datei endet auf `exe`, was der übliche Name für Programme unter DOS oder Windows ist.

Der erste Teil des Namens ist wieder genauso wie der erste Teil der Quelltextdatei, wenn man also mit `Ufos.c` startet, erhält man vom Compiler `Ufos.obj` und später vom Linker `Ufos.exe`.

- ✔ In den meisten C-Compilern werden die Arbeiten von Compiler und Linker in einem Durchgang nacheinander erledigt. Teilweise sieht man »compiling« und »linking« als zwei Schritte aufgelistet (was sie ja auch sind), aber in Ihrem C-Compiler läuft dies zusammen ab.
- ✔ Wie der Compiler meldet auch der Linker Fehler, wenn er irgendetwas nicht versteht. In diesem Fall muss man die Fehlermeldung entziffern und das Programm neu kompilieren.
- ✔ Texteditor → Compiler → Linker
- ✔ Quelltext → Objektcode → Programm

Richtig! Obwohl wir mit einer Datei anfangen, haben wir zum Schluss mindestens drei Dateien auf der Festplatte: `Ufos.c`, `Ufos.obj` und `Ufos.exe`. Manche Compiler überschwemmen die Platte sogar mit noch mehr Dateien. Wie man Ordnung schafft, sehen wir am Ende dieses Kapitels.

Das erste C-Programm – eine alte Tradition

Seit den alten Römern enthält jedes Buch über Programmierung das traditionell erste C-Programm. Dieses Buch folgt der Regel, obwohl das *C für Dummies*-Programm mehr Selbstbewusstsein hat als andere Bücher über C. Also:

```
#include <stdio.h>

int main()
{
   printf("Ade, grausame Welt! ");
   return 0;
}
```

Listing 1.1: Name: `Goodbye.c`

Huch! Treten Sie einen Schritt zurück und blinzeln Sie erst einmal! Aber gewöhnen Sie sich daran, so werden die Programme in dem Buch vorgestellt.

Sie sollten sofort das Offensichtliche sehen. Was kommt Ihnen bekannt vor? Ja, Buchstaben aus dem Alphabet, a bis z, aber meist kleingeschrieben. Und einige der Zeichen haben Sie auch schon mal auf der Tastatur gesehen, die runden Klammern `()` und die geschweiften Klammern `{ }`. Bis auf `printf`, `int` und das komische sollten Ihnen die anderen Wörter bekannt vorkommen (jedenfalls wenn Sie ein wenig Englisch verstehen):

```
include
int
main
return
"Ade, grausame Welt!"
```

Das ist gut – bisher sieht C einfach schlecht aus.

Ihre Aufgabe lautet: Nehmen Sie den monströsen Text und verwandeln Sie ihn in ein Programm, das die folgende Zeile auf dem Bildschirm ausgibt:

```
Ade, grausame Welt!
```

- ✔ Coole Leute sagen nicht »sie programmieren«, sondern sie »coden«. Weichlinge programmieren, Freaks *coden*.
- ✔ Der Abschnitt »Wie die Sprache C aussieht« weiter hinten in diesem Kapitel behandelt die Details von `Goodbye.c`, wie es funktioniert und die wundervollen Kleinigkeiten darin. Schauen Sie nach, wenn Sie bereits jetzt mehr dazu wissen wollen.

In *C für Dummies* ist das erste Programm `Goodbye` und es zeigt die Meldung `Ade, grausame Welt!` an. Andere Bücher über C – eigentlich alle – nennen ihr erstes Programm `Hallo`. Es zeigt die tolle Meldung `Hallo Welt!` an. Ich schätze mal, das soll zeigen, wie lustig und sympathisch C ist. Oder die C-Programmierer glauben, dass die ganze Welt vor den Bildschirmen hockt und dies deshalb eine gelungene

Begrüßung ist. Mein Vorschlag: Leute, schreibt mal ein Programm, das `Hallo Ihr Introvertierten, geht mal raus und lernt ein paar Maedels kennen!` auf dem Bildschirm erscheinen lässt. (Ja, so was werden Sie bald programmieren können – weiterlesen.)

Spickzettel für die Eingabe des Programms `Goodbye.c`

Beginnend mit der ersten Zeile, geben Sie Folgendes ein:

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

Geben Sie das Gatter `#` (auch als Nummernzeichen bekannt), das Wort `include`, ein Leerzeichen, eine öffnende spitze Klammer `<`, das Wort `stdio`, einen Punkt, ein `h` und eine schließende spitze Klammer `>` ein. Verwenden Sie keine Großbuchstaben. Drücken Sie zweimal ↵, um die Zeile zu beenden und eine leere Folgezeile zu erhalten.

Geben Sie das Wort `int`, ein Leerzeichen und dann gleich `main`, gefolgt von einer öffnenden runden Klammer `(` und einer schließenden runden Klammer `)` ein. Drücken Sie ↵, um die Zeile zu beenden.

```
int main()
```

Die nächste Zeile enthält nur ein Zeichen: eine öffnende geschweifte Klammer `{`. Drücken Sie ↵.

```
{
```

Drücken Sie die ⇆-Taste, um die nächste Zeile einzurücken, falls Ihr Editor dies nicht automatisch macht:

```
    printf("Ade, grausame Welt! ");
```

Geben Sie `printf` (als ein Wort), gefolgt von einer öffnenden runden Klammer `(` und einem Anführungszeichen `"` ein. Und dann tippen Sie `Ade, grausame Welt!`. Geben Sie einen Backslash und ein kleines `n`, dann wieder ein Anführungszeichnen `"` und eine schließende runde Klammer`)` ein. Danach geben Sie ein Semikolon `;` ein und drücken ↵.

Jetzt fehlt noch etwas: Tippen Sie `return`, gefolgt von einer `0` und einem Semikolon `;` – kein Komma, sondern ein Semikolon. Danach drücken Sie wieder die Taste ↵.

```
    return 0;
```

Beginnen Sie die nächste Zeile, indem Sie die Taste Pos1 drücken und damit den Cursor wieder an den Anfang der Zeile zurücksetzen. Geben Sie eine schließende geschweifte Klammer `}` ein und drücken Sie danach ↵.

```
}
```

Speichern! Kompilieren! Linken! Starten!

Vier Schritte sind notwendig, um ein C-Programm zu erstellen. Es handelt sich um *Speichern*, *Kompilieren*, *Linken* und *Starten*. Die meisten C-Umgebungen erledigen das Linken automatisch, manche sogar auch das Starten. Egal ob es automatisch abläuft oder nicht – es ist vorhanden.

Speichern bedeutet, den Quelltext zu speichern. Man erstellt den Quelltext in einem Texteditor und speichert ihn als Textdatei mit der Endung `c`.

Kompilierung ist der Vorgang, bei dem die Anweisungen im Quelltext vom Compiler in etwas übersetzt werden, das der Mikroprozessor des Computers verstehen kann.

Linken ist der Schritt, bei dem alle Anweisungen zu einem Programm zusammengepackt werden. (Zur Erinnerung: Das geschieht oftmals automatisch.)

Zum Schluss *starten* Sie das so erzeugte Programm. Ja richtig, es ist ein echtes Programm, genauso wie andere auf der Festplatte.

- ✔ Das Programm, das übersetzt, heißt *Compiler*. Das Programm, das zusammensetzt heißt *Linker*.
- ✔ In integrierten Entwicklungsumgebungen gibt es häufig einen Menübefehl für diesen Vorgang. Dieser Befehl heißt dann zum Beispiel ALLES NEU ERSTELLEN oder ERSTELLEN und befindet sich im Menü Projekt.

Wenn Sie mit einem Compiler unter Windows arbeiten, passiert Ihnen oft Folgendes, wenn das Programm automatisch von der Entwicklungsumgebung gestartet wird: Ein Fenster geht kurz auf und wieder zu. Fertig. Wenn Sie ganz schnell sind (oder Ihr Computer ganz langsam ist), können Sie sehen, dass in diesem Fenster Ihr Programm lief. Das ist unbefriedigend. Sie werden sich später noch Tricks erarbeiten, wie Sie das Fenster länger sichtbar machen können, zunächst wenden Sie folgenden Trick an: Rufen Sie unter Windows im Startmenü den Befehl Ausführen auf und geben Sie `cmd` ein. Ein Kommandozeilenfenster (auch als Eingabeaufforderung bekannt) öffnet sich. Wechseln Sie mithilfe von `cd` in das Verzeichnis, in dem Ihr Programm gespeichert ist, und starten Sie es durch Eingabe von `Goodbye` und ↵. Voilà! Wie Sie sehen, bleibt die Ausgabe nun erhalten. Sie können das Fenster während der Arbeit offen halten und das Programm hierüber immer wieder starten.

Die notwendigen Editier- und Kompilierkünste

Beim ersten Mal wird nicht gleich alles in Ihrem C-Programm glattgehen. Schnell hat sich ein Tippfehler in den Quelltext eingeschlichen, ein Beispiel: `printf` wird `prinft` oder nur `print`. Falls das passiert, spuckt der Compiler einen üblen Fehler aus und das Programm wird nicht erzeugt. Oder anstelle von `Welt` schreibt das Programm `Wetl`. Diese Art Missgeschick ist kein wirklicher Fehler. Es ist eher ein bisschen peinlich, dennoch, auch so was muss irgendwie repariert werden.

Um einen Fehler zu beheben, müssen Sie üblicherweise zwei Schritte erledigen:

1. Den Quelltext neu editieren und wieder speichern
2. Den Quelltext neu kompilieren und das reparierte Programm erzeugen

Das sind immer noch Schritte aus dem C-Entwicklungsprozess, den wir schon kennenlernten. Als Programmierer versuchen Sie, so etwas nach Möglichkeit zu vermeiden. Im wirklichen Leben passiert es leider viel zu oft.

- ✔ Es passiert.
- ✔ Wie man Fehler behebt, wird besonders im nächsten Abschnitt behandelt. Dieses Kapitel erklärt Ihnen die notwendigen Schritte zur Reparatur des Quelltextes und zur Neuerstellung des Programms.

Die Quelltextdatei ändern

Ihr Quelltext ist nicht in Stein gemeißelt – oder in Silizium, was hier besser passt. Er kann geändert werden. Manchmal sind die Änderungen notwendig, nämlich im Fehlerfall und im Falle von peinlichen Tippfehlern und Ähnlichem. Es ist auch denkbar, dass Sie einfach Ihr Programm ändern wollen, vielleicht eine Kleinigkeit hinzufügen oder eine Meldung ergänzen, eben das, was man so *Feintuning* nennt. In diesem Fall muss man seinen Quellcode ändern.

Das `Goodbye`-Programm zeigt eine Meldung auf dem Bildschirm an: `Ade, grausame Welt!` Genau genommen können Sie es jede beliebige Nachricht anzeigen lassen. Hierzu benutzen Sie Ihren Texteditor und ändern Sie den Sourcecode, sodass er eine andere Nachricht darstellt.

Sie sollten nun dies tun: Ändern Sie mit dem Texteditor `Goodbye.c`. Falls Sie eine integrierte Entwicklungsumgebung benutzen, klicken Sie auf das Fenster, das den Quelltext von `Goodbye.c` enthält.

Nachdem Sie im Editor sind und den Quelltext sehen können, kann die dargestellte Nachricht geändert werden. Sie müssen nur die folgende Zeile ändern:

```
printf("Ade, grausame Welt!");
```

Ersetzen Sie den Text `Ade, grausame Welt!` – und nur diesen Teil der Zeile – durch einen anderen Text. Vermischen Sie das nicht mit dem komischen oder irgendwas anderem außerhalb der Anführungszeichen.

Zum Beispiel könnten wir den Text durch `Ich komme wieder!` ersetzen. Ändern Sie den Quelltext, sodass er wie folgt aussieht:

```
printf("Ich komme wieder!");
```

Diese Modifikation ändert die Ausgabe des Programms in `Ich komme wieder!` Wenn Ihr Computer dies oft genug sagt, glauben Sie noch, Sie wären im Kino.

Wenn Sie mit dem Editieren fertig sind, prüfen Sie Ihre Arbeit zweimal. Speichern Sie die Datei wieder auf Platte. Nun können Sie Ihren Quelltext neu übersetzen, wie es im nächsten Abschnitt behandelt wird.

- ✔ »Ändern Ihrer Quelltextdatei« bedeutet, Ihren Texteditor zu benutzen, um den Quelltext zu ändern, nämlich die Datei, die die Anweisungen in der Sprache C enthält.
- ✔ Sie werden Ihren Quelltext ändern müssen, um Fehler zu beseitigen, die der Compiler oder der Linker entdeckt haben, oder um etwas zu ändern. Dies passiert oft.

In den integrierten Entwicklungsumgebungen erscheint Ihr Quelltext in einem Editorfenster. Klicken Sie auf dieses Fenster und führen Sie die notwendigen Änderungen durch.

- ✔ Wenn die Änderungen durchgeführt sind, speichern Sie die Datei wieder.
- ✔ Das Speichern der Datei überschreibt das Original (`Goodbye.c` würde also durch eine neuere Version ersetzt). Denken Sie also daran, dass Sie gegebenenfalls die Datei unter einem anderen Namen abspeichern, wenn Sie das Original aufheben wollen.

Neukompilierung (oder: Spiel's noch mal in der Sprache C)

Neukompilierung bedeutet, das Programm erneut zu erstellen – also die Schritte für die erste Erzeugung noch einmal zu wiederholen. Dies geschieht üblicherweise nach dem Ändern des Quelltextes, wie wir es gerade eben getan haben. Weil der Quelltext anders ist, müssen Sie den Compiler neu damit füttern, um ein neues, besseres (und hoffentlich nun fehlerfreies) Programm zu erzeugen.

Für die Neukompilierung gehen Sie so vor, wie zuvor bei der Kompilierung beschrieben.

- ✔ Nachdem Sie die Quelltextdatei geändert haben, müssen Sie Ihre Programmdatei neu erstellen. Auf diese Weise werden Fehler beseitigt oder Programme geändert.
- ✔ Einige Compiler benötigen auch ein erneutes Linken, andere linken automatisch neu; dies ist Ihnen schon bei der ersten Programmerstellung begegnet.
- ✔ Wenn Sie Fehler nach dem Neukompilieren entdecken, müssen Sie erneut eine Änderung der Änderung durchführen und anschließend neu neukompilieren (na ja, das könnte man so fortsetzen, aber man spricht nur von Neukompilierung).

»Aber mein Compiler kompiliert's nicht neu!«

Dies ist ein seltener und seltsamer Zustand, aber ich habe das schon bei einigen Compilern gesehen. Notwendigerweise schaut der Compiler nach den Dateien auf der Festplatte, aber er sieht keine Änderungen – obwohl Sie Ihren Quelltext geändert und neu gespeichert haben. Dieses Problem kann man auf zwei Arten lösen:

- ✔ Erstens: Speichern Sie einfach noch einmal. Das wird häufig schon ausreichen.
- ✔ Zweitens: Rufen Sie den Befehl ALLES NEU ERSTELLEN (oder so ähnlich, die Befehlsbezeichnungen unterscheiden sich je nach Umgebung) im Menü Projekt auf. Das Programm wird nun vollständig mit allen Dateien neu übersetzt.

Raus mit den alten Sachen, lasst uns was Neues machen

Änderungen und Neukompilierung sind Alltag, wenn man C programmiert. Man beachte: Es kann sich um *viele* Änderungen und Neukompilierungen handeln. Aber erzählen Sie keinem Ihrer Freunde davon und behaupten Sie, dass alle Ihre C-Programme auf Anhieb funktionieren.

Gelegentlich wollen Sie einfach neu anfangen. Dazu müssen Sie Ihren Texteditor leeren und sozusagen auf einem leeren Blatt Papier – oder hier besser einem leeren Bildschirm – neu mit der Arbeit beginnen. Sie können dann ein neues Programm erstellen, kompilieren, linken, starten und erhalten ein neues und wunderbares Kunstwerk, indem Sie den Computer Ihrem Willen unterwerfen, nur bewaffnet mit Ihrer Kenntnis der Sprache C (okay, vielleicht ein paar Kapitel später).

Um ein neues Programm in einer integrierten Entwicklungsumgebung zu erzeugen, wählen Sie im Menü Datei den Befehl Neu. Dadurch erhalten Sie ein neues, leeres Fenster, in dem Sie Ihr nächstes Programm niederschreiben können.

- ✔ Ihre alten Projekte bleiben in der Entwicklungsumgebung erhalten, bis Sie deren Fenster schließen.
- ✔ Arbeiten Sie unter der Kommandoebene, verwenden Sie Ihren Editor, um eine neue Datei zu erzeugen. Hier reicht es normalerweise aus, einfach den Editor zu starten, er beginnt dann von selbst mit einem neuen, leeren Fenster.

Mit Fehlern auf Du und Du

Fehler passieren. Sogar den besten Entwicklern unterlaufen Fehler. Die Jungs, die Word, Excel und die ganzen anderen bekannten Softwarepakete geschrieben haben? Jeden Tag Fehler. Bill Gates? Fehler über Fehler. Steve Jobs? Lacht schon mal los, während wir noch die Fehler zählen.

Fehler sind nichts, worüber man sich aufregen muss. Und das Beste ist, dass der Compiler Ihnen sagt – mit unbarmherziger Genauigkeit –, was und wo der Fehler ist. Denken Sie mal an die Alpträume Ihrer Schulzeit in Mathematik: Da hat einfach einer »FALSCH!« neben Ihr Ergebnis geschrieben, egal wie klein der Fehler war. Ja, Computer können vergeben – und dies wird sehr hilfreich sein.

Mist! Ein Fehler. Aber bevor Sie aus dem Fenster springen …

Hier kommt ein neues Programm, `Error.c`. Nicht gerade der optimistischste Name. Dies ist ein »verseuchtes« Programm, eines, das einen Fehler enthält:

```
#include <stdio.h>

int main()
```

```
{
   printf("Dieses Programm hat einen Fehler ")
   return 0;
}
```

Listing 1.2: Name: `Error.c`

Tippen Sie zunächst den Quellcode in Ihrem Texteditor ein, und zwar genau so, wie er da steht. Speichern Sie danach den Quellcode in der Datei `Error.c`.

Erzeugen Sie nun das Programm `Error`; dies geschieht wieder durch Kompilieren und Linken, wie es weiter vorn in diesem Kapitel beschrieben wurde. Schauen Sie dort noch einmal nach, falls Sie Hilfe brauchen.

Unglücklicherweise wird bei der Kompilierung des Programms ein Fehler erzeugt. Der nächste Abschnitt »Junge, was'n Fehler!« beschäftigt sich mit der Autopsie des Fehlers.

- ✔ Sie können das Fenster mit `Goodbye.c` nun schließen, wenn Sie wollen. Wir werden das Programm in diesem Buch nicht mehr benutzen, aber Sie können es jederzeit zu Demonstrationszwecken laufen lassen, um Ihre Freunde zu beeindrucken.
- ✔ Bitte nutzen Sie die Spickzettel, wenn Sie bei der Eingabe von Programmen Hilfe benötigen. Diese Abschnitte gibt es im Buch nur, falls Sie zusätzliche Hilfe beim Eintippen benötigen. Spätere Kapitel enthalten Beschreibungen, wie man die Programme laufen lässt, mit ein paar Infos zur Eingabe, aber nicht mehr so ausführlich wie im ersten Kapitel.

Spickzettel für die Eingabe von `Error.c`

Beginnend mit der ersten Zeile, geben Sie ein:

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

Schreiben Sie das Gatter #, das Wort `include`, ein Leerzeichen, eine öffnende spitze Klammer, das Wort `stdio`, einen Punkt, ein `h` und eine schließende spitze Klammer. Verwenden Sie keine Großbuchstaben. Drücken Sie zweimal ↵, um die Zeile zu beenden und eine leere Folgezeile zu erhalten.

Schreiben Sie das Wort `int`, ein Leerzeichen und dann gleich `main`, gefolgt von einer öffnenden runden Klammer und einer schließenden runden Klammer. Drücken Sie ↵, um die Zeile zu beenden.

```
int main()
```

Die nächste Zeile enthält nur ein Zeichen: eine öffnende geschweifte Klammer {. Drücken Sie ↵.

```
{
```

Drücken Sie die ⇥-Taste, um die nächste Zeile einzurücken, falls Ihr Editor dies nicht automatisch macht:

```
    printf("Dieses Programm hat einen Fehler ")
```

Geben Sie `printf` (als ein Wort), gefolgt von einer öffnenden runden Klammer und einem Anführungszeichen ein und dann `Dieses Programm hat einen Fehler!`. Geben Sie einen Backslash und ein kleines `n` ein, dann wieder Anführungszeichen und eine schließende runde Klammer. Drücken Sie anschließend ↵.

In der nächsten Zeile drücken Sie zunächst Tab, falls der Editor den Zeilenanfang nicht von selbst richtig einrückt, und danach geben Sie wieder `return` gefolgt von der `0` und einem Semikolon ein.

Beginnen Sie die nächste Zeile, indem Sie Pos1 drücken und damit den Cursor wieder an den Anfang der Zeile zurücksetzen. Geben Sie eine schließende geschweifte Klammer `}` ein und drücken Sie danach Enter.

```
}
```

Im ganzen Buch werden Sie angewiesen, den Quelltext einzugeben, zu speichern, zu kompilieren, zu linken und zu starten. Diese Anweisungsfolge schreibe ich ab jetzt kürzer: »Kompilieren Sie das Programm« oder »Starten Sie das Programm«. Dies ist entschuldbar, weil ich nicht jedes Mal die ganze Folge wiederholen kann. Außerdem werden Sie mit der Zeit so vertraut damit, sodass Sie diese Wiederholungen auch gar nicht brauchen.

Junge, was'n Fehler!

Fehler in `Error.c`! Tauchalarm! Schockzustand!

Okay, wir haben's ja erwartet. (Um genau zu sein: Vielleicht haben Sie den Fehler ja schon vorher einmal ungewollt gesehen.) Abhängig vom verwendeten Compiler wird in irgendeiner Form folgender Fehler erscheinen:

```
...error.c(6): Syntaxfehler: fehlendes ';' vor '}'
```

Es können auch noch andere Meldungen angezeigt werden, aber die Zeile 6 wird irgendwo in der Meldung so oder so ähnlich vorkommen.

Die Fehlermeldung untersuchen

Bisher sieht die Fehlermeldung ziemlich rüde aus. Aber was ihr an Freundlichkeit fehlt, enthält sie dafür an Informationen. Egal welchen Compiler Sie verwenden, Sie sollten in der Meldung folgende Informationen erhalten:

- ✔ Das Programm, das den Fehler enthält: `Error.c`
- ✔ Die Zeile, in der der Fehler steht, hier also Zeile 6 (aber trauen Sie der Angabe nicht zu sehr – der Fehler kann auch eine Zeile höher oder tiefer sein)
- ✔ Der Typ des Fehlers, hier also ein Syntaxfehler (es wurde also etwas falsch geschrieben)

✔ Der Ort des Fehlers, also vor der }

Es mag noch nicht ganz klar sein, was falsch ist, aber man bekommt eine Menge Hinweise. Am wichtigsten die Zeilennummer:

```
...error.c(6):
```

Nun gut, eigentlich ist der Fehler ja in Zeile 5, aber die Sprache C ist flexibel und der Compiler kann vor der Zeile 6 nicht entdecken, dass etwas in Zeile 5 fehlt.

Der Fehlertyp ist genauso wichtig. Ein Syntaxfehler bedeutet Tippfehler, der Begriff Syntax bezeichnet die Art und Weise, wie Sprachen geschrieben werden. Auf Englisch heißt das übrigens *syntax error*. In der Sprache C ist ein Semikolon ; am Ende eines jeden Satzes notwendig, genauso wie Sie am Ende eines Satzes einen Punkt machen.

Unter Umständen sagt Ihr Compiler Ihnen das sogar:

```
fehlendes ';'
```

Die Lösung? Ändern Sie Ihren Quelltext und beseitigen Sie den Fehler, indem Sie in `Error.c` in der Zeile 5 das fehlende ; an das Ende der Zeile setzen. Wenn Sie sich Zeile 6 ansehen, werden Sie keinen Fehler sehen … also wandern Sie mit Ihren Augen nach oben, immer im Hinterkopf »fehlendes ; fehlendes ; fehlendes ;« und Sie werden das Problem bald gefunden haben.

✔ Fehler sind nicht das Ende der Welt! Eher der Anfang.

✔ Die Zeilennummer bezieht sich auf eine Zeile im Quelltext. Deswegen verwenden auch alle Texteditoren Zeilennummern, die auf dem Bildschirm irgendwo am oberen oder unteren Rand angezeigt werden.

✔ In integrierten Entwicklungsumgebungen wird der Cursor in der Regel sogar auf die Stelle oder Zeile gesetzt, an beziehungsweise in der Fehler zu finden ist – oder zumindest in die Nähe. Dafür haben Sie ja auch extra zahlen müssen.

Die Zeilennummer ist nicht unbedingt exakt, der Fehler kann auch in einer vorigen oder einer Folgezeile sein. Aber die Angabe ist zumindest sehr nahe dran.

✔ Andere beliebte Syntaxfehler außer dem vergessenen Semikolon sind fehlende Klammern, runde oder geschweifte, fehlende Anführungszeichen und überzählige Klammern.

✔ Möglicherweise denken Sie sich nun: »Computer, wenn du schon weißt, wo der Fehler ist, dann mach ihn doch auch weg.« Aber Computer arbeiten nicht so; solche Schlussfolgerungen sind ihnen fremd. Das ist eben der Nachteil von »Mach, was ich sage«: Computer können keine Gedanken lesen, also müssen SIE präzise sein. Immerhin sind sie sehr kleinlich, wenn es darum geht, Fehler zu finden.

Das fehlerhafte Programm reparieren

Um das Ganze wieder geradezubiegen, müssen wir das Programm reparieren. Dazu muss der Quelltext editiert, gespeichert und neu kompiliert werden.

Das Programm `Error.c` kann korrigiert werden, indem ein fehlendes Semikolon eingefügt wird. Die Zeile sollte also wie folgt aussehen:

```
printf("Dieses Programm arbeitet fehlerfrei ");
```

Die Zeile sollte immer noch eingerückt sein. Ich habe auch gleich noch den Text geändert, zusätzlich zum fehlenden Semikolon am Zeilenende.

Speichern Sie `Error.c` und kompilieren Sie das Programm neu. Nun sollte alles problemlos laufen. Falls nicht, wissen Sie ja nun, wie man mit Fehlern umgeht.

Sie können nun das Programm starten und die Ausgabe bewundern:

```
Dieses Programm arbeitet fehlerfrei!
```

Ich kann Ihnen nicht immer sagen, wo die Fehler in Ihrem Programm stecken. `Error.c` ist das einzige Programm im Buch, das einen absichtlichen Fehler enthält. Wenn Sie auf einen Fehler treffen, sollten Sie die Fehlermeldung untersuchen, um den Ort und Grund des Fehlers einzukreisen. Danach überprüfen Sie den eingetippten Quelltext anhand des im Buch angegebenen Quelltextes. Auf diese Weise finden Sie, was falsch ist. Aber wenn Sie ganz allein unterwegs sind und eigene Programme schreiben, dann hilft Ihnen wirklich nur die Fehlermeldung bei der Jagd nach dem Übeltäter.

Damit müssen Sie sich nicht belasten

Es gibt zwei Arten von Fehlern: Fehler und Warnungen. Manche Compiler nennen Fehler auch *kritische Fehler*.

Eine Warnung bedeutet so viel wie »Also appetitlich sieht's ja nicht gerade aus, aber ich serviere es trotzdem«. Wahrscheinlich wird Ihr Programm laufen, aber eventuell nicht ganz so wie gewünscht. Oder vielleicht ist der Compiler auch nur besonders sorgfältig. Bei den meisten Compilern kann man einige der übergenauen Warnungen ausschalten.

Der Fehler, oder auch kritische Fehler, bedeutet: »Herrscher über meinen Speicher, du hast etwas so Kriminelles getan, ich kann es nicht verantworten, dieses Programm laufen zu lassen«. Vielleicht ist das ein bisschen übertrieben. Letztlich ist der Compiler aber nicht in der Lage, Ihre Anweisungen zu verstehen.

Die fürchterlichen Linkerfehler

Beenden Sie das `Error.c`-Projekt noch nicht. Falls Sie zu schnell waren, öffnen Sie die Quelltextdatei noch einmal.

Ändern Sie die dritte Zeile im Quelltext, sodass sie wie folgt aussieht:

```
int Main()
```

Falls es Ihnen nicht auffällt, das Word `main` wurde in `Main` geändert, ein häufiger Tippfehler. Aber dies ist kein Syntaxfehler. C arbeitet so, dass es einfach annimmt, dass `Main` irgendetwas Ernsthaftes ist und dass es funktionieren wird. Aber der Linker, der die Programmteile miteinander verbindet, findet den Fehler, wenn er sich darüber wundert, wo denn eigentlich die Funktion `main` zu finden sein soll. Denn wie Sie im nächsten Abschnitt erfahren werden, muss jedes C-Programm eine Funktion `main` enthalten.

Speichern Sie die Datei und kompilieren Sie das Programm neu.

Es gibt einen Fehler! Schauen Sie sich an, wie eine mögliche Ausgabe aussehen könnte:

```
error.obj : Fehler LNK2001: nicht aufgelöstes externes Symbol _main
```

Es ist nun schwieriger, den Fehler zu finden, weil bei Linkerfehlern keine Zeilennummer angegeben werden.

- ✔ Ein Linkerfehler wird durch das Auftauchen des Begriffs *Linker* irgendwo in der Fehlermeldung angezeigt, manchmal unterscheidet sich ein solcher Fehler auch nur durch das *L* in der Fehlernummer, während ein Compilerfehler ein *C* in der Fehlernummer enthält.
- ✔ Es ist schwieriger, Linkerfehler zu akzeptieren als Compilerfehler, wo doch der Compiler automatisch für Sie linkt. Aber es passiert dennoch. Und anders als beim Compiler sind hier alle Fehler »kritisch«.

Linkerfehler beheben

Die Aufgabe des Linkers ist es, das Programm aus Einzelteilen zusammenzusetzen. Wenn er etwas findet, das er nicht erkennt, denkt er: »He, das könnte etwas aus einem anderen Programmteil sein«. Daher rutscht der Fehler zunächst beim Compiler durch. Aber wenn der Linker nachsieht, wo das unbekannte Wort definiert ist, hält er dann seine Fehlerflagge mit aller Gewalt hoch, nachdem er nichts findet.

Um Linkerfehler zu beheben, folgen Sie den gleichen Schritten wie bei der Behebung von Compilerfehlern. Mit der Fehlermeldung als Führer suchen Sie die verursachende Zeile im Quelltext. Dann beheben Sie den Tippfehler oder den Fehler, um das Programm zu korrigieren.

Ändern Sie also die Zeile mit dem `Main` wie folgt:

```
int main()
```

Speichern Sie die Änderung und erzeugen Sie das Programm neu. Nun sollte kein Fehler mehr angezeigt werden.

Linkerfehler enthalten häufig eine nutzlose Zugabe beim beanstandeten Wort: einen Unterstrich _.

Bei Linkerfehlern wird in der Regel keine Zeilennummer angegeben, aber man sieht zumindest den beanstandeten Begriff. Benutzen Sie die Suchfunktion des Texteditors, um nach dem Begriff im Quelltext zu suchen. Die Beseitigung läuft immer gleich, egal welchen Grund es gibt: ändern und neu kompilieren.

Linkerfehler bedeuten normalerweise, dass das Programm nicht erzeugt wurde. Das ist gut, denn ohne das fehlende Stück würde die `exe`-Datei Ihren Computer abstürzen lassen.

Alles über Fehler

Eine häufig vorkommende Aussage in der Computerprogrammierung ist, dass man häufiger Fehler beseitigt als Programme schreibt. Fehler sind überall, und die Beseitigung kann Jahre dauern, deswegen ist es auch so schwierig, gute Software zu schreiben.

Syntaxfehler: Die häufigsten und am leichtesten zu findenden Fehler; der Compiler entdeckt diese Fehler bei der Übersetzung des Quelltextes. Die meisten Syntaxfehler sind Tippfehler oder ähnliche Kleinigkeiten. Normalerweise zeigt der Compiler diese mit genauem Ort und Grund an.

Linkerfehler: Kommen als Erstes ins Spiel, wenn Anweisungen falsch geschrieben werden. In fortgeschrittener Programmierung, wenn man mit vielen Quelltextdateien oder Modulen arbeitet, deuten Linkerfehler auf fehlende oder fehlerhafte Module hin. Außerdem kann dies passieren, wenn der Linker eine spezielle Bibliotheksdatei (Library oder kurz Lib) sucht und nicht findet. Auch hier wird ein Linkerfehler mit einem etwas anderen Text erzeugt.

Laufzeit- oder Runtimefehler: Ereignen sich, während das Programm läuft. Dies sind keine Bugs – das sind Dinge, die für Compiler und Linker absolut akzeptabel sind, aber nicht genau das, wofür sie eigentlich von Ihnen gedacht waren (das ist in C nichts Ungewöhnliches). Der häufigste Laufzeitfehler ist ein Zugriff auf einen Nullzeiger. Geduld, das lernen Sie noch später.

Bugs: Der schwierigste Typ Fehler, dem Sie begegnen. Der Compiler erzeugt ganz exakt das Programm gemäß Ihren geschriebenen Anweisungen, aber ob das Programm auch das Gewünschte tut, stellt sich erst bei Tests heraus. Falls nicht, müssen Sie am Quelltext noch etwas arbeiten. Bugs umfassen alles von zu langsamer Geschwindigkeit bis zu Dingen, die nur manchmal oder gar nicht funktionieren. Diese Fehler sind am schwersten zu entdecken und ein hochgradiger Frustrations- und Demotivationsgrund.

Wie die Sprache C aussieht

Jede neue Sprache wirkt verwirrend auf Sie. Ihre Muttersprache hat einen bestimmten Verlauf oder ein Wortmuster. Und die ganzen Buchstaben passen irgendwie zueinander. An fremde, neuartige Dinge gewöhnt man sich am besten durch Übung und Benutzung. Aber Sie brauchen eine Straßenkarte, damit Sie die Dinge einordnen können. Es macht keinen Sinn, blind ein C-Programm einzutippen, wenn Sie nicht die geringste Ahnung haben, was abläuft.

Der große Zusammenhang

Abbildung 1.1 stellt den Quelltext von `Goodbye.c` dar, das Beispiel vom Anfang dieses Kapitels.

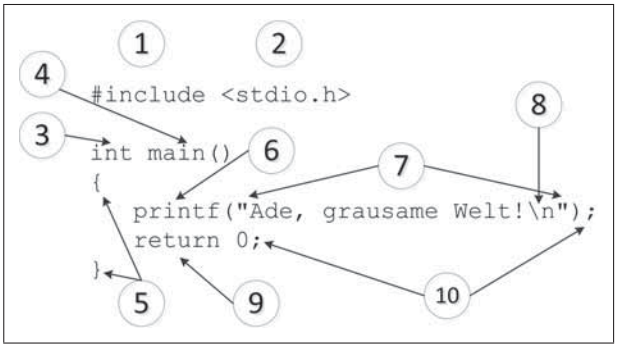

Abbildung 1.1: `Goodbye.c` *und sei*

Jedes Programm benötigt einen Einsprungpunkt. Wenn Sie ein Programm starten, wird es vom Betriebssystem auf seine Reise geschickt – wie ein Schiff den Hafen verlässt. Wie es letzte Pflicht des Lademeisters ist, sendet das Betriebssystem den Mikroprozessor an der richtigen Stelle ins Programm. Der Mikroprozessor übernimmt dann ab dem zugewiesenen Einsprungpunkt die Bearbeitung.

In allen C-Programmen ist der Einsprungpunkt die Funktion `main()`. Jedes C-Programm hat eine, auch `Goodbye.c` und die anderen Programme, die Sie bisher geschrieben haben. Die `main()`-Funktion ist die Maschine, die das Programm zum Leben erweckt und den Text am Bildschirm erzeugt.

Andere C-Programme verrichten andere Arbeiten in ihren `main()`-Funktionen. Aber egal was darin steht, es ist immer die erste Anweisung für den Computer, sobald das Programm gestartet wird.

- ✔ `main()` ist der Name der ersten (beziehungsweise höchsten) Funktion in jedem C-Programm. C-Programme können auch andere Funktionen enthalten, aber `main()` ist die wichtigste.
- ✔ Wenn man eine Funktion aus einem C-Programm erwähnt, schreibt man hinter den Namen Klammern, so wie in `main()`. Das hat nichts zu bedeuten, jeder macht das und wir werden es auch tun.

Bei einigen Fehlermeldungen haben Sie vielleicht auch schon den Hinweis gelesen »... in Funktion main«. Dies bedeutet, dass der Fehler innerhalb der Funktion `main()` aufgetreten ist.

- ✔ Eine Funktion ist eine Maschine – eine Ansammlung von Anweisungen, die etwas tun. C-Programme können eine Menge Funktionen haben, aber die `main()`-Funktion ist die oberste und wichtigste in einem C-Programm. Sie darf nicht fehlen.

Funktion. Funktion. Funktion. Gewöhnen Sie sich an das Wort.

Stückliste

Hier einige der interessanten Teile unseres C-Programms aus Abbildung 1.1:

1. `#include` wird auch Präprozessor-Direktive genannt, was ziemlich eindrucksvoll klingt und auch nicht unbedingt der richtige Begriff sein muss, aber Sie müssen sich das nicht merken. Sie bewirkt, dass der Compiler an dieser Stelle ein anderes Programm oder eine Datei einfügt (oder inkludiert, wie man auch auf Neudeutsch hochtrabend sagen kann). Dies spart in der Regel eine Menge Arbeit.

2. `<stdio.h>` ist ein Dateiname in spitzen Klammern (was man als den Versuch von C deuten kann, Sie dazu zu zwingen, sämtliche Klammerarten zu kennen). Die ganze Zeile `#include <stdio.h>` sagt dem Compiler, dass er die Datei `Stdio.h`, die standardmäßigen Ein- und Ausgabeanweisungen (englisch input/output), benutzen soll.

3. `int main` bezeichnet die Funktion `main`. Das `int` bezeichnet den Typ der Funktion oder das, was die Funktion erzeugt. Jedes C-Programm erzeugt mindestens eine Zahl, selbst unser kleines Beispiel; das `int` steht für die Erzeugung einer ganzen Zahl.

4. Zwei leere runde Klammern folgen dem Funktionsnamen. Manchmal stehen noch Dinge zwischen den Klammern, dazu kommen wir in Kapitel 5.

5. Die geschweiften Klammern fassen die Bestandteile der Funktion ein. Alles zwischen `{` und `}` gehört in Abbildung 1.1 zur Funktion `main()`.

6. `printf` ist eine andere Funktion, die den Computer etwas Bestimmtes tun lässt.

7. Zum `printf` gehören auch runde Klammern, diesmal allerdings mit Inhalt. Sie fassen hier Text oder auch einen so genannten String ein. Alles zwischen den Anführungszeichen `"` und `"` ist Teil des Strings.

8. Ein interessanter Teil des Strings ist das . Es handelt sich um einen Backslash mit einem kleinen `n`. Dies stellt das gleiche Zeichen dar, das auch durch die ↵-Taste erzeugt wird. Es beendet den String mit einer »neuen Zeile«.

9. Wie bereits erwähnt, muss jedes C-Programm etwas erzeugen, aber eigentlich sind wir noch gar nicht so weit. Wir liefern als relativ banales Ergebnis einfach eine `0` zurück, was die Aufgabe von `return` ist.

10. Schlussendlich enden die Zeile von `printf` und `return` mit einem Semikolon (`;`). Das Semikolon ermöglicht dem Compiler zu erkennen, wo eine Anweisung endet und eine andere beginnt – so wie ein Punkt das Ende eines Satzes kennzeichnet.

✔ Text in einem Programm heißt *String*. Zum Beispiel `"bla-bla-bla"` wäre ein String. Ein String wird von Anführungszeichen eingerahmt.

✔ Die Sprache C ist aus Schlüsselwörtern zusammengesetzt, die in Anweisungen auftauchen. Die Anweisungen enden mit einem Semikolon.

Darf ich vorstellen? Die Sprache C mit ihren Schlüsselwörtern

Die Sprache C ist ziemlich kurz. Es gibt nur 33 Schlüsselwörter in C – wäre Spanisch doch auch so einfach. Tabelle 1.1 listet die Schlüsselwörter auf.

asm	enum	signed
auto	extern	sizeof
break	float	static
case	for	struct
char	goto	switch
const	if	typedef
continue	int	union
default	long	unsigned
do	register	void
double	return	volatile
else	short	while

Tabelle 1.1: Schlüsselwörter der Sprache C

Nicht schlecht, oder? Aber das sind nicht alle Wörter, denen man in C begegnet. Andere Wörter oder Anweisungen werden *Funktionen* genannt. Diese umfassen Kostbarkeiten wie `printf` und einige Dutzend andere Funktionen, die zusammen mit den grundlegenden Schlüsselwörtern die Erzeugung von Programmen ermöglichen.

Arbeiten Sie unter Windows oder Linux, finden Sie zusätzliche spezielle Funktionen für die jeweiligen Betriebssysteme, die auf den grundlegenden Kern der C-Standardfunktionen aufgesetzt sind. Und falls Sie unter Windows programmieren, finden Sie Hunderte von spezifischen Funktionen nur für Windows, sodass der Umfang der Sprache C plötzlich enorm wächst. Nein, Sie müssen sich nicht alle merken. Deswegen kommen alle C-Compiler mit einer Sprachreferenz daher, die man nach einiger Zeit direkt neben dem Bildschirm liegen hat.

Sprachen sind mehr als nur eine Ansammlung von Wörtern. Sie umfassen auch eine Grammatik, die beschreibt, wie man die Wörter zusammenfasst, sodass verständliche Ideen hervorgebracht werden.

Zusätzlich zur Grammatik benötigt eine Sprache auch Regeln, Ausnahmen, Pünktchen und Klümpchen und allerlei Schnickschnack und Verdrehungen. Programmiersprachen sind natürlichen Sprachen recht ähnlich, auch hier gibt es verschiedene Teile und eine Menge Regeln.

- ✔ Sie werden sich die 33 Wörter nicht merken müssen.
- ✔ Wahrscheinlich werden Sie die meiste Zeit mit weniger als der Hälfte auskommen.
- ✔ Einige der Schlüsselwörter sind wirkliche englische Wörter. Andere sind Abkürzungen oder Zusammenfassungen von zwei oder mehr Wörtern.

- ✔ Jedes Schlüsselwort ist fest mit einer Problemstellung verbunden. Man benutzt nicht einfach nur das Schlüsselwort `else`, man muss es in einem Zusammenhang benutzen.
- ✔ Funktionen wie `printf` benötigen Klammern und eine Menge Informationen in den Klammern. (Machen Sie sich aber darüber jetzt noch keine Gedanken. Nicken Sie einfach und sprechen Sie mir nach: »Stimmt schon, `printf` braucht 'ne Menge Informationen«.)

Übrigens, wegen der Tatsache, dass `printf` eine Funktion ist und kein Schlüsselwort, muss am Programmanfang die Zeile `#include <stdio.h>` eingefügt werden. Die Datei `Stdio.h` enthält die genauen Informationen, wie `printf` funktioniert und was es dazu benötigt. Wenn Sie die Zeile mit dem `#include` weglassen, meckert der Compiler, dass er `printf` nun nicht mehr kennt.

Andere Sprachelemente von C

Es gibt noch eine Menge anderer Sprachelemente in C, die es für den Neuling ziemlich bizarr aussehen lassen. Im Moment steht zwischen Wissen und Unkenntnis nur die Zeit. Also machen Sie sich keine Gedanken wegen der Dinge, die Sie (noch) nicht wissen, sondern behalten Sie erst einmal die folgenden Punkte im Hinterkopf:

- ✔ Die Sprache C benutzt Wörter – Schlüsselwörter, Funktionen und so weiter – als Basiselemente.
- ✔ Außer den Wörtern gibt es auch Symbole. Manchmal werden diese Symbole Operatoren genannt, manchmal auch anders. Zum Beispiel wird das Pluszeichen `+` in C benutzt, um Dinge zu addieren.
- ✔ Für die Wörter gibt es Optionen und Regeln, wie man sie benutzt. All diese Regeln werden in den C-Handbüchern der Compiler erklärt; man muss sie nicht auswendig können. Dennoch werden Ihnen einige mit der Zeit in Fleisch und Blut übergehen.
- ✔ Klammern werden in C immer benutzt, um eine bestimmte Art von Informationen zusammenzufassen, die von C-Wörtern benötigt werden. Diese Wörter werden zusammengesetzt, um Anweisungen zu erstellen, was den Sätzen einer natürlichen Sprache entspricht. Die Anweisungen enden alle mit einem Semikolon.
- ✔ Geschweifte Klammern werden benutzt, um Teile des Programms zu gruppieren. Einige Schlüsselwörter benutzen geschweifte Klammern, um die zu ihnen gehörenden Teile an sich zu binden, und alle von Ihnen in einem Programm erstellten Funktionen verwenden geschweifte Klammern für den gleichen Zweck. Dies konnten Sie in diesem Kapitel bei der Funktion `main()` sehen.
- ✔ All diese Dinge zusammengepackt (und eine Menge anderes Zeugs, das ich im Moment mal lieber nicht erwähne) bilden die Syntax der Sprache C. Syntax bezeichnet die Art und Weise, wie man Sprachen zusammensetzt.

Eingaben und Ausgaben (die hier mal was Gutes sind)

Bei Computern dreht sich alles um Ein- oder Ausgaben. Eingabe – Ausgabe: Sie geben etwas ein und erhalten eine Antwort, eine Reaktion. Stellen Sie eine Frage, werfen Sie einen Euro ein und Sie erhalten Ihr Antwortkärtchen – darum geht es. Wie schon zu Beginn dieses Kapitels gesagt: Es ist Ihre Aufgabe als Programmierer, Programme zu schreiben, die etwas tun. Zum jetzigen Zeitpunkt mit unserem Wissensstand ist es okay, wenn das triviale Dinge sind. Schon bald werden Sie Programme schreiben, die wirklich etwas tun.

Stellen Sie sich bei Herrn Computer vor

Um den Nutzen und die Anwendung von Eingaben und Ausgaben kennenzulernen, können Sie das folgende Programm ausprobieren, `Whoru.c` – der Name leitet sich vom Englischen »Who are you?« ab.

Der Sinn und Zweck des Programms ist es, dass Sie Ihren Namen über die Tastatur eingeben und dass der Computer daraufhin Ihren Namen auf dem Bildschirm anzeigt, zusammen mit einer freundlichen Begrüßung.

```
#include <stdio.h>

int main()
{
   char me[20];
   printf("Wie heissen Sie?");
   scanf("%s", me);
   printf("Hallo %s, nett, Sie kennenzulernen! ", me);
   return 0;
}
```

Listing 1.3: Name: `Whoru.c`

Starten Sie Ihren Texteditor und beginnen Sie wieder mit einer neuen Datei. Geben Sie den Quelltext wie oben beschrieben ein und speichern Sie die Datei unter dem Namen `Whoru.c` ab. Inhalte interessieren uns noch nicht, erst mal abtippen und singen Sie dabei von mir aus vor sich hin, wenn Sie das entspannt.

Kompilieren Sie das Programm `Whoru.c` mit dem Compiler. Falls Ihr Compiler den Linkvorgang als separaten Schritt ausführt, linken Sie natürlich auch das Programm.

Wenn Syntaxfehler oder andere Dinge auftreten, prüfen Sie noch einmal Ihre Eingabe auf Abweichungen von den Vorgaben hier im Buch. Achten Sie auch auf Kleinigkeiten, Semikolons, Kommas oder Prozentzeichen.

Wenn alles geklappt hat, haben Sie nun auch die Datei `Whoru.exe` auf der Festplatte. Starten Sie das Programm aber noch nicht! Das kommt erst gleich. Noch einen Moment Geduld.

Bei den integrierten Entwicklungsumgebungen können Sie mit dem Menübefehl DATEI|NEU ein neues Fenster für den Editor öffnen.

- ✔ Eine *Variable* ist ein Speicherplatz, an dem man Informationen in einem Programm ablegen kann. Eine Stringvariable, so wie `me` in `Whoru.c`, wird benutzt, um Text zu speichern. Numerische Variablen speichern Zahlen und Werte. Kapitel 3 wird Ihren brennenden Wissensdurst zu diesem Thema befriedigen.
- ✔ Beseitigen Sie eventuelle Fehler durch Änderungen und Neukompilierung.
- ✔ Ein typischer Anfängerfehler: Anführungszeichen vergessen! Das zweite " fehlt! Vergewissern Sie sich, dass Sie immer ein Paar von " benutzen. Falls eines fehlt, erhalten Sie einen Fehler. Das Gleiche gilt für alle Klammersorten: Nehmen Sie immer ein Pärchen, so wie bei der Arche Noah.
- ✔ Falls Linkerfehler auftreten, achten Sie auf die Zeilennummer, falls vorhanden, oder suchen Sie nach dem beanstandeten Wort im Quelltext.

Die Belohnung!

Falls der Compiler das Programm `Whoru` nicht automatisch startet – einige tun dies –, starten Sie es nun. Die Ausgabe in der Kommandozeile sollte so aussehen:

```
Wie heissen Sie?
```

Das Programm wartet nun auf Ihre Eingabe. Also los, geben Sie Ihren Namen ein und drücken Sie ↵.

Falls Sie `C-Guru` eingetippt haben, wird folgende Zeile dargestellt:

```
Hallo C-Guru, nett, Sie kennenzulernen!
```

Falls die Ausgabe anders aussieht oder das Programm nicht so arbeitet oder einen Fehler erzeugt, prüfen Sie noch einmal den Quelltext. Ändern Sie ihn, um Fehler zu beseitigen, und kompilieren Sie das Programm neu.

In einigen Entwicklungsumgebungen müssen Sie die Fenster wechseln, damit Sie das Fenster mit der Kommandozeile sehen.

Dies ist ein Beispiel für ein Programm, das eine Eingabe übernimmt und etwas ausgibt. Es macht mit der Eingabe nichts außer dem Erzeugen der Ausgabe, aber das reicht schon für die Erfüllung des EVA-Prinzips: Eingabe – Verarbeitung – Ausgabe.

Pst! Geheime Informationen über die Arbeitsweise von `Whoru.c`

Wenn Ihr C-Programm läuft, arbeitet der Computer zuerst die `main`-Funktion ab (es passieren zuvor noch ein paar andere Dinge, aber das interessiert uns hier nicht). In `Whoru.c` gibt es fünf Anweisungen in der Funktion `main` – die wie folgt arbeiten:

```
char me[20];
```

Die vorstehende Zeile sagt dem Compiler, dass er eine Stringvariable erzeugen soll. `char` steht für *Char*acter, das englische Wort für Zeichen. Dies steht hier im Gegensatz zu einem Speicher für Zahlen oder einem Schuhschrank, in dem Sie Ihre alten Schuhe aufheben. Der Name der Stringvariablen ist `me` (man kann einer Variablen praktisch jeden Namen geben – Kapitel 3 enthält die Regeln dazu). Die Zahl in den eckigen Klammern sagt dem Compiler, dass er für 20 Zeichen Speicherplatz organisieren muss – das sind hier 20 Bytes. Sie können also Text mit maximal 20 Zeichen Länge in `me` speichern.

```
printf("Wie heissen Sie?");
```

Diese Zeile sagt dem Compiler, dass er die Funktion `printf` aufrufen soll. `printf` zeigt den String `Wie heissen Sie?` auf dem Bildschirm an. Das ist Aufforderung des Programms, dass es nun gerne eine Eingabe hätte.

```
scanf("%s", me);
```

Hier wird die Funktion `scanf` aufgerufen. `scanf` liest für eine spezielle Eingabe die Tastatur ein. Die Art der Eingabe ist definiert als `%s`, was `scanf` mitteilt: »Hole eine Eingabe von einem String von der Tastatur ab und höre auf, wenn die ↵-Taste gedrückt wird«. Der Teil mit dem `me` sagt der `scanf`-Funktion, dass der Text in der Variablen `me` zu speichern ist.

```
printf("Hallo %s, nett, Sie kennenzulernen! ", me);
```

Diese nächste Anweisung sagt dem Compiler, dass wieder die `printf`-Funktion benötigt wird. Der Text in Anführungszeichen wird ausgegeben, aber `printf` zaubert aus dem `%s` noch etwas Besonderes. Das `%s` bedeutet so viel wie »Ich bin nur ein Platzhalter – hier ist noch eine Lücke, fülle diese Stelle mit Inhalt«. Der Inhalt, der diesen Platzhalter füllt, wird aus der Variablen `me` geholt, die deswegen hinter dem Text nach einem Komma auch noch angegeben wird.

Und schließlich muss das Programm noch ein Ergebnis erzeugen. Damit zeigt man dem Aufrufer an, ob das Programm einwandfrei gearbeitet hat und alles in Ordnung war. Will man »hier war alles prima, keine Probleme« melden, so meldet man eine `0`. Das `return` selbst kümmert sich darum, dass diese `0` auch geliefert wird.

```
return 0;
```

Glauben Sie's oder nicht, all dieses Zeugs wird nach einigen wenigen Kapiteln auch für Sie Sinn ergeben. Sie werden in kürzester Zeit zum rebellischen C-Programmierer. Keine Frage.

Erst das Chaos, dann die Ordnung

Das hier ist eine Bonusrunde, damit Sie jetzt bereits auf das Problem der »Unordnung« aufmerksam werden, bevor es Ihnen über den Kopf wächst. Was ich meine, ist der Ramsch, der auf Ihrer Festplatte bei der Entwicklung von C-Programmen hinterlassen wird. Die vielen Hilfs- und Zwischendateien, die der Compiler nebenbei erzeugt, wenn er Ihr Programm erstellt.

Es ist ein bisschen arg naiv anzunehmen, dass das Schreiben von C-Programmen eine nette und saubere Sache ist. Aber genau wie der Typ im Park, der eigentlich nur eine Taube füttern wollte, merken Sie bald, was auf Sie zurollt.

Der C-Compiler räumt sein Zimmer nicht auf

Der Compiler wirbelt über Ihre Festplatte wie ein Tornado, er spuckt ganze Lastwagenladungen mit Dateien aus. Das muss man im Griff behalten, außerdem müssen Sie wissen, welche Dateien unbedingt aufgehoben werden müssen und welche nur Abfall sind. Erinnert ein bisschen ans Aufräumen des Kinderzimmers.

Für jedes erstellte C-Programm gibt es mindestens drei Dateien auf der Festplatte:

- ✔ Der Quelltext, Endung `c`
- ✔ Die Objektdatei, Endung `obj` oder Endung `o`
- ✔ Das Programm, Endung `exe`

Einige Compiler erzeugen noch mehr Dateien. Wenn Sie bisher also alle Programme aus diesem Kapitel erstellt haben, finden Sie bestimmt mindestens 15 Dateien auf der Festplatte.

- ✔ Wenn Sie Quelltexte editieren, erzeugt Ihr Editor möglicherweise ein Backup von der Datei, Endung `bak` für Backup oder auch mit einer Endung, die mit einer Tilde ~ beginnt. Löschen Sie diese Dateien nur im Ausnahmefall.

Microsoft Visual C++ erzeugt zusätzliche Dateien, `bsc` und `sbr`, diese werden für das Suchen im Quelltext benötigt, außerdem noch Zwischendateien wie `pch` und `pdb`, damit wird der Compilervorgang beschleunigt. Dazu gesellen sich noch einige andere; programmiert man unter Windows, gesellen sich weitere hinzu. Sie werden auch Dateien mit der Endung `dsp` und `dsw` finden; diese werden für die Projektverwaltung benötigt. Löschen Sie diese Dateien nicht.

Der Borland C++ Builder erzeugt als Zwischendateien unter anderem `tds`, als Projektdateien findet man hier die Endung `bpr`. Diese Dateien sollten Sie ebenfalls nicht löschen.

Der GNU-C-Compiler, wie er mit der frei verfügbaren Entwicklungsumgebung Code::Blocks mitkommt, ist relativ genügsam. Nur eine `cbp`-Datei wird zur Speicherung von Projekteinstellungen angelegt.

✔ Größere Projekte werden auch Dateien mit den Endungen `h` und `mak` enthalten; löschen Sie diese Dateien ebenfalls niemals!

Was aufheben, was wegwerfen?

Ich empfehle zunächst einmal, gar keine Datei zu löschen. Wichtig zu wissen, dass die beiden bedeutendsten Dateien im Moment auf `c` und auf `exe` enden.

✔ Solange Sie die Quelltextdateien (`*.c`) aufheben, können Sie jederzeit das Programm wieder neu erzeugen.

✔ Die Objektdateien `obj` könnten später gelöscht werden, wenn das Programm fertig ist (bei größeren Projekten sollten Sie dies aber nicht tun, denn bei Änderungen dauert der Kompiliervorgang viel länger, weil alles neu erzeugt werden muss).

Löschen Sie keine Backup-Dateien (`*.bak` und `*.~`), bis das Projekt fertig ist und alles wunschgemäß läuft. Der Grund dafür hat mit *Magie* zu tun und ist schwer zu erklären: Manchmal ändern Sie etwas, eine Kleinigkeit, und plötzlich spielt das Programm total verrückt. Falls Sie die Sicherheitskopie noch zur Hand haben, können Sie wieder auf den alten Stand zurückgreifen, indem Sie die Quelltextdatei löschen und die Sicherheitskopie nach `c` umbenennen. Ach ja: Danach wäre eine Neukompilierung eine gute Idee.

Und dann noch ein Wort zur Magie: Ich weiß nicht, woran das liegt, manchmal ändert man eine Kleinigkeit und nichts geht mehr. Und irgendwie bekommt man das auch nicht mehr hin, wenn man die Änderung rückgängig macht. Fragen Sie einen anderen erfahrenen Programmierer, er wird Ihnen das bestätigen können. Daher ist die Sache mit den Backups keine schlechte Sache. Wenn Ihnen so etwas passiert, führen Sie Änderungen am Backup danach wirklich vorsichtig durch.

Organisation ist alles!

Wie jedes Projekt auf dem Computer sollte auch jedes C-Projekt ein eigenes Verzeichnis auf der Festplatte bekommen. Zum Beispiel sollten Sie für Ihre C-Versuche ein Verzeichnis anlegen, nennen Sie es einfach `C` oder `Cprojekt` oder wie auch immer. Darunter sollten Sie für jedes Projekt ein Unterverzeichnis anlegen. Der Sinn? Organisation und Ordnung. Innerhalb dieses Verzeichnisses hat alles mit C zu tun, damit Sie sich später auch wieder zurechtfinden.

Andere Verzeichnisse sollten für die Projekte entstehen, an denen Sie arbeiten. Ein neues Sortierprogramm? Verzeichnis `Nsort`. Ein neues Shoot'em up-Ballerspiel? Verzeichnis `Shootem`.

Abbildung 1.2 zeigt, wie eine solche Baumstruktur arbeitet. Beachten Sie, dass die Wurzel des Baums, das Hauptverzeichnis, überall liegen kann. Im Verzeichnis des Compilers, im Hauptverzeichnis der Platte, wo auch immer.

✔ Falls Sie bereits angefangen haben, alles zu verstreuen, sollten Sie nun zunächst gemäß den Ratschlägen ein wenig Ordnung schaffen.

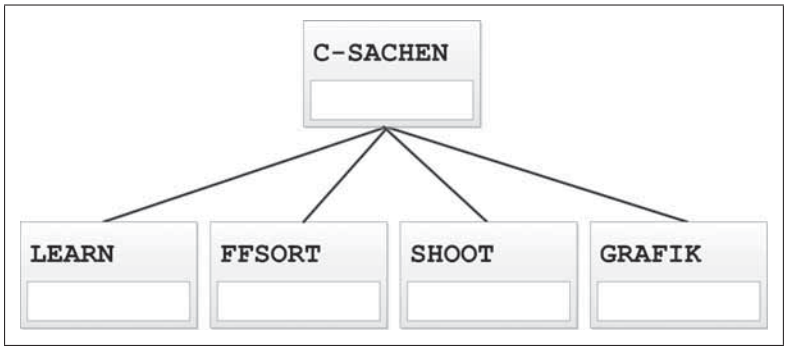

Abbildung 1.2: Organisation der C-Projekte auf der Festplatte

Warum der Aufwand? Zunächst mal, weil Sie damit die notwendigen Dateien zusammenhalten. C erzeugt eine Menge andere Sachen und allzu leicht geht etwas auf der Festplatte verloren, gerade bei größeren Projekten. Darum gewöhnen Sie sich von Anfang an daran.

Wichtige C-Regeln, an die Sie nie denken werden

C ist gespickt mit Regeln. Sie lernen das nun, damit Sie gleich mit dem Vergessen beginnen können. Deswegen wird der Compiler bösartige Fehlermeldungen nach Ihnen werfen. Daran müssen Sie sich gewöhnen ... wie an unfreundliche Busfahrer, verspätete Züge und höhere Steuern.

Ein Wort der Warnung vorab: Sie müssen sich diese Regeln nicht merken! Die meisten tauchen später noch einmal auf, wenn Sie C-Programme schreiben und gegen die entsprechenden Regeln verstoßen.

Das hilfreiche Regelprogramm

Um sich die Regeln, gegen die man am leichtesten verstößt, einfach zu merken, habe ich diese in einem Programm mit dem Namen `Rules` zusammengefasst. Es zeigt einige Zeilen auf dem Bildschirm an, die Sie immer wieder daran erinnern können.

```
#include <stdio.h>

int main()
{
   printf("Klammern immer paarweise! ");
   printf("Kommentare immer paarweise! ");
   printf("Anweisungen enden mit einem ; ");
   printf("Leerzeichen kann man weglassen! ");
   printf("main() ist immer notwendig! ");
   printf("C besteht fast nur aus Kleinschreibung!
```

```
    Gross-/Kleinschreibung werden unterschieden! ");
    printf("Unterstriche _ beachten wir nicht! ");
    printf("Variablen werden vor der Benutzung deklariert! ");
    return 0;
}
```

Listing 1.4: Name: `Rules.c`

Prüfen Sie noch mal die Eingabe und starten Sie dann das Programm `Rules.exe`. Ihre Ausgabe sollte so aussehen:

```
Klammern immer paarweise!
Kommentare immer paarweise!
Anweisungen enden mit einem ;
Leerzeichen kann man weglassen!
main() ist immer notwendig!
C besteht fast nur aus Kleinschreibung!    Gross-/Kleinschreibung werden unter-
schieden!
Unterstriche _ beachten wir nicht!
Variablen werden vor der Benutzung deklariert!
```

- ✔ Achten Sie mal auf Zeile 10 und 11: Obwohl die Zeile im Quelltext aufgeteilt ist, wird nur eine einzige Zeile in der Ausgabe erzeugt. Das liegt daran, weil das Zeichen dem `printf` mitteilt, dass die beiden Zeilen miteinander verbunden werden sollen.
- ✔ Das sind alle Regeln? Ha! Sie sollten sich glücklich schätzen. Im Grunde sollten Sie der Kakerlaken-Regel folgen: Für jede, die Sie sehen, gibt es noch zehn weitere in den dunklen Ecken. Mal im Ernst, die obigen Regeln werden am häufigsten verletzt. Zumindest ging es mir so.

Zeit für eine Bonusrunde

Schauen Sie sich mal den folgenden Textstring an:

```
"Das ist ein Textstring."
```

Das Teil nennt sich String. Es ist ein Text, eingesperrt zwischen zwei Anführungszeichen; auf diese Weise handhabt C Texte.

In diesem Kapitel wurde die Funktion `printf` mehrfach benutzt, um Text auszugeben, Text wie diesen. In fast jedem Fall endete der Text mit:

```
\n
```

Wie im Griechischen, oder? Auf C-isch heißt das »Gib mal 'ne neue Zeile rüber«. Das `n` steht hier für »neue Zeile« oder eigentlich »newline«.

Entfernen Sie zum Ausprobieren mal im Programm `Rules` alle `\n` aus dem Quelltext. Starten Sie dann das Programm. Sie sollten die folgende unübersichtliche Ausgabe erhalten:

Klammern immer paarweise!Kommentare immer paarweise!Anweisungen enden mit einem ; Leerzeichen kann man weglassen!main() ist immer notwendig!C besteht fast nur aus Kleinschreibung! Gross-/Kleinschreibung werden unterschieden!Unterstriche _ beachten wir nicht!Variablen werden vor der Benutzung deklariert!

Ohne `\n` im Text werden alle Texte bei der Ausgabe hintereinander geschrieben. Mit dem `\n` fügt die `printf`-Funktion immer am Ende des Textes eine neue Zeile ein – und das Ganze sieht schön ordentlich aus.

- ✔ In C wird das `\n` in einem String so benutzt, als wenn Sie die ↵-Taste gedrückt hätten.
- ✔ Es ist immer ein `\n`, mit kleinem `n`. C ist fast immer Kleinschreibung.
- ✔ Das `\n` wird newline genannt, aber man sagt auch »Släsch n« oder »Bäcksläsch n« dazu – solange Sie es nicht laut sagen, ist das auch okay.
- ✔ Tabelle 2.1 in Kapitel 2 enthält einen Überblick über vergleichbare Buchstaben, die ähnliche Wirkungen wie `\n` haben.