

Der Inhalt (im Überblick)

	Einführung	xxv
1	Erste Schritte mit C: <i>Eintauchen</i>	1
2	Speicher und Zeiger: <i>Worauf zeigst du?</i>	41
2.5	Strings: <i>Stringtheorie</i>	83
3	Kleine Werkzeuge erstellen: <i>Eine Sache tun, und das gut</i>	103
4	Mehrere Quelldateien: <i>Zerlegen und zusammenbauen</i>	157
	1. C-Projekt: <i>Arduino</i>	207
5	Structs, Unions und Bitfelder: <i>Eigene Strukturen</i>	217
6	Datenstrukturen und dynamischer Speicher: <i>Brücken bauen</i>	267
7	Fortgeschrittene Funktionen: <i>Ihre Funktionen auf Vordermann bringen</i>	311
8	Statische und dynamische Bibliotheken: <i>Code-Wiederverwendung</i>	351
	2. C-Projekt: <i>OpenCV</i>	389
9	Prozesse und Systemaufrufe: <i>Grenzverletzungen</i>	397
10	Interprozesskommunikation: <i>Ein nettes Gespräch</i>	429
11	Sockets und Netzwerke: <i>127.0.0.1 ist ein toller Ort</i>	467
12	Threads: <i>Parallelwelten</i>	501
	3. C-Projekt: <i>Blasteroids</i>	523
A	Was übrig bleibt: <i>Die Top Ten (nicht behandelt)</i>	539
B	C-Themen: <i>Zusammenfassungen im Überblick</i>	553

Der Inhalt (jetzt ausführlich)

Einführung

Ihr Gehirn und C. Sie versuchen, etwas zu lernen, und Ihr Hirn tut sein Bestes, damit das Gelernte nicht *hängen bleibt*. Es denkt nämlich: »Wir sollten lieber ordentlich Platz für wichtigere Dinge lassen, z. B. für das Wissen darüber, welche Tiere einem gefährlich werden könnten oder dass es eine ganz schlechte Idee ist, nackt Snowboard zu fahren.« Tja, wie schaffen wir es nun, Ihr Gehirn davon zu überzeugen, dass Ihr Leben davon abhängt, etwas über C zu wissen?

Für wen ist dieses Buch?	xxvi
Wir wissen, was Sie denken.	xxvii
Metakognition	xxix
So machen Sie sich Ihr Gehirn untertan	xxxii
Lies mich	xxxii
Die technischen Gutachter	xxxiv
Danksagungen	xxxv

Erste Schritte mit C

Eintauchen

1

Wollen Sie dem Rechner auf die Finger schauen?

Müssen Sie **Hochleistungscode** für ein neues Spiel schreiben? Einen **Arduino**-Controller programmieren? Oder diese raffinierte **externe Bibliothek** in Ihrer iPhone-App einsetzen? Wenn das der Fall ist, wird C Ihr bester Freund werden. C arbeitet auf einer **viel elementarerer Ebene** als die meisten anderen Programmiersprachen. Wenn Sie C verstanden haben, werden Sie auch viel besser verstehen, **was eigentlich im Herzen der Maschine vor sich geht**. C kann Ihnen sogar helfen, andere Sprachen besser zu verstehen. Zögern Sie nicht! Machen Sie Ihren Compiler bereit – auf dass Sie schon bald loslegen können.

C, die Sprache für kleine, schnelle Programme	2
Aber wie sieht ein vollständiges C-Programm aus?	5
Aber wie führen Sie das Programm aus?	9
Zwei Arten von Befehlen	14
So sieht der Code bislang aus	15
Kartenzählen? In C?	17
Vergleiche kennen nicht nur Gleichheit ...	18
Wie sieht der Code jetzt aus?	25
Weichen stellen	26
Wenn einmal keinmal ist ...	29
Schleifen haben oft eine ähnliche Struktur ...	30
Mit break ausbrechen ...	31
Ihr C-Werkzeugkasten	40

Speicher und Zeiger

Worauf zeigst du?

2

Wenn Sie C wirklich beherrschen wollen, müssen Sie verstehen, wie C mit Speicher umgeht.

C bietet Ihnen ziemlich umfangreiche Möglichkeiten, zu *steuern*, wie Ihr Programm den **Speicher des Systems** nutzt. In diesem Kapitel werden wir den Vorhang lüften und Ihnen zeigen, was passiert, wenn Sie **Variablen lesen und schreiben**. Sie werden erfahren, **wie Arrays funktionieren**, wie man einige **garstige Speicherprobleme vermeidet**, und natürlich auch einsehen lernen, dass der Weg zum gewieften C-Programmierer nur über **die Beherrschung von Zeigern und der Speicheradressierung** führt.

C-Code enthält Zeiger	42
Ein Blick in den Speicher	43
Segel setzen mit Zeigern	44
Versuchen wir, der Variablen einen Zeiger zu übergeben	47
Speicherzeiger einsetzen	48
Wie übergibt man einer Funktion einen String?	53
Array-Variablen sind wie Zeiger ...	54
Was der Computer denkt, wenn er Ihren Code ausführt	55
Aber Array-Variablen sind nicht das Gleiche wie Zeiger	59
Warum Arrays wirklich mit 0 beginnen	61
Warum Zeiger Typen haben	62
Zeiger für die Dateneingabe verwenden	65
Aufgepasst mit scanf()	66
fgets() ist eine Alternative zu scanf()	67
Stringlitterale können nie aktualisiert werden	72
Wenn Sie einen String ändern wollen, kopieren Sie ihn	74
Speicher speichern	80
Ihr C-Werkzeugkasten	81

Strings

Stringtheorie

2.5

Strings muss man nicht nur lesen.

Sie haben erfahren, dass Strings in C eigentlich `char-Arrays` sind, aber was C Sie mit ihnen *anstellen* lässt, das wissen Sie noch nicht. Dazu müssen wir uns ***string.h*** zuwenden. *string.h* ist ein Teil der C-Standardbibliothek, der sich ganz der **Stringmanipulation** widmet. Wenn Sie Strings **verkett**en, einen String in einen anderen **kopieren** oder zwei Strings **vergleichen** wollen, können die Funktionen in *string.h* nützlich sein. In diesem Kapitel werden Sie sehen, wie Sie ein **Array mit Strings** erstellen, bevor wir uns genauer ansehen werden, wie man mit der Funktion **`strstr()`** *in Strings sucht*.

Frank verzweifelt gesucht	84
Ein Array mit Arrays erstellen	85
Strings finden, die den Suchtext enthalten	86
Die Funktion <code>strstr()</code> nutzen	89
Zeit, dass wir uns unseren Code ansehen	94
Array von Arrays vs. Array von Zeigern	98
Ihr C-Werkzeugkasten	101

Kleine Werkzeuge erstellen

3

Eine Sache tun, und das gut

Alle Betriebssysteme beinhalten kleine Werkzeuge.

Kleine in C geschriebene Werkzeuge erledigen **kleine spezielle Aufgaben**, schreiben oder lesen Dateien oder filtern Daten. Wenn Sie komplexere Aufgaben bewältigen müssen, können Sie *mehrere Werkzeuge hintereinanderschalten*. Aber wie werden diese kleinen Werkzeuge erstellt? In diesem Kapitel werden wir uns die Bausteine der Erstellung kleiner Werkzeuge anschauen. Sie werden lernen, wie man **Kommandozeilenoptionen** steuert, wie man **Informationsströme** verarbeitet und **Umleitungen** einsetzt, um im Handumdrehen Werkzeuge zu bauen.

Kleine Werkzeuge können große Probleme lösen	104
So sollte das Programm funktionieren	108
Aber Sie nutzen noch keine Dateien ...	109
Sie können Ihre Daten umleiten	110
Die Standardfehlerausgabe	120
Standardmäßig wird die Standardfehlerausgabe an den Bildschirm gebunden	121
fprintf() schreibt in einen Datenstrom	122
Aktualisieren wir den Code, damit er fprintf() nutzt	123
Kleine Werkzeuge sind flexibel	128
Ändern Sie geo2json nicht	129
Eine andere Aufgabe erfordert ein anderes Werkzeug	130
Eingaben und Ausgaben mit einer Pipe verbinden	131
Das bermuda-Werkzeug	132
Aber was ist, wenn Sie mehr als eine Datei ausgeben wollen?	137
Der eigene Datenstrom	138
main() kann mehr	141
Lassen Sie die Bibliothek für sich wirken	149
Ihr C-Werkzeugkasten	156

Mehrere Quelldateien

4

Zerlegen und zusammenbauen

Wenn Sie ein großes Programm erstellen, heißt das nicht, dass Sie auch eine große Quelldatei haben wollen.

Können Sie sich vorstellen, wie schwierig und zeitaufwendig die Wartung einer einzigen Quelldatei bei umfangreichen Programmen werden kann? In diesem Kapitel werden Sie erfahren, wie Ihnen C ermöglicht, Quellcode in **kleine, handhabbare Happen** zu zerlegen und diese dann zu **einem großen Programm** zusammenzusetzen. Auf dem Weg dorthin werden Sie etwas mehr über die **Feinheiten von Datentypen** erfahren und werden jemandem über den Weg laufen, der einer Ihrer besten Freunde werden wird: **make**.

Datentypen-Schnellkurs	162
Großes darf man nicht in Kleines stecken	163
Mit Casts floats in ganze Zahlen packen	164
Oh nein, arbeitslose Schauspieler am Werk ...	168
Schauen wir uns an, was dem Code widerfahren ist	169
Compiler mögen keine Überraschungen	171
Die Deklaration von der Definition trennen	173
Ihre erste Header-Datei	174
Bei Gemeinsamkeiten ...	182
Sie können Code auf mehrere Dateien aufteilen	183
Kompilieren – was steckt dahinter?	184
Der gemeinsame Code braucht einen Header	186
Es ist kein Mysterium ... oder doch?	189
Nicht alles neu kompilieren	190
Erst aus Quellen Objektdateien machen	191
Das Nachhalten der Dateien ist aufwendig	196
Die Erstellung mit make automatisieren	198
Wie make funktioniert	199
make mit einem makefile über Ihren Code informieren	200
Abheben!	205
Ihr C-Werkzeugkasten	206