

|                      |           |
|----------------------|-----------|
| <b>Vorwort</b> ..... | <b>XI</b> |
|----------------------|-----------|

|                                   |          |
|-----------------------------------|----------|
| <b>Teil I: Die Umgebung</b> ..... | <b>1</b> |
|-----------------------------------|----------|

|  |          |
|--|----------|
| <b>1 Richten Sie sich Ihre Umgebung ein</b> .....                              | <b>3</b> |
| Einen Paketmanager einsetzen .....   | 4        |
| C unter Windows kompilieren .....  | 6        |
| POSIX für Windows .....  | 6        |
| C mit POSIX kompilieren .....  | 8        |
| C ohne POSIX kompilieren .....   | 8        |
| Wo bitte geht es zur Bibliothek? .....   | 9        |
| Ein paar meiner Lieblings-Flags .....  | 11       |
| Pfade .....  | 12       |
| Runtime-Linking .....  | 15       |
| Makefiles verwenden .....  | 16       |
| Variablen setzen .....   | 17       |
| Die Regeln .....   | 19       |
| Bibliotheken über ihren Quellcode nutzen .....                                 | 23       |
| Bibliothek über ihren Quellcode nutzen – auch wenn Ihr Sysadmin das nicht will | 24       |
| C-Programme über Here-Dokumente kompilieren .....                              | 26       |
| Header-Dateien an der Befehlszeile einbinden .....                             | 26       |
| Der vereinheitlichte Header .....  | 27       |
| Here-Dokumente .....   | 28       |
| Von Stdin kompilieren .....  | 29       |

|          |   |           |
|----------|---|-----------|
| <b>2</b> | <b>Debuggen, Testen, Dokumentieren</b>                                | <b>31</b> |
|          | Einen Debugger verwenden  | 31        |
|          | GDB-Variablen   | 35        |
|          | Geben Sie Ihre Strukturen aus   | 37        |
|          | Mit Valgrind auf Fehler prüfen  | 40        |
|          | Unit-Tests  | 41        |
|          | Ein Programm als Bibliothek verwenden                                 | 44        |
|          | Abdeckung   | 46        |
|          | Dokumentation einweben  | 47        |
|          | Doxygen   | 47        |
|          | Literaler Code mit CWEB   | 49        |
|          | Fehlerprüfung   | 51        |
|          | Wie ist der Anwender in den Fehler involviert?                        | 51        |
|          | Der Kontext, in dem der Anwender arbeitet                             | 53        |
|          | Wie sollte ein Hinweis auf einen Fehler zurückgegeben werden?         | 54        |
| <b>3</b> | <b>Verpacken Sie Ihr Projekt</b>                                      | <b>55</b> |
|          | Die Shell   | 56        |
|          | Shell-Befehle durch ihre Ausgabe ersetzen                             | 57        |
|          | Die Shell für Schleifen nutzen, um auf einem Satz Dateien zu arbeiten | 58        |
|          | Dateien prüfen  | 60        |
|          | fc  | 63        |
|          | Makefiles vs. Shell-Skripten  | 65        |
|          | Packen Sie Ihren Code mit den Autotools                               | 67        |
|          | Ein Autotools-Beispiel  | 69        |
|          | Das Makefile durch makefile.am beschreiben                            | 72        |
|          | Das configure-Skript  | 76        |
| <b>4</b> | <b>Versionsverwaltung</b>   | <b>81</b> |
|          | Änderungen per diff   | 82        |
|          | Git-Objekte   | 83        |
|          | Der Stash   | 87        |
|          | Bäume und ihre Zweige   | 88        |
|          | Merging   | 89        |
|          | Der Rebase  | 91        |
|          | Remote-Repositories   | 92        |

|  |           |
|--|-----------|
| <b>5 Mit anderen zusammenspielen</b>                         | <b>95</b> |
| Das Vorgehen   | 95        |
| Schreiben, damit es von anderen Sprachen gelesen werden kann | 95        |
| Die Wrapper-Funktion   | 96        |
| Datenstrukturen über die Grenze schmuggeln                   | 97        |
| Linken   | 98        |
| Python als Host  | 99        |
| Kompilieren und linken                                       | 100       |
| Das bedingte Unterverzeichnis für Automake                   | 101       |
| Distutils mit Unterstützung durch die Autotools              | 102       |

---

|                             |            |
|-----------------------------|------------|
| <b>Teil II: Die Sprache</b> | <b>105</b> |
|-----------------------------|------------|

|  |            |
|--|------------|
| <b>6 Ihr Weg zum Zeiger</b>                                      | <b>107</b> |
| Automatischer, statischer und manueller Speicher                 | 107        |
| Persistente Statusvariablen                                      | 110        |
| Zeiger ohne malloc   | 111        |
| Strukturen werden kopiert, Arrays werden als Alias weitergegeben | 113        |
| malloc und Speichertricks  | 115        |
| Das Schicksal liegt in den Sternen                               | 117        |
| All die Zeigerarithmetik, die Sie kennen müssen                  | 118        |

|  |            |
|--|------------|
| <b>7 C-Syntax, die Sie ignorieren können</b>                   | <b>123</b> |
| Kümmern Sie sich nicht darum, explizit aus main zurückzukehren | 124        |
| Lassen Sie Deklarationen fließen                               | 124        |
| Die Array-Größe zur Laufzeit setzen                            | 126        |
| Weniger Casting  | 127        |
| Enums und Strings  | 128        |
| Labels, goto, switch und break                                 | 130        |
| Durchdachtes goto  | 131        |
| switch   | 132        |
| Veraltetes Float   | 135        |

|  |            |
|--|------------|
| <b>8 Hindernisse und Gelegenheiten</b>                           | <b>139</b> |
| Robuste und ansprechende Makros schreiben                        | 139        |
| Präprozessortricks   | 143        |
| Mit static und extern verlinken                                  | 146        |
| Extern zu verlinkende Elemente nur in Header-Dateien deklarieren | 148        |

|   |            |
|---|------------|
| Das Schlüsselwort <code>const</code> . . . . .  | 150        |
| Nomen-Adjektiv-Form . . . . .   | 151        |
| Spannungen . . . . .  | 152        |
| Tiefe . . . . .   | 153        |
| Das Problem mit <code>char const **</code> . . . . .                                    | 154        |
| <b>9 Text</b> . . . . .   | <b>157</b> |
| Den Umgang mit Strings mithilfe von <code>asprintf</code> einfacher gestalten . . . . . | 157        |
| Sicherheit . . . . .  | 159        |
| Konstante Strings . . . . .   | 159        |
| Strings mit <code>asprintf</code> erweitern . . . . .                                   | 161        |
| Ein Loblied auf <code>strtok</code> . . . . .   | 162        |
| Unicode . . . . .   | 167        |
| Das Kodieren für C-Code . . . . .   | 169        |
| Unicode-Bibliotheken . . . . .  | 170        |
| Der Beispielcode . . . . .  | 171        |
| <b>10 Bessere Strukturen</b> . . . . .  | <b>175</b> |
| Compound-Literale . . . . .   | 176        |
| Initialisierung per Compound-Literal . . . . .  | 177        |
| Variadische Makros . . . . .  | 177        |
| Listen sicher abschließen . . . . .   | 179        |
| <code>foreach</code> . . . . .  | 180        |
| Eine Funktion vektorisieren . . . . .   | 180        |
| Designated Initializers . . . . .   | 182        |
| Arrays und Structs mit Nullen initialisieren . . . . .                                  | 184        |
| Typedefs retten Ihnen den Tag . . . . .   | 185        |
| Über Stil . . . . .   | 186        |
| Mehrere Elemente aus einer Funktion zurückgeben . . . . .                               | 187        |
| Fehler melden . . . . .   | 189        |
| Flexible Eingabewerte für Funktionen . . . . .  | 191        |
| Deklaren Sie Ihre Funktion im <code>printf</code> -Stil . . . . .                       | 192        |
| Optionale und benannte Argumente . . . . .  | 193        |
| Eine alte Funktion aufpolieren . . . . .  | 195        |
| Der <code>void</code> -Zeiger und die Strukturen, auf die er zeigt . . . . .            | 201        |
| Funktionen mit generischen Eingabewerten . . . . .                                      | 201        |
| Generische Strukturen . . . . .   | 206        |

|  |            |
|--|------------|
| <b>11 Objektorientierte Programmierung in C</b>                  | <b>211</b> |
| Was Sie nicht bekommen (und warum Sie es nicht vermissen werden) | 212        |
| Gültigkeitsbereich   | 212        |
| Überladen mit Operator-Überladung                                | 215        |
| Strukturen und Dictionaries erweitern                            | 220        |
| Eine Struktur erweitern  | 221        |
| Ein Dictionary implementieren                                    | 225        |
| Lassen Sie Ihren Code auf Zeigern auf Objekte basieren           | 229        |
| Funktionen in Ihren Structs                                      | 230        |
| Referenzen zählen  | 235        |
| Beispiel: Ein Substring-Objekt                                   | 235        |
| Ein agentenbasiertes Modell der Gruppenbildung                   | 239        |
| <b>12 Bibliotheken</b>   | <b>247</b> |
| GLib   | 247        |
| POSIX  | 248        |
| Mit mmap riesige Datensätze verarbeiten                          | 248        |
| Einfaches Threading mit Pthreads                                 | 251        |
| Die GNU Scientific Library                                       | 259        |
| SQLite   | 261        |
| Die Abfragen   | 262        |
| libxml und cURL  | 264        |
| <b>Epilog</b>  | <b>269</b> |
| <b>Glossar</b>   | <b>271</b> |
| <b>Bibliografie</b>  | <b>275</b> |
| <b>Index</b>   | <b>277</b> |