For Heike and Mouna
Table of Contents

About the Author ................................................................. XVII
About the Technical Reviewer .................................................. XIX
Preface ................................................................................ XXI
  Who This Book Is For .......................................................... XXI
  Who This Book Is (Probably) Not For ..................................... XXII
  How to Read This Book ........................................................ XXII
  What’s In and What Not ........................................................ XXIII
  The HyperSpec ................................................................... XXIV
  Which Implementation .......................................................... XXIV
  Source Code ....................................................................... XXV
  The Index ........................................................................... XXV
  Typographical Conventions .................................................. XXV
  Acknowledgements ............................................................. XXVI

1. Symbols and Packages ....................................................... 1
  1-1. Understanding the Role of Packages and the Symbol Nomenclature 1
  1-2. Making Unique Symbols .................................................. 5
  1-3. Making Symbols Inaccessible .............................................. 9
      How Can We Fix This? ...................................................... 11
  1-4. Avoiding Name Conflicts .................................................. 12
      When Name Conflicts Do Not Occur .................................... 15
  1-5. Using Symbols As Stand-Ins for Arbitrary Forms ................. 15
  1-6. Searching for Symbols by Name ........................................ 17
  1-7.Iterating Through All Symbols of a Package ....................... 19
      What To Do If You Don’t Like LOOP ................................. 20
  1-8. Understanding COMMON LISP’s Case (In)Sensitivity ........... 21
      Style Hint: Don’t Use CamelCase! ...................................... 25
  1-9. Using Symbols As String Substitutes .................................. 26
      So, What Should You Use? ................................................ 27
  1-10. “Overloading” of Standard COMMON LISP Operators ......... 28

2. Conses, Lists, and Trees ....................................................... 31
  2-1. Understanding Conses ..................................................... 31
      List Access ..................................................................... 34
      Testing Whether Something Is a Cons or a List .................... 35
  2-2. Creating Lists ............................................................... 37
      Converting Vectors to Lists .............................................. 38
  2-3. Transposing a Matrix ....................................................... 39
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-4.</td>
<td>Using List Interpolation</td>
<td>40</td>
</tr>
<tr>
<td>2-5.</td>
<td>Adding Objects to the End of a List</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>The Tail Wagging the List</td>
<td>44</td>
</tr>
<tr>
<td>2-6.</td>
<td>“Splicing” into a List</td>
<td>45</td>
</tr>
<tr>
<td>2-7.</td>
<td>Detecting Shared Structure in Lists</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Isolating the Non-Shared Part</td>
<td>51</td>
</tr>
<tr>
<td>2-8.</td>
<td>Working with Trees</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>More Complicated Trees</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>COMMON LISP’s Standard Tree Functions</td>
<td>55</td>
</tr>
<tr>
<td>2-9.</td>
<td>Working with Stacks</td>
<td>55</td>
</tr>
<tr>
<td>2-10.</td>
<td>Implementing a Queue</td>
<td>56</td>
</tr>
<tr>
<td>2-11.</td>
<td>Destructuring and Pattern Matching</td>
<td>58</td>
</tr>
<tr>
<td>3.</td>
<td>Strings and Characters</td>
<td>61</td>
</tr>
<tr>
<td>3-1.</td>
<td>Getting the ASCII Code of a Character</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>The Other Way Around</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>The Limit</td>
<td>63</td>
</tr>
<tr>
<td>3-2.</td>
<td>Naming Characters</td>
<td>63</td>
</tr>
<tr>
<td>3-3.</td>
<td>Using Different Character Encodings</td>
<td>65</td>
</tr>
<tr>
<td>3-4.</td>
<td>Comparing Strings or Characters</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>Internationalization</td>
<td>70</td>
</tr>
<tr>
<td>3-5.</td>
<td>Escaping Characters in String Literals and Variable Interpolation</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>Is It Still a Literal?</td>
<td>72</td>
</tr>
<tr>
<td>3-6.</td>
<td>Controlling Case</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>What About Unicode?</td>
<td>74</td>
</tr>
<tr>
<td>3-7.</td>
<td>Accessing or Modifying a Substring</td>
<td>75</td>
</tr>
<tr>
<td>3-8.</td>
<td>Finding a Character or a Substring Within a String</td>
<td>77</td>
</tr>
<tr>
<td>3-9.</td>
<td>Trimming Strings</td>
<td>77</td>
</tr>
<tr>
<td>3-10.</td>
<td>Processing a String One Character at a Time</td>
<td>79</td>
</tr>
<tr>
<td>3-11.</td>
<td>Joining Strings</td>
<td>81</td>
</tr>
<tr>
<td>3-12.</td>
<td>Reading CSV Data</td>
<td>83</td>
</tr>
<tr>
<td>4.</td>
<td>Numbers and Math</td>
<td>87</td>
</tr>
<tr>
<td>4-1.</td>
<td>Using Arbitrarily Large Integers</td>
<td>87</td>
</tr>
<tr>
<td>4-2.</td>
<td>Understanding Fixnums</td>
<td>89</td>
</tr>
<tr>
<td>4-3.</td>
<td>Performing Modular Arithmetic</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>Efficiency Considerations</td>
<td>93</td>
</tr>
<tr>
<td>4-4.</td>
<td>Switching Bases</td>
<td>94</td>
</tr>
<tr>
<td>4-5.</td>
<td>Performing Exact Arithmetic with Rational Numbers</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>Various Ways of Converting Numbers to Integers</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>How Not to Use FLOOR and Friends</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>Converting Floating-Point Numbers to Rationals and Vice Versa</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>Mixing Rationals and Floats</td>
<td>101</td>
</tr>
<tr>
<td>4-6.</td>
<td>Controlling the Default Float Format</td>
<td>102</td>
</tr>
<tr>
<td>4-7.</td>
<td>Employing Arbitrary Precision Floats</td>
<td>104</td>
</tr>
<tr>
<td>4-8.</td>
<td>Working with Complex Numbers</td>
<td>106</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>4-9. Parsing Numbers</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>4-10. Testing Whether Two Numbers Are Equal</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>Don’t Ever Use EQ with Numbers!</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>4-11. Computing Angles Correctly</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>4-12. Calculating Exact Square Roots</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>5. Arrays and Vectors</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>5-1. Working with Multiple Dimensions</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>5-2. Understanding Vectors and Simple Arrays</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>5-3. Obtaining the Size of an Array</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>5-4. Providing Initial Contents</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>A Warning About Identical Objects</td>
<td>122</td>
<td></td>
</tr>
<tr>
<td>5-5. Treating Arrays As Vectors</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>5-6. Making the Length of Vectors Flexible</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>5-7. Adjusting Arrays</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>5-8. Using an Array As a “Window” into Another Array</td>
<td>129</td>
<td></td>
</tr>
<tr>
<td>5-9. Restricting the Element Type of an Array</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>Upgrading Element Types</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>5-10. Copying an Array</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>A Warning About Object Identity</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>6. Hash Tables, Maps, and Sets</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>6-1. Understanding the Basics of Hash Tables</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>Why Does GETHASH Return Two Values?</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>How Many Entries Does the Hash Table Have?</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>6-2. Providing Default Values For Hash Table Lookups</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>6-3. Removing Hash Table Entries</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>6-4. Iterating Through a Hash Table</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>Don’t Rely on Any Order!</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>Don’t Modify While You’re Iterating!</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>Can’t This Be More Concise, Please?</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>6-5. Understanding Hash Table Tests and Defining Your Own</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>What Is SXHASH For?</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>6-6. Controlling Hash Table Growth</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>6-7. Getting Rid of Hash Table Entries Automatically</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>6-8. Representing Maps As Association Lists</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>Combining Lookup and Manipulation</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>Why Would Anybody Prefer Alists over Hash Tables?</td>
<td>162</td>
<td></td>
</tr>
<tr>
<td>6-9. Representing Maps As Property Lists</td>
<td>163</td>
<td></td>
</tr>
<tr>
<td>When to Prefer Plists over Alists</td>
<td>165</td>
<td></td>
</tr>
<tr>
<td>The Plist of a Symbol</td>
<td>166</td>
<td></td>
</tr>
<tr>
<td>6-10. Working with Sets</td>
<td>166</td>
<td></td>
</tr>
<tr>
<td>Representing Sets As Hash Tables</td>
<td>169</td>
<td></td>
</tr>
<tr>
<td>Representing Sets As Bit Patterns</td>
<td>169</td>
<td></td>
</tr>
<tr>
<td>7. Sequences and Iteration</td>
<td>171</td>
<td></td>
</tr>
<tr>
<td>7-1. Filtering a Sequence</td>
<td>171</td>
<td></td>
</tr>
</tbody>
</table>
# Table of Contents

7-2. Searching a Sequence ............................................. 172
7-3. Sorting and Merging Sequences ................................. 175
7-4. Mixing Different Sequence Types .............................. 177
7-5. Re-Using a Part of a Sequence .................................. 177
7-6. Repeating Some Values Cyclically .............................. 179
    Alternatives .................................................... 181
7-7. Counting Down ..................................................... 182
7-8. Iterating over “Chunks” of a List .............................. 184
7-9. Closing over Iteration Variables .............................. 186
7-10. “Extending” Short Sequences in Iterations .................. 187
7-11. Breaking out of LOOP .......................................... 188
7-12. Making Sense of the MAP... Zoo .............................. 191
    The Sequence Variants ....................................... 194
7-13. Defining Your Own Sequence Types ......................... 194
7-14. Iterating with ITERATE ........................................ 196
7-15. Iterating with SERIES ......................................... 200
    What the Example Does ....................................... 201

8. The Lisp Reader ....................................................... 203
8-1. Employing the Lisp Reader for Your Own Code .............. 203
    Why READ Is Potentially Dangerous ......................... 205
    What READ Doesn’t Do ....................................... 205
    The Optional Arguments to READ ............................ 206
    Go Wild! ...................................................... 206
8-2. Troubleshooting Literal Object Notation ...................... 206
    This Also Applies to Strings! ............................... 208
8-3. Evaluating Forms at Read Time ................................. 208
    What to Look Out For ....................................... 210
    Alternatives .................................................. 210
8-4. Embedding Literal Arrays into Your Code .................... 211
    The Usual Warning .......................................... 212
8-5. Understanding the Different Ways to Refer to a Function .. 213
8-6. Repeating Something You Already Typed ...................... 214
    They Don’t Only Look Identical, They Are Identical! ...... 216
8-7. Safely Experimenting with Readtables ......................... 216
    Temporarily Switching to Standard IO Syntax ................ 218
8-8. Changing the Syntax Type of a Character .................... 219
    The Six Syntax Types ....................................... 220
    How to Actually Change the Syntax Type ................... 222
    Some Things Never Change .................................. 222
8-9. Creating Your Own Reader Macros .............................. 223
    What Reader Macro Functions Do ............................ 224
8-10. Working with Dispatching Macro Characters .................. 226
8-11. Preserving Whitespace ......................................... 228
Table of Contents

9. Printing .................................................. 231
   9-1. Using the Printing Primitives ......................... 231
       Printing Objects So That They Can Be Read Back in Again 235
       Shortcuts ........................................... 236
   9-2. Printing to and into Strings ....................... 237
   9-3. Printing NIL As a List ................................ 239
   9-4. Extending FORMAT Control Strings Over More Than One Line 240
   9-5. Using Functions As FORMAT Controls ............... 241
   9-6. Creating Your Own FORMAT Directives .............. 243
   9-7. Recursive Processing of FORMAT Controls .......... 245
   9-8. Controlling How Your Own Objects Are Printed .... 247
   9-9. Controlling the Pretty Printer .................... 249
   9-10. Printing Long Lists ................................ 253
   9-11. Pretty-Printing Compound Objects .................. 257
       Using the Pretty Printer from FORMAT ............... 260
   9-12. Modifying the Pretty Printer ...................... 262

10. Evaluation, Compilation, Control Flow ................. 265
    10-1. Comparing Arbitrary Lisp Objects ................. 265
        Comparing State .................................... 266
        Constants .......................................... 269
    10-2. Using Constant Variables as Keys in CASE Macros 269
    10-3. Using Arbitrary Variable Names for Keyword Parameters 271
        Keyword Names Don’t Have to Be Keywords ........ 272
        Keyword Dames Don’t Have to Be Constant ........ 273
    10-4. Creating “Static Local Variables,” Like in C .... 273
    10-5. “Preponing” the Computation of Values .......... 275
    10-6. Modifying the Behavior of Functions You Don’t Have the Source Of 278
    10-7. Swapping the Values of Variables (or Places) .... 280
    10-8. Creating Your Own Update Forms for “Places” ... 283
        Using DEFSETF ...................................... 285
        Using DEFINE-SETF-EXPANDER ..................... 286
        So, Which One Do I Use? ......................... 291
        Using DEFINE-MODIFY-MACRO ...................... 291
        Multiple-Valued Places ......................... 293
    10-9. Working with Environments ......................... 294
    10-10. Commenting Out Parts of Your Code ............... 299
        Some Notes About ; and #| .......................... 302
        How ;, #|, and Others Are Implemented ............ 302

11. Concurrency ............................................ 303
    11-1. Managing Lisp Processes .......................... 304
        Escape Hatches .................................... 307
        Threads Are Expensive ............................ 308
    11-2. Accessing Shared Resources Concurrently ........ 308
        Locks ............................................... 312
**Table of Contents**

- Atomic Operations .......................................................... 313
- More Problems ..................................................................... 314

11-3. Using Special Variables in Concurrent Programs ............ 317
  - Per-Thread Initial Bindings ............................................... 319
  - Variables That Are Always Global ...................................... 319

11-4. Communicating with Other Threads ............................... 320
  - Alternatives ..................................................................... 322

11-5. Parallelizing Algorithms Without Threads and Locks ...... 322
  - What the Example Does .................................................... 325
  - Fine-Tuning ..................................................................... 326
  - Ptrees .............................................................................. 326
  - Alternatives ..................................................................... 330

11-6. Determining the Number of Cores ............................... 330

12. Error Handling and Avoidance ..................................... 333
- 12-1. Checking Types at Run Time ....................................... 333
  - Alternatives ..................................................................... 335
- 12-2. Adding Assertions to Your Code ................................. 336
  - Disabling Assertions in “Production Code” ......................... 338
- 12-3. Defining Your Own Conditions ..................................... 338
  - How Conditions Are Printed ............................................. 340
- 12-4. Signaling a Condition .................................................. 341
  - Condition Designators ..................................................... 343
- 12-5. Handling Conditions .................................................. 344
  - Ignoring Errors .................................................................. 349
- 12-6. Providing and Using Restarts ....................................... 350
  - Visible Restarts .................................................................. 354
  - Predefined Restarts ............................................................ 355
- 12-7. Getting Rid of Warning Messages ............................... 356
- 12-8. Protecting Code from Non-Local Exits ......................... 357
  “WITH” Macros ..................................................................... 360

13. Objects, Classes, Types .................................................. 361
- 13-1. Defining Types ............................................................ 361
  - Compound Type Specifiers ............................................... 363
  - Derived Types .................................................................... 365
- 13-2. Using Classes As Types ............................................... 366
- 13-3. Writing Methods for Built-In Classes ......................... 367
- 13-4. Providing Constructors for Your Classes .................... 369
- 13-5. Marking Slots As “Private” ........................................... 372
- 13-6. Changing the Argument Precedence Order .................. 374
- 13-7. Automatically Initializing Slots on First Usage ............ 376
- 13-8. Changing and Redefining Classes on the Fly ................ 377
  - Objects Changing Their Class ........................................... 380
  - Redefining Classes ............................................................ 381
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-5. Splitting a Filename into its Component Parts</td>
<td>445</td>
</tr>
<tr>
<td>15-6. Renaming a File</td>
<td>447</td>
</tr>
<tr>
<td>Implementation-Specific Alternatives</td>
<td>449</td>
</tr>
<tr>
<td>Don’t Expect “Move” Behavior!</td>
<td>449</td>
</tr>
<tr>
<td>15-7. Deleting a File</td>
<td>450</td>
</tr>
<tr>
<td>What Does “Success” Mean Anyway?</td>
<td>451</td>
</tr>
<tr>
<td>15-8. Deleting a Directory</td>
<td>452</td>
</tr>
<tr>
<td>15-9. Copying a File</td>
<td>453</td>
</tr>
<tr>
<td>15-10. Processing the Contents of a Directory Recursively</td>
<td>454</td>
</tr>
<tr>
<td>The CL-FAD Library</td>
<td>455</td>
</tr>
<tr>
<td>15-11. Getting the Pathname a Stream Is Associated With</td>
<td>456</td>
</tr>
<tr>
<td>15-12. Dealing with Symbolic Links</td>
<td>457</td>
</tr>
<tr>
<td>What If I Want the Symlinks?</td>
<td>459</td>
</tr>
<tr>
<td>15-13. Navigating a Directory Tree</td>
<td>459</td>
</tr>
<tr>
<td>15-14. Figuring Out (Source) File Locations Programmatically</td>
<td>461</td>
</tr>
<tr>
<td>15-15. Understanding Logical Pathnames</td>
<td>463</td>
</tr>
<tr>
<td>What Exactly Are Logical Pathnames?</td>
<td>467</td>
</tr>
<tr>
<td>So, Maybe Logical Pathnames Aren’t Totally Useless</td>
<td>467</td>
</tr>
<tr>
<td>16. Developing and Debugging</td>
<td>469</td>
</tr>
<tr>
<td>16-1. Embracing Lisp’s Image-Based Development Style</td>
<td>469</td>
</tr>
<tr>
<td>The Role of the Source Code</td>
<td>471</td>
</tr>
<tr>
<td>16-2. Deciding Which IDE to Use</td>
<td>471</td>
</tr>
<tr>
<td>A Brief History of Emacs (As Seen from Lisp)</td>
<td>474</td>
</tr>
<tr>
<td>Alternatives (?)</td>
<td>475</td>
</tr>
<tr>
<td>16-3. Debugging with the Debugger</td>
<td>475</td>
</tr>
<tr>
<td>Entering the Debugger Intentionally</td>
<td>478</td>
</tr>
<tr>
<td>Without SLIME</td>
<td>479</td>
</tr>
<tr>
<td>Logging Backtraces</td>
<td>480</td>
</tr>
<tr>
<td>16-4. Tracing Functions</td>
<td>481</td>
</tr>
<tr>
<td>Graphical Tracing</td>
<td>483</td>
</tr>
<tr>
<td>16-5. Stepping Through Your Code</td>
<td>484</td>
</tr>
<tr>
<td>16-6. Acquiring Information About Functions, Macros, and Variables</td>
<td>486</td>
</tr>
<tr>
<td>Accessing the HyperSpec</td>
<td>488</td>
</tr>
<tr>
<td>Cross-Reference Information</td>
<td>489</td>
</tr>
<tr>
<td>16-7. Inspecting and Modifying (Compound) Objects</td>
<td>489</td>
</tr>
<tr>
<td>The SLIME Inspector</td>
<td>492</td>
</tr>
<tr>
<td>16-8. Browsing Your Lisp Image</td>
<td>492</td>
</tr>
<tr>
<td>Alternatives</td>
<td>493</td>
</tr>
<tr>
<td>16-9. “Undoing” Definitions</td>
<td>494</td>
</tr>
<tr>
<td>16-10. Distinguishing Your IDE’s Streams</td>
<td>497</td>
</tr>
<tr>
<td>16-11. Utilizing the REPL’s Memory</td>
<td>499</td>
</tr>
<tr>
<td>IDE History Features</td>
<td>500</td>
</tr>
<tr>
<td>16-12. Recording Your Work</td>
<td>501</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>17. Optimization</td>
<td>503</td>
</tr>
<tr>
<td>17-1. Understanding the Importance of the Right Algorithms</td>
<td>504</td>
</tr>
<tr>
<td>17-2. Deciding If and Where to Optimize</td>
<td>505</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>508</td>
</tr>
<tr>
<td>Statistical Profiling</td>
<td>510</td>
</tr>
<tr>
<td>Some Things Cannot Be Profiled</td>
<td>512</td>
</tr>
<tr>
<td>Where to Go From Here</td>
<td>512</td>
</tr>
<tr>
<td>A Warning About Empirical Data</td>
<td>513</td>
</tr>
<tr>
<td>What Does TIME Do?</td>
<td>513</td>
</tr>
<tr>
<td>CPUs Can Deceive You</td>
<td>513</td>
</tr>
<tr>
<td>17-3. Asking the Compiler to Optimize</td>
<td>515</td>
</tr>
<tr>
<td>Implementation-Defined Optimize Qualities</td>
<td>517</td>
</tr>
<tr>
<td>Can I Have This Global, Please?</td>
<td>517</td>
</tr>
<tr>
<td>Compilers Aren’t Wizards</td>
<td>518</td>
</tr>
<tr>
<td>17-4. Obtaining Optimization Hints from the Compiler</td>
<td>518</td>
</tr>
<tr>
<td>17-5. Helping the Compiler by Providing Type Information</td>
<td>522</td>
</tr>
<tr>
<td>Generic Operations</td>
<td>525</td>
</tr>
<tr>
<td>Boxing</td>
<td>525</td>
</tr>
<tr>
<td>How to Declare Types</td>
<td>526</td>
</tr>
<tr>
<td>The Scope of Type Declarations</td>
<td>527</td>
</tr>
<tr>
<td>Declaring the Return Type of Forms</td>
<td>528</td>
</tr>
<tr>
<td>Type Inference</td>
<td>528</td>
</tr>
<tr>
<td>Pitfalls</td>
<td>529</td>
</tr>
<tr>
<td>17-6. Reducing “Consing”</td>
<td>529</td>
</tr>
<tr>
<td>“Consing” and the Heap</td>
<td>532</td>
</tr>
<tr>
<td>Reusing Data Structures</td>
<td>533</td>
</tr>
<tr>
<td>Destructive Functions</td>
<td>534</td>
</tr>
<tr>
<td>“Hidden” Consing</td>
<td>535</td>
</tr>
<tr>
<td>“Tuning” the Garbage Collector</td>
<td>536</td>
</tr>
<tr>
<td>17-7. Using the Stack Instead of the Heap</td>
<td>536</td>
</tr>
<tr>
<td>Multiple Values</td>
<td>540</td>
</tr>
<tr>
<td>17-8. Optimizing Recursive Functions</td>
<td>542</td>
</tr>
<tr>
<td>17-9. Helping the Compiler with Alternative Implementation Strategies</td>
<td>544</td>
</tr>
<tr>
<td>17-10. Avoiding Repeated Computations</td>
<td>547</td>
</tr>
<tr>
<td>17-11. Avoiding Function Calls</td>
<td>549</td>
</tr>
<tr>
<td>Alternatives That Aren’t Really Alternatives</td>
<td>553</td>
</tr>
<tr>
<td>The NOSINLINE Declaration and Compiler Macros</td>
<td>553</td>
</tr>
<tr>
<td>17-12. Utilizing the Disassembler</td>
<td>553</td>
</tr>
<tr>
<td>Inline Assembly Code</td>
<td>557</td>
</tr>
<tr>
<td>17-14. Optimizing Array Access</td>
<td>558</td>
</tr>
<tr>
<td>17-15. Comparing Different Implementations</td>
<td>560</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>18. Libraries</td>
<td>563</td>
</tr>
<tr>
<td>18-1. Organizing Your Code</td>
<td>563</td>
</tr>
<tr>
<td>Components</td>
<td>564</td>
</tr>
<tr>
<td>Dependencies</td>
<td>565</td>
</tr>
<tr>
<td>How Does ASDF Find System Definitions?</td>
<td>567</td>
</tr>
<tr>
<td>Additional Information</td>
<td>568</td>
</tr>
<tr>
<td>What About the Names?</td>
<td>568</td>
</tr>
<tr>
<td>Advanced Usage of ASDF</td>
<td>568</td>
</tr>
<tr>
<td>18-2. Employing Open Source Libraries</td>
<td>569</td>
</tr>
<tr>
<td>Installing QUICKLISP</td>
<td>570</td>
</tr>
<tr>
<td>Using QUICKLISP for Your Own Code</td>
<td>571</td>
</tr>
<tr>
<td>18-3. Creating a Project’s Skeleton Automatically</td>
<td>571</td>
</tr>
<tr>
<td>18-4. Avoiding Wheel-Reinvention</td>
<td>572</td>
</tr>
<tr>
<td>18-5. Using Libraries to Write Portable Code</td>
<td>574</td>
</tr>
<tr>
<td>18-6. Utilizing Regular Expressions</td>
<td>576</td>
</tr>
<tr>
<td>Regex Syntax</td>
<td>577</td>
</tr>
<tr>
<td>Scanners</td>
<td>578</td>
</tr>
<tr>
<td>Convenience Features</td>
<td>579</td>
</tr>
<tr>
<td>Modifying and Dissecting Strings</td>
<td>581</td>
</tr>
<tr>
<td>More Information</td>
<td>581</td>
</tr>
<tr>
<td>18-7. Obtaining Data via HTTP</td>
<td>581</td>
</tr>
<tr>
<td>Parsing HTML</td>
<td>583</td>
</tr>
<tr>
<td>18-8. Creating Dynamic Web Sites with Lisp</td>
<td>584</td>
</tr>
<tr>
<td>Generating HTML</td>
<td>588</td>
</tr>
<tr>
<td>Web Frameworks</td>
<td>589</td>
</tr>
<tr>
<td>19. Interfacing with Other Languages</td>
<td>591</td>
</tr>
<tr>
<td>19-1. Calling C Functions from Lisp</td>
<td>592</td>
</tr>
<tr>
<td>How the FFI Finds and Loads Shared Libraries</td>
<td>594</td>
</tr>
<tr>
<td>How the FFI Calls C Functions</td>
<td>596</td>
</tr>
<tr>
<td>How the FFI Converts Between C and Lisp Types</td>
<td>598</td>
</tr>
<tr>
<td>The “stdcall Problem”</td>
<td>599</td>
</tr>
<tr>
<td>19-2. Working with C Pointers</td>
<td>600</td>
</tr>
<tr>
<td>Typed Pointers</td>
<td>602</td>
</tr>
<tr>
<td>19-3. Accessing and Generating C Arrays</td>
<td>604</td>
</tr>
<tr>
<td>Giving C Access to Lisp Arrays</td>
<td>606</td>
</tr>
<tr>
<td>19-4. Handling C Structs and Unions</td>
<td>607</td>
</tr>
<tr>
<td>Passing Structs by Value</td>
<td>611</td>
</tr>
<tr>
<td>19-5. Converting Between Lisp and C Strings</td>
<td>611</td>
</tr>
<tr>
<td>19-6. Calling Lisp Functions from C</td>
<td>614</td>
</tr>
<tr>
<td>19-7. Generating FFI Code Automatically</td>
<td>615</td>
</tr>
<tr>
<td>19-8. Embedding C in COMMON LISP</td>
<td>617</td>
</tr>
<tr>
<td>19-9. Calling C++ from Lisp</td>
<td>618</td>
</tr>
<tr>
<td>Automating the Process</td>
<td>620</td>
</tr>
<tr>
<td>ECL and CLASP</td>
<td>622</td>
</tr>
</tbody>
</table>
# Table of Contents

19-10. Using **JAVA** from Lisp ............................................. 622  
   Alternatives .............................................................. 625  
19-11. Reading and Writing JSON .......................................... 628  
19-12. Reading and Writing XML ........................................... 631  
19-13. Using **PROLOG** from **COMMON LISP** ............................ 634  

20. Graphical User Interfaces .............................................. 637  
20-1. Using a Web Browser as the GUI for Your Lisp Program ......... 638  
   What the Example Does ................................................. 641  
20-2. Building Applications with the “**Lisp Toolkit**” ................. 642  
   What the Example Does ................................................. 645  
20-3. Creating **COMMON LISP** GUIs Through **JAVA** .................. 646  
   What the Example Does ................................................ 648  
   A Better Example ....................................................... 648  
20-4. Using CAPI to Build Graphical User Interfaces .................. 652  
   What the Example Does ................................................. 655  
20-5. Using Lisp on Mobile Devices ....................................... 657  
   Alternatives .............................................................. 660  

21. Persistence ............................................................... 661  
21-1. Serializing Your Data ............................................... 662  
21-2. Accessing Relational Databases .................................... 664  
21-3. Keeping Your Database in RAM ..................................... 668  
21-4. Using a Lisp Object Database ..................................... 672  

22. The World Outside ...................................................... 677  
22-1. Accessing Environment Variables ................................... 677  
   The Windows Registry and Application-Specific Settings .......... 679  
22-2. Accessing the Command-Line Arguments ......................... 679  
22-3. Querying Your Lisp for Information About Its Environment .... 680  
22-4. Delivering Stand-Alone Executables .............................. 682  
   Delivering Programs with the Commercial Lisps .................. 685  
22-5. Customizing Your Lisp ............................................... 685  
   Saving an Image ........................................................ 687  
22-6. Running External Programs ......................................... 688  
22-7. Embedding Lisp ....................................................... 692  
   Creating Shared Libraries with **LISPWORKS** ..................... 694  
   Embedding **ABCL** in a **JAVA** Program .......................... 695  
22-8. Measuring Time ....................................................... 696  
22-9. Working with Dates and Times ..................................... 698  
   The **LOCAL-TIME** Library ........................................... 701  
22-10. Working with the Garbage Collector .............................. 703  
    Finalizers ..................................................................... 705  

Index .................................................................................. 709
About the Author

Edmund Weitz is well known in the Common Lisp community for his open source libraries and for being one of the organizers of the European Common Lisp Meeting. He has a PhD in mathematics and has been a freelance Common Lisp consultant for clients in the United States, Europe, and Asia since 2002. He now works as a professor for math and computer science at the University of Applied Sciences in Hamburg, Germany.
About the Technical Reviewer

Hans Hübner has been a professional programmer for three decades. After becoming proficient in the then-ubiquitous BASIC, he was an enthusiastic follower of the object-orientation wave, mostly programming in C++. Perl led him into the dynamic programming language field, where he eventually picked up Common Lisp and applied it in a multitude of industrial projects for clients in various industries. He is now leading a small company developing Lisp-based applications in the healthcare space.
Preface

I don’t know about you, but for me, discovering COMMON LISP and learning to program in it was kind of a revelation. I came to it around 1999 when I was earning my money with “mainstream” languages and tried out various others just out of interest. I was immediately hooked and I’ve never looked back. Although it is probably fair to say that COMMON LISP is nowadays a niche language, I managed to make a living out of programming in it for more than a decade and met lots of others who did the same. I still think that COMMON LISP is the most enjoyable language to work with in spite of its inevitable shortcomings. If you have bought this book, or if you are considering buying it, you probably have the same feeling.

However, more than 15 years ago, the situation compared to today was quite bleak. There weren’t even half as many COMMON LISP implementations as now; there were only very few ready-to-use and useful open source libraries, and there was no infrastructure. Plus, while for something like, say, PERL, there were quite a few high-quality and recent books available, for COMMON LISP all books were, although at least partly quite good, simply outdated and didn’t reflect the current situation anymore. One of the many things that changed for the better in this century is that one really good new Lisp book finally appeared (more on it shortly) and I believe that it was a very important contribution to Lisp’s resurgence. And I also believe that—although some claim that nobody reads book anymore—more good books are needed.

I was thus glad that Apress asked me to author a Lisp book and I accepted immediately. I started the Common Lisp Cookbook project many years ago (which, unfortunately, petered out due to various reasons) and at that time may have had hopes to turn it into a “real” book. But looking back to it today, I’m actually happy this didn’t happen, as I feel that with several more years of experience and given the current state of the COMMON LISP “world,” I’m in a much better position to write a useful book.

Who This Book Is For

In order to get the most out of this book, you should be somewhat familiar with COMMON LISP. This means you should have at least one COMMON LISP compiler on your hard drive which you should have already used to write a few simple programs. You should know how to define functions with DEFUN and how to call them;
you should know how to declare local variables with LET and how to use them; you should have used some basic arithmetic and list functions like + or LIST; you should not be surprised by strings or characters; you should be familiar with forms like IF for branches or DOTIMES for iteration. And so on. It would also be good if you knew your way around in the HyperSpec (see page XXIV), so that you can look up things that you don’t know.

Apress, the publisher of this book, has also published Peter Seibel’s *Practical Common Lisp* (2005), which in my opinion is the best from-scratch introduction to COMMON LISP that has been written so far. If you haven’t read it yet, please give it a try—it’s really worth it! The book you have in your hands right now assumes you have understood at least the basic concepts of most of what Peter explains in his book, and I’ll sometimes refer you to specific sections of it instead of repeating things.¹

### Who This Book Is (Probably) Not For

Due to its very nature as a collection of “recipes,” this book can’t be considered a textbook. If you’re totally new to COMMON LISP, you shouldn’t read this book as an introduction. Rather, I’d recommend starting with Peter’s book, which I already mentioned in the previous paragraph. Once you’ve mastered it, please return to this book for the next steps.

Also, if you’re a seasoned professional and have used COMMON LISP for years, this book is likely not for you.² I’d be pleased if you’d find something in this book you didn’t know yet. (And chances are you will, as COMMON LISP is such a huge language. I myself learned or relearned a lot when writing the book.) But you’d probably have been able to figure this out yourself if necessary.

### How to Read This Book

In a way, like all authors, I’d wish that you’d read the whole book from cover to cover. But that’s pretty unlikely, given how it is organized. You’ll probably look at individual recipes or chapters whenever the need arises. Still, some topics are too complex to cram into a few pages. That’s why I tried to provide lots of cross-references to related recipes or places where more information can be found. I also endeavored to create a comprehensive index, which will hopefully help you to find things which are “hidden” in not so obvious places.

But even with all these pointers, it might be a good idea to at least skim the table of contents for a rough idea about what’s covered in the book. And maybe you’ll read

---

¹In my opinion, it is definitely worth buying the book, but Peter has been nice enough to make its contents available online for free. So, if you just need a refresher here and there, bookmark a link to [http://www.gigamonkeys.com/book/](http://www.gigamonkeys.com/book/).

²And I’m wondering if you even made it this far in the preface.
some of the chapters just out of interest, and not only because there’s a deadline that you have to meet…

What’s In and What Not

If you look at the table of contents, you’ll notice that the book covers a lot of different areas, from core features of the language, to third party additions, to tasks like debugging. And it is, to my knowledge, the first COMMON LISP book which discusses topics such as environment access, logical pathnames, foreign function interfaces, multithreading, graphical user interfaces, persistence, delivery of executables, and optimization.

But although the book is quite big, it still can’t do full justice to all of COMMON LISP and to all interesting open source libraries written in the last decades. I tried to find a healthy mixture and let myself be guided by features I’ve used myself. My apologies in advance if I missed exactly those parts you were looking for.

Having said that, there are two areas where the book is scarce on purpose. One is the absolute basics. I mentioned Practical Common Lisp and I’ve been trying not to repeat what Peter already explained thoroughly. There will be some inevitable overlaps, but in general, if there’s something you’re missing, you should first check if there isn’t already a chapter in Peter’s book devoted to that topic.

(Still, there will be basics explained and discussed here, which you’ll already know very well, and you might be wondering why I “wasted” time and space for them. Well, this is a book of recipes that also aims to serve as some kind of reference. And each reader is different. If a recipe is called Understanding Foo, and you already understand Foo, just skip it…)

The other area this book doesn’t spend a lot of time on is macros. They make cameo appearances here and there, but they don’t get their own chapter. The reason is that there are at least two books solely devoted to this topic. One is the “classic” On Lisp by Paul Graham (Prentice Hall, 1993) and the other is Let over Lambda by Doug Hoyte (Lulu.com, 2008). Once you’ve read and understand these two books, and Peter’s book, there won’t be anything left I could teach you about macros.

Finally, from giving university lectures in mathematics for several years I’ve learned that it’s not always a good idea from a pedagogical perspective to provide all the

---

3I tend to think that for every COMMON LISP pro, there are some parts of the language they’ve never used and probably don’t even know (or care) about.
4Well, at least two. For example, I also rarely mention structures (the ones defined by DEFSTRUCT) anywhere in the book because I view them as an historical artefact superseded by CLOS. And there are certainly other things I only treat negligently.
6Be careful with these books, though, if you’re new to COMMON LISP. In my opinion, some people have the tendency to be overly impressed by macros and to (mis)use them just because they seem to be “cool.”
“dirty” details at once. More often, it is beneficial to “lie” a bit or to at least “hide” the full truth. I’ve certainly not written anything in this book that’s actually wrong (at least not on purpose), but I might have left out some rare special cases in order to be more concise or in order to not spoil the overall narrative. If you’re in doubt, you can always consult the standard (more on this next).

The HyperSpec

COMMON LISP is a language defined by a standard. That means there’s an official document which describes in painstaking detail how the individual parts of the language are expected to work and interact. Not only is this standard very detailed and comprehensive, it is also (for a technical document) surprisingly easy to read. The bad news is that the “real” standard is only available as an ugly PDF through ANSI’s online store for a rather steep price. (At least that was the situation when I wrote this preface and it has been like that for years.)

The good news is that there’s a wonderful online version of the standard which is actually much more useful than the PDF because of its extensive cross-referencing and additional material. From a legal point of view, this so-called HyperSpec is not “the standard,” but for all practical purposes it is. This is why, whenever I’m talking about “the standard” in this book, I’m referring to this online version (which also goes by the acronym CLHS).

You should have the standard on your hard drive already, but if you don’t, you can find it at http://www.lispworks.com/documentation/common-lisp.html. And see also page 488.

Which Implementation

Unlike many other popular programming languages (but like C++, for example) there are lots of different implementations of COMMON LISP. As long as we’re talking about standard COMMON LISP, it shouldn’t matter which implementation you’re using. But this book also talks extensively about non-standard additions. On the one hand, it would have been a pretty bad idea to base the whole book on one particular implementation. On the other hand, I needed to actually run and test my code somehow. I eventually decided to use a mixture of different compilers and different platforms. That’s why, for example, you might see different prompts in different platforms.

7It can be found at the LISPWORKS web site because its creation (by Kent Pitman) was funded by LISPWORKS’s predecessor company, HARLEQUIN.
8There are simply too many which are quite good (see Recipe 17-15). And although one may be the most popular now, this might change over time. Also, popularity on one particular operating system doesn’t necessarily translate to popularity on other platforms.
9For what it’s worth I mostly used LISPWORKS on Windows, OS X, and Linux, SBCL on Windows and Linux, ALLEGROCL on Windows and Linux, and occasionally CLOzureCL on Linux or OS X.
recipes. Whenever possible I tried to pinpoint where your results may differ if you’re not using the same Lisp I happened to use for that recipe.

Source Code

This book doesn’t contain large pieces of code but consists mostly of small demonstration snippets. Still, if you want to save some typing, everything that can be found in the framed boxes is available for download; from the Apress web site (at www.apress.com) as well as from http://weitz.de/cl-recipes/.

The Index

I personally like books with extensive indexes and thus tried to provide for my own book an index as comprehensive as possible. This means, for example, that you won’t only find one entry for a specific COMMON LISP function (the one for the recipe where it is explained), but usually several such entries, including references to pages in other recipes where it is “just” used so that you can see it “in action.” Likewise, you might, say, find an index entry for Gaussian integers. This is not because this book explains what Gaussian integers are, but because maybe you remember one specific example and later want to look it up.

So, the general policy when compiling the index was to provide as many ways of finding stuff as possible. I hope you’ll find it useful. (And maybe you’ll even browse the index to find stuff you might have otherwise overlooked.)

For technical reasons, if an index entry refers to page n, you will in some cases have to search for a code excerpt on page n + 1 instead.

Typographical Conventions

Many of the recipes in this book will show examples where we interact with Lisp images. They’ll look like so:

```
CL-USER 1 > (* 6 7)
42
```

Here the “CL-USER 1 >” part is the prompt of the REPL, so (* 6 7) is what you’re supposed to enter and 42 is what your Lisp is expected to return. Depending on the implementation we’re using for a specific example, the prompt can also be something different, such as simply “*” and nothing else.

---

I also used other Lisps, like CLISP, ECL, or ABCL for specific recipes.

Also, my technical reviewer used OS X as his main testing platform, so that platform should be covered even if I myself used it only sparingly.
The output of the REPL has sometimes been edited a bit in order to increase readability or to save space.

Although COMMON LISP behaves as if it were case insensitive (see Recipe 1-8), we’ll follow the usual convention to use uppercase letters (as in “LIST”) in running text.

The names of programming languages and libraries are set in small caps (as in “QUICKLISP”). Wherever possible I tried to follow typical usage regarding the distribution of lowercase and uppercase letters in those names but I’m sure I got it wrong in some cases.

Acknowledgements

I would like to thank my technical reviewer, Hans Hübner, (who, by the way, is also one of the best and knowledgeable hackers I personally know) for his great work. He found some really embarrassing mistakes…

Arthur Lemmens (another great Lisp hacker and a good friend) gave the whole book another very thorough review. And both Hans and Arthur not only found errors but also provided many useful suggestions to improve the text.

Any remaining errors in this book (and whatever else you might not like about it) are, of course, my responsibility.

Zach Beane has done a lot of things for the COMMON LISP community already, but I think his most important contribution to date has been the invention of QUICKLISP (see Recipe 18-2), which made finding and installing Lisp libraries so much easier than it was before. Without QUICKLISP, writing a book like this, which talks about so many open source libraries from diverse sources, would have been almost impossible. Zach was also nice enough to review the recipes which pertain to his own code.

Thanks to all the implementors and to all authors of open source libraries for their contributions to the COMMON LISP ecosystem. Thanks to the members of the PRO mailing list and specifically to Thomas Burdick for helping me with some tricky issues with respect to the standard. Thanks to Wes Henderson for his help with the MOCL recipe.

Since I started to work with COMMON LISP “in earnest,” I’ve met a lot of Lisp hackers through conferences or joint projects. I’ve learned tons of stuff from them because these guys (and girls) are all very intelligent and knowledgeable. But apart from that they are also just nice persons of whom I’m glad I made the acquaintance. In addition to Arthur, Hans, and Zach, I’d like to mention Jans Aasman, Marc

10 For example, he maintains the blog aggregator Planet Lisp at http://planet.lisp.org/ and publishes useful and interesting tips on his blog Common Lisp Tips at http://lisptips.com/. He’s also a prolific creator of open source software. See more at http://xach.com/lisp/.

11 See https://mailman.common-lisp.net/listinfo/pro.
Battyani, Dave Fox, Luke Gorrie, Jeremy Jones, Nick Levine, David McClain, Scott McKay, Christophe Rhodes, Martin Simmons, Dimitri Simos, Robert Strandh, Ernst van Wanig, Didier Verna, Gail Zacharias, and the late Dan Weinreb. (And I’m sure I forgot some. My apologies!)

The whole book was written and typeset using Donald Knuth’s \TeX. All graphics, except for a few screenshots in Chapter 20, were generated using Till Tantau’s \TikZ and \PGF packages.

Many thanks to the people at Apress who made this all happen, especially to Mark Powers.

And finally, my love to Heike and Mouna (without whom the book would probably have been finished earlier but life would certainly be less fun).