

# PLEAC: Objective CAML

## Foreword

Following the spirit of the *Perl Cookbook*<sup>1</sup> by Tom Christiansen and Nathan Torkington, published by O'Reilly, the *PLEAC Project*<sup>2</sup> aims to gather fans of programming in order to implement the solutions in other programming languages.

In this document, you'll find an implementation of the solutions of the Perl Cookbook in the *Objective CAML* language.

This work is released under the terms of the *GNU Free Documentation License*<sup>3</sup>, except for the Perl portions which are copyrighted by O'Reilly & Associates yet made freely available.

<sup>1</sup> <http://www.oreilly.com/catalog/cookbook/>

<sup>2</sup> <http://pleac.sourceforge.net/>

<sup>3</sup> <http://www.gnu.org/copyleft/fdl.html>

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Strings</b>   | <b>10</b> |
| 1.1      | Accessing Substrings                                   | 10        |
| 1.2      | Establishing a Default Value                           | 11        |
| 1.3      | Exchanging Values Without Using Temporary Variables    | 12        |
| 1.4      | Converting Between ASCII Characters and Values         | 12        |
| 1.5      | Processing a String One Character at a Time            | 13        |
| 1.6      | Reversing a String by Word or Character                | 16        |
| 1.7      | Expanding and Compressing Tabs                         | 17        |
| 1.8      | Expanding Variables in User Input                      | 17        |
| 1.9      | Controlling Case                                       | 17        |
| 1.10     | Interpolating Functions and Expressions Within Strings | 18        |
| 1.11     | Indenting Here Documents                               | 19        |
| 1.12     | Reformatting Paragraphs                                | 19        |
| 1.13     | Escaping Characters                                    | 20        |
| 1.14     | Trimming Blanks from the Ends of a String              | 21        |
| 1.15     | Parsing Comma-Separated Data                           | 21        |
| 1.16     | Soundex Matching                                       | 22        |
| 1.17     | Program: fixstyle                                      | 23        |
| 1.18     | Program: psgrep  | 24        |
| <b>2</b> | <b>Numbers</b>   | <b>28</b> |
| 2.1      | Checking Whether a String Is a Valid Number            | 28        |
| 2.2      | Comparing Floating-Point Numbers                       | 28        |
| 2.3      | Rounding Floating-Point Numbers                        | 29        |
| 2.4      | Converting Between Binary and Decimal                  | 30        |
| 2.5      | Operating on a Series of Integers                      | 31        |
| 2.6      | Working with Roman Numerals                            | 32        |
| 2.7      | Generating Random Numbers                              | 33        |
| 2.8      | Generating Different Random Numbers                    | 34        |
| 2.9      | Making Numbers Even More Random                        | 34        |
| 2.10     | Generating Biased Random Numbers                       | 34        |
| 2.11     | Doing Trigonometry in Degrees, not Radians             | 35        |
| 2.12     | Calculating More Trigonometric Functions               | 35        |
| 2.13     | Taking Logarithms                                      | 35        |
| 2.14     | Multiplying Matrices                                   | 35        |
| 2.15     | Using Complex Numbers                                  | 36        |
| 2.16     | Converting Between Octal and Hexadecimal               | 37        |
| 2.17     | Putting Commas in Numbers                              | 38        |
| 2.18     | Printing Correct Plurals                               | 38        |
| 2.19     | Program: Calculating Prime Factors                     | 39        |
| <b>3</b> | <b>Dates and Times</b>                                 | <b>41</b> |
| 3.1      | Finding Today's Date                                   | 41        |
| 3.2      | Converting DMYHMS to Epoch Seconds                     | 41        |
| 3.3      | Converting Epoch Seconds to DMYHMS                     | 41        |
| 3.4      | Adding to or Subtracting from a Date                   | 42        |
| 3.5      | Difference of Two Dates                                | 42        |
| 3.6      | Day in a Week/Month/Year or Week Number                | 42        |
| 3.7      | Parsing Dates and Times from Strings                   | 42        |
| 3.8      | Printing a Date  | 43        |
| 3.9      | High-Resolution Timers                                 | 43        |

|          |  |           |
|----------|--|-----------|
| 3.10     | Short Sleeps . . . . .   | 44        |
| 3.11     | Program: hopdelta . . . . .  | 44        |
| <b>4</b> | <b>Arrays</b>  | <b>50</b> |
| 4.1      | Specifying a List In Your Program . . . . .                            | 50        |
| 4.2      | Printing a List with Commas . . . . .                                  | 50        |
| 4.3      | Changing Array Size . . . . .  | 52        |
| 4.4      | Doing Something with Every Element in a List . . . . .                 | 53        |
| 4.5      | Iterating Over an Array by Reference . . . . .                         | 55        |
| 4.6      | Extracting Unique Elements from a List . . . . .                       | 56        |
| 4.7      | Finding Elements in One Array but Not Another . . . . .                | 57        |
| 4.8      | Computing Union, Intersection, or Difference of Unique Lists . . . . . | 57        |
| 4.9      | Appending One Array to Another . . . . .                               | 58        |
| 4.10     | Reversing an Array . . . . .   | 59        |
| 4.11     | Processing Multiple Elements of an Array . . . . .                     | 59        |
| 4.12     | Finding the First List Element That Passes a Test . . . . .            | 59        |
| 4.13     | Finding All Elements in an Array Matching Certain Criteria . . . . .   | 60        |
| 4.14     | Sorting an Array Numerically . . . . .                                 | 61        |
| 4.15     | Sorting a List by Computable Field . . . . .                           | 61        |
| 4.16     | Implementing a Circular List . . . . .                                 | 63        |
| 4.17     | Randomizing an Array . . . . .   | 64        |
| 4.18     | Program: words . . . . .   | 64        |
| 4.19     | Program: permute . . . . .   | 65        |
| <b>5</b> | <b>Hashes</b>  | <b>67</b> |
| 5.1      | Adding an Element to a Hash . . . . .                                  | 67        |
| 5.2      | Testing for the Presence of a Key in a Hash . . . . .                  | 68        |
| 5.3      | Deleting from a Hash . . . . .   | 69        |
| 5.4      | Traversing a Hash . . . . .  | 69        |
| 5.5      | Printing a Hash . . . . .  | 72        |
| 5.6      | Retrieving from a Hash in Insertion Order . . . . .                    | 72        |
| 5.7      | Hashes with Multiple Values Per Key . . . . .                          | 73        |
| 5.8      | Inverting a Hash . . . . .   | 73        |
| 5.9      | Sorting a Hash . . . . .   | 75        |
| 5.10     | Merging Hashes . . . . .   | 75        |
| 5.11     | Finding Common or Different Keys in Two Hashes . . . . .               | 76        |
| 5.12     | Hashing References . . . . .   | 76        |
| 5.13     | Resizing a Hash . . . . .  | 77        |
| 5.14     | Finding the Most Common Anything . . . . .                             | 77        |
| 5.15     | Representing Relationships Between Data . . . . .                      | 77        |
| 5.16     | Program: dutree . . . . .  | 79        |
| <b>6</b> | <b>Pattern Matching</b>  | <b>82</b> |
| 6.1      | Copying and Substituting Simultaneously . . . . .                      | 84        |
| 6.2      | Matching Letters . . . . .   | 85        |
| 6.3      | Matching Words . . . . .   | 85        |
| 6.4      | Commenting Regular Expressions . . . . .                               | 86        |
| 6.5      | Finding the Nth Occurrence of a Match . . . . .                        | 87        |
| 6.6      | Matching Multiple Lines . . . . .                                      | 89        |
| 6.7      | Reading Records with a Pattern Separator . . . . .                     | 90        |
| 6.8      | Extracting a Range of Lines . . . . .                                  | 90        |
| 6.9      | Matching Shell Globs as Regular Expressions . . . . .                  | 92        |
| 6.10     | Speeding Up Interpolated Matches . . . . .                             | 93        |

|          |   |            |
|----------|---|------------|
| 6.11     | Testing for a Valid Pattern                         | 94         |
| 6.12     | Honoring Locale Settings in Regular Expressions     | 96         |
| 6.13     | Approximate Matching                                | 97         |
| 6.14     | Matching from Where the Last Pattern Left Off       | 98         |
| 6.15     | Greedy and Non-Greedy Matches                       | 99         |
| 6.16     | Detecting Duplicate Words                           | 100        |
| 6.17     | Expressing AND, OR, and NOT in a Single Pattern     | 103        |
| 6.18     | Matching Multiple-Byte Characters                   | 106        |
| 6.19     | Matching a Valid Mail Address                       | 108        |
| 6.20     | Matching Abbreviations                              | 109        |
| 6.21     | Program: urlify                                     | 110        |
| 6.22     | Program: tcgrep                                     | 111        |
| 6.23     | Regular Expression Grabbag                          | 120        |
| <b>7</b> | <b>File Access</b>                                  | <b>125</b> |
| 7.1      | Opening a File                                      | 126        |
| 7.2      | Opening Files with Unusual Filenames                | 127        |
| 7.3      | Expanding Tildes in Filenames                       | 127        |
| 7.4      | Making Perl Report Filenames in Errors              | 128        |
| 7.5      | Creating Temporary Files                            | 128        |
| 7.6      | Storing Files Inside Your Program Text              | 129        |
| 7.7      | Writing a Filter                                    | 129        |
| 7.8      | Modifying a File in Place with Temporary File       | 131        |
| 7.9      | Modifying a File in Place with -i Switch            | 132        |
| 7.10     | Modifying a File in Place Without a Temporary File  | 132        |
| 7.11     | Locking a File                                      | 133        |
| 7.12     | Flushing Output                                     | 134        |
| 7.13     | Reading from Many Filehandles Without Blocking      | 135        |
| 7.14     | Doing Non-Blocking I/O                              | 135        |
| 7.15     | Determining the Number of Bytes to Read             | 136        |
| 7.16     | Storing Filehandles in Variables                    | 136        |
| 7.17     | Caching Open Output Filehandles                     | 136        |
| 7.18     | Printing to Many Filehandles Simultaneously         | 138        |
| 7.19     | Opening and Closing File Descriptors by Number      | 138        |
| 7.20     | Copying Filehandles                                 | 139        |
| 7.21     | Program: netlock                                    | 139        |
| 7.22     | Program: lockarea                                   | 142        |
| <b>8</b> | <b>File Contents</b>                                | <b>143</b> |
| 8.1      | Reading Lines with Continuation Characters          | 145        |
| 8.2      | Counting Lines (or Paragraphs or Records) in a File | 145        |
| 8.3      | Processing Every Word in a File                     | 146        |
| 8.4      | Reading a File Backwards by Line or Paragraph       | 147        |
| 8.5      | Trailing a Growing File                             | 148        |
| 8.6      | Picking a Random Line from a File                   | 148        |
| 8.7      | Randomizing All Lines                               | 148        |
| 8.8      | Reading a Particular Line in a File                 | 149        |
| 8.9      | Processing Variable-Length Text Fields              | 151        |
| 8.10     | Removing the Last Line of a File                    | 151        |
| 8.11     | Processing Binary Files                             | 152        |
| 8.12     | Using Random-Access I/O                             | 152        |
| 8.13     | Updating a Random-Access File                       | 152        |
| 8.14     | Reading a String from a Binary File                 | 153        |

|           |   |            |
|-----------|---|------------|
| 8.15      | Reading Fixed-Length Records                                  | 155        |
| 8.16      | Reading Configuration Files                                   | 155        |
| 8.17      | Testing a File for Trustworthiness                            | 156        |
| 8.18      | Program: tailwtmp   | 157        |
| 8.19      | Program: tctee  | 158        |
| 8.20      | Program: laston   | 160        |
| <b>9</b>  | <b>Directories</b>  | <b>162</b> |
| 9.1       | Getting and Setting Timestamps                                | 162        |
| 9.2       | Deleting a File   | 163        |
| 9.3       | Copying or Moving a File                                      | 164        |
| 9.4       | Recognizing Two Names for the Same File                       | 164        |
| 9.5       | Processing All Files in a Directory                           | 165        |
| 9.6       | Globbering, or Getting a List of Filenames Matching a Pattern | 166        |
| 9.7       | Processing All Files in a Directory Recursively               | 167        |
| 9.8       | Removing a Directory and Its Contents                         | 169        |
| 9.9       | Renaming Files  | 170        |
| 9.10      | Splitting a Filename into Its Component Parts                 | 171        |
| 9.11      | Program: symirror   | 171        |
| 9.12      | Program: lst  | 172        |
| <b>10</b> | <b>Subroutines</b>  | <b>176</b> |
| 10.1      | Accessing Subroutine Arguments                                | 176        |
| 10.2      | Making Variables Private to a Function                        | 178        |
| 10.3      | Creating Persistent Private Variables                         | 178        |
| 10.4      | Determining Current Function Name                             | 178        |
| 10.5      | Passing Arrays and Hashes by Reference                        | 180        |
| 10.6      | Detecting Return Context                                      | 180        |
| 10.7      | Passing by Named Parameter                                    | 181        |
| 10.8      | Skipping Selected Return Values                               | 181        |
| 10.9      | Returning More Than One Array or Hash                         | 181        |
| 10.10     | Returning Failure   | 182        |
| 10.11     | Prototyping Functions   | 182        |
| 10.12     | Handling Exceptions   | 182        |
| 10.13     | Saving Global Values  | 183        |
| 10.14     | Redefining a Function   | 183        |
| 10.15     | Trapping Undefined Function Calls with AUTOLOAD               | 184        |
| 10.16     | Nesting Subroutines   | 184        |
| 10.17     | Program: Sorting Your Mail                                    | 184        |
| <b>11</b> | <b>References and Records</b>                                 | <b>187</b> |
| 11.1      | Taking References to Arrays                                   | 187        |
| 11.2      | Making Hashes of Arrays                                       | 188        |
| 11.3      | Taking References to Hashes                                   | 189        |
| 11.4      | Taking References to Functions                                | 189        |
| 11.5      | Taking References to Scalars                                  | 190        |
| 11.6      | Creating Arrays of Scalar References                          | 190        |
| 11.7      | Using Closures Instead of Objects                             | 191        |
| 11.8      | Creating References to Methods                                | 192        |
| 11.9      | Constructing Records  | 192        |
| 11.10     | Reading and Writing Hash Records to Text Files                | 195        |
| 11.11     | Printing Data Structures                                      | 195        |
| 11.12     | Copying Data Structures                                       | 196        |

|           |   |            |
|-----------|---|------------|
| 11.13     | Storing Data Structures to Disk . . . . .                                 | 196        |
| 11.14     | Transparently Persistent Data Structures . . . . .                        | 197        |
| 11.15     | Program: Binary Trees . . . . .   | 197        |
| <b>12</b> | <b>Packages, Libraries, and Modules</b>                                   | <b>200</b> |
| 12.1      | Defining a Module's Interface . . . . .                                   | 202        |
| 12.2      | Trapping Errors in require or use . . . . .                               | 202        |
| 12.3      | Delaying use Until Run Time . . . . .                                     | 203        |
| 12.4      | Making Variables Private to a Module . . . . .                            | 203        |
| 12.5      | Determining the Caller's Package . . . . .                                | 203        |
| 12.6      | Automating Module Clean-Up . . . . .                                      | 204        |
| 12.7      | Keeping Your Own Module Directory . . . . .                               | 204        |
| 12.8      | Preparing a Module for Distribution . . . . .                             | 205        |
| 12.9      | Speeding Module Loading with SelfLoader . . . . .                         | 206        |
| 12.10     | Speeding Up Module Loading with Autoloader . . . . .                      | 206        |
| 12.11     | Overriding Built-In Functions . . . . .                                   | 206        |
| 12.12     | Reporting Errors and Warnings Like Built-Ins . . . . .                    | 207        |
| 12.13     | Referring to Packages Indirectly . . . . .                                | 207        |
| 12.14     | Using h2ph to Translate C #include Files . . . . .                        | 208        |
| 12.15     | Using h2xs to Make a Module with C Code . . . . .                         | 211        |
| 12.16     | Documenting Your Module with Pod . . . . .                                | 212        |
| 12.17     | Building and Installing a CPAN Module . . . . .                           | 213        |
| 12.18     | Example: Module Template . . . . .  | 213        |
| 12.19     | Program: Finding Versions and Descriptions of Installed Modules . . . . . | 214        |
| <b>13</b> | <b>Classes, Objects, and Ties</b>   | <b>216</b> |
| 13.1      | Constructing an Object . . . . .  | 217        |
| 13.2      | Destroying an Object . . . . .  | 218        |
| 13.3      | Managing Instance Data . . . . .  | 218        |
| 13.4      | Managing Class Data . . . . .   | 220        |
| 13.5      | Using Classes as Structs . . . . .  | 220        |
| 13.6      | Cloning Objects . . . . .   | 221        |
| 13.7      | Calling Methods Indirectly . . . . .                                      | 222        |
| 13.8      | Determining Subclass Membership . . . . .                                 | 222        |
| 13.9      | Writing an Inheritable Class . . . . .                                    | 223        |
| 13.10     | Accessing Overridden Methods . . . . .                                    | 224        |
| 13.11     | Generating Attribute Methods Using AUTOLOAD . . . . .                     | 224        |
| 13.12     | Solving the Data Inheritance Problem . . . . .                            | 224        |
| 13.13     | Coping with Circular Data Structures . . . . .                            | 225        |
| 13.14     | Overloading Operators . . . . .   | 226        |
| 13.15     | Creating Magic Variables with tie . . . . .                               | 230        |
| <b>14</b> | <b>Database Access</b>  | <b>235</b> |
| 14.1      | Making and Using a DBM File . . . . .                                     | 235        |
| 14.2      | Emptying a DBM File . . . . .   | 236        |
| 14.3      | Converting Between DBM Files . . . . .                                    | 237        |
| 14.4      | Merging DBM Files . . . . .   | 237        |
| 14.5      | Locking DBM Files . . . . .   | 237        |
| 14.6      | Sorting Large DBM Files . . . . .   | 238        |
| 14.7      | Treating a Text File as a Database Array . . . . .                        | 239        |
| 14.8      | Storing Complex Data in a DBM File . . . . .                              | 240        |
| 14.9      | Persistent Data . . . . .   | 242        |
| 14.10     | Executing an SQL Command Using DBI and DBD . . . . .                      | 243        |

|           |   |            |
|-----------|---|------------|
| 14.11     | Program: ggh - Grep Netscape Global History                 | 244        |
| <b>15</b> | <b>User Interfaces</b>                                      | <b>247</b> |
| 15.1      | Parsing Program Arguments                                   | 247        |
| 15.2      | Testing Whether a Program Is Running Interactively          | 247        |
| 15.3      | Clearing the Screen   | 247        |
| 15.4      | Determining Terminal or Window Size                         | 248        |
| 15.5      | Changing Text Color   | 248        |
| 15.6      | Reading from the Keyboard                                   | 249        |
| 15.7      | Ring the Terminal Bell                                      | 250        |
| 15.8      | Using POSIX termios   | 250        |
| 15.9      | Checking for Waiting Input                                  | 251        |
| 15.10     | Reading Passwords   | 251        |
| 15.11     | Editing Input   | 252        |
| 15.12     | Managing the Screen   | 253        |
| 15.13     | Controlling Another Program with Expect                     | 255        |
| 15.14     | Creating Menus with Tk                                      | 257        |
| 15.15     | Creating Dialog Boxes with Tk                               | 259        |
| 15.16     | Responding to Tk Resize Events                              | 260        |
| 15.17     | Removing the DOS Shell Window with Windows Perl/Tk          | 261        |
| 15.18     | Program: Small termcap program                              | 261        |
| 15.19     | Program: tkshufflepod                                       | 262        |
| <b>16</b> | <b>Process Management and Communication</b>                 | <b>266</b> |
| 16.1      | Gathering Output from a Program                             | 266        |
| 16.2      | Running Another Program                                     | 267        |
| 16.3      | Replacing the Current Program with a Different One          | 268        |
| 16.4      | Reading or Writing to Another Program                       | 268        |
| 16.5      | Filtering Your Own Output                                   | 269        |
| 16.6      | Preprocessing Input   | 271        |
| 16.7      | Reading STDERR from a Program                               | 272        |
| 16.8      | Controlling Input and Output of Another Program             | 274        |
| 16.9      | Controlling the Input, Output, and Error of Another Program | 274        |
| 16.10     | Communicating Between Related Processes                     | 275        |
| 16.11     | Making a Process Look Like a File with Named Pipes          | 278        |
| 16.12     | Sharing Variables in Different Processes                    | 281        |
| 16.13     | Listing Available Signals                                   | 281        |
| 16.14     | Sending a Signal  | 282        |
| 16.15     | Installing a Signal Handler                                 | 283        |
| 16.16     | Temporarily Overriding a Signal Handler                     | 283        |
| 16.17     | Writing a Signal Handler                                    | 284        |
| 16.18     | Catching Ctrl-C   | 284        |
| 16.19     | Avoiding Zombie Processes                                   | 284        |
| 16.20     | Blocking Signals  | 285        |
| 16.21     | Timing Out an Operation                                     | 285        |
| 16.22     | Program: sigrand  | 286        |
| <b>17</b> | <b>Sockets</b>  | <b>292</b> |
| 17.1      | Writing a TCP Client  | 292        |
| 17.2      | Writing a TCP Server  | 293        |
| 17.3      | Communicating over TCP                                      | 293        |
| 17.4      | Setting Up a UDP Client                                     | 294        |
| 17.5      | Setting Up a UDP Server                                     | 296        |

|           |   |            |
|-----------|---|------------|
| 17.6      | Using UNIX Domain Sockets                             | 297        |
| 17.7      | Identifying the Other End of a Socket                 | 298        |
| 17.8      | Finding Your Own Name and Address                     | 298        |
| 17.9      | Closing a Socket After Forking                        | 299        |
| 17.10     | Writing Bidirectional Clients                         | 299        |
| 17.11     | Forking Servers                                       | 300        |
| 17.12     | Pre-Forking Servers                                   | 301        |
| 17.13     | Non-Forking Servers                                   | 302        |
| 17.14     | Writing a Multi-Homed Server                          | 305        |
| 17.15     | Making a Daemon Server                                | 306        |
| 17.16     | Restarting a Server on Demand                         | 307        |
| 17.17     | Program: backsniff                                    | 308        |
| 17.18     | Program: fwdport                                      | 309        |
| <b>18</b> | <b>Internet Services</b>                              | <b>313</b> |
| 18.1      | Simple DNS Lookups                                    | 313        |
| 18.2      | Being an FTP Client                                   | 314        |
| 18.3      | Sending Mail  | 316        |
| 18.4      | Reading and Posting Usenet News Messages              | 317        |
| 18.5      | Reading Mail with POP3                                | 320        |
| 18.6      | Simulating Telnet from a Program                      | 321        |
| 18.7      | Pinging a Machine                                     | 324        |
| 18.8      | Using Whois to Retrieve Information from the InterNIC | 327        |
| 18.9      | Program: expn and vrfy                                | 330        |
| <b>19</b> | <b>CGI Programming</b>                                | <b>332</b> |
| 19.1      | Writing a CGI Script                                  | 332        |
| 19.2      | Redirecting Error Messages                            | 333        |
| 19.3      | Fixing a 500 Server Error                             | 334        |
| 19.4      | Writing a Safe CGI Program                            | 335        |
| 19.5      | Making CGI Scripts Efficient                          | 335        |
| 19.6      | Executing Commands Without Shell Escapes              | 336        |
| 19.7      | Formatting Lists and Tables with HTML Shortcuts       | 337        |
| 19.8      | Redirecting to a Different Location                   | 340        |
| 19.9      | Debugging the Raw HTTP Exchange                       | 341        |
| 19.10     | Managing Cookies                                      | 342        |
| 19.11     | Creating Sticky Widgets                               | 344        |
| 19.12     | Writing a Multiscreen CGI Script                      | 346        |
| 19.13     | Saving a Form to a File or Mail Pipe                  | 349        |
| 19.14     | Program: chemiserie                                   | 351        |
| <b>20</b> | <b>Web Automation</b>                                 | <b>359</b> |
| 20.1      | Fetching a URL from a Perl Script                     | 359        |
| 20.2      | Automating Form Submission                            | 360        |
| 20.3      | Extracting URLs                                       | 360        |
| 20.4      | Converting ASCII to HTML                              | 363        |
| 20.5      | Converting HTML to ASCII                              | 366        |
| 20.6      | Extracting or Removing HTML Tags                      | 366        |
| 20.7      | Finding Stale Links                                   | 368        |
| 20.8      | Finding Fresh Links                                   | 369        |
| 20.9      | Creating HTML Templates                               | 370        |
| 20.10     | Mirroring Web Pages                                   | 373        |
| 20.11     | Creating a Robot                                      | 373        |

|       |   |     |
|-------|---|-----|
| 20.12 | Parsing a Web Server Log File . . . . . | 375 |
| 20.13 | Processing Server Logs . . . . .        | 375 |
| 20.14 | Program: <code>htmlsub</code> . . . . . | 379 |
| 20.15 | Program: <code>hrefsub</code> . . . . . | 380 |

# 1 Strings

```
let string = "\\n" (* two characters, \ and an n *)
let string = "Jon 'Maddog' Orwant" (* literal single quotes *)

(*-----*)

let string = "\n" (* a "newline" character *)
let string = "Jon \"Maddog\" Orwant" (* literal double quotes *)

let a = "
  This is a multiline here document
  terminated by one double quote
"
```

## 1.1 Accessing Substrings

```
let value = String.sub string offset count
let value = String.sub string offset (String.length string - offset)
(* or *)
let value = sub_end string offset
(* using *)
let sub_end string offset =
  String.sub string offset (String.length string - offset)

(*-----*)

(* get a 5-byte string, skip 3, then grab 2 8-byte strings,
   then the rest *)

(* split at 'sz' byte boundaries *)
let rec split_every_n_chars sz = function
| "" -> []
| s ->
  try
    let (beg, rest) = String.sub s 0 sz, sub_end s sz in
    beg :: split_every_n_chars sz rest
  with _ -> [s]

let fivers = split_every_n_chars 5 string

(* chop string into individual characters *)
let chars = List.map (fun x -> x.[0]) (split_every_n_chars 1 string)

(*-----*)

let string = "This is what you have";;
(* Indexes are left to right. There is no possibility to index
   directly from right to left *)
(* "T" *)
let first = String.sub string 0 1
(* "is" *)
let start = String.sub string 5 2
```

```

(* "you have" *)
let rest  = String.sub string 13 (String.length string - 13)
(* "e" *)
let last  = String.sub string (String.length string - 1) 1
(* "have" *)
let theend = String.sub string (String.length string - 4) 4
(* "you" *)
let piece  = String.sub string (String.length string - 8) 3

(*-----*)

let string = "This is what you have";;
Printf.printf "%s" string ;
(* This is what you have *)

(* Change "is" to "wasn't" *)
let string = (String.sub string 0 5) ^ "wasn't" ^ sub_end string 7
(* This wasn't what you have *)

(* This wasn't wonderful *)
let string = (String.sub string 0 (String.length string -12)) ^
  "ondrous";;

(* delete first character *)
let string = String.sub string 1 (String.length string - 1)
(* his wasn't wondrous *)

(* delete last 10 characters *)
let string = String.sub string 0 (String.length string -10)
(* his wasn' *)

```

## 1.2 Establishing a Default Value

```

(* Because OCaml doesn't have the same notion of truth or
   definedness as Perl, most of these examples just can't be
   done as they are in Perl.  Some can be approximated via the
   use of options, but remember, unbound variables are not
   automatically assigned the value of None -- the variable has
   to have been explicitly bound to None (or Some x) beforehand. *)

(* use b if b is not None, else use c *)
let a = match b with None -> c | _ -> b;;

(* set x to y if x is currently None *)
let x = match x with None -> y | _ -> x;;

(* Note that these are much closer to Perl's notion of definedness
   than truth. *)

(* We can set foo to either bar or "DEFAULT VALUE" in one of two ways
   keep foo as a string option. *)
let foo = match bar with Some x -> bar | _ -> Some "DEFAULT VALUE";;

```

```

(* Use foo as a string. *)
let foo = match bar with Some x -> x | _ -> "DEFAULT VALUE";;

let dir = if Array.length Sys.argv > 1 then argv.(1) else "/tmp";;

(* None of the other examples really make sense in OCaml terms... *)

```

### 1.3 Exchanging Values Without Using Temporary Variables

```

let var1, var2 = var2, var1

(*-----*)

let temp    = a
let a      = b
let b      = temp

(*-----*)

let a      = "alpha"
let b      = "omega"
let a, b = b, a    (* the first shall be last -- and versa vice *)

(*-----*)

let alpha, beta, production = "January", "March", "August"

(* move beta      to alpha,
   move production to beta,
   move alpha     to production *)

let alpha, beta, production = beta, production, alpha

```

### 1.4 Converting Between ASCII Characters and Values

```

let num = Char.code char
let char = Char.chr num

(*-----*)

(* char and int are distinct datatypes in OCaml *)
printf "Number %d is character %c\n" num (Char.chr num)
(* Number 101 is character e *)

(*-----*)

(* convert string to list of chars *)
let explode s =
  let rec f acc = function
    | -1 -> acc
    | k -> f (s.[k] :: acc) (k - 1)
  in f [] (String.length s - 1)

```

```

(* convert list of chars to string *)
let implode l =
  let s = String.create (List.length l) in
  let rec f n = function
    | x :: xs -> s.[n] <- x; f (n + 1) xs
    | [] -> s
  in f 0 l

(* ascii is list of ints. *)
let ascii = List.map Char.code (explode string)
let string = implode (List.map Char.ord ascii)

(*-----*)

let ascii_value = Char.code 'e'    (* now 101 *)
let character   = Char.chr 101     (* now 'e' *)

(*-----*)

printf "Number %d is character %c\n" 101 (Char.chr 101)

(*-----*)

let ascii_character_numbers =
  List.map Char.code (explode "sample")
let () =
  List.iter (printf "%d ") ascii_character_numbers;
  printf "\n"
(* 115 97 109 112 108 101 *)

(* same *)
let word = implode (List.map Char.chr ascii_character_numbers)
let word = implode (List.map Char.chr [115; 97; 109; 112; 108; 101])

let () = printf "%s\n" word
(* sample *)

(*-----*)

let hal = "HAL"
let ascii = List.map Char.code (explode hal)
let ascii = List.map (( + ) 1) ascii (* add one to each ASCII value *)
let ibm = implode (List.map Char.chr ascii)
let () = printf "%s\n" ibm          (* prints "IBM" *)

```

## 1.5 Processing a String One Character at a Time

```

(* One can split a string into an array of character, or corresponding
   ASCII codes as follows, but this is not necessary to process the
   strings a character at a time: *)

let array_of_chars =
  Array.init (String.length s) (fun i -> s.[i])

```

```

let array_of_codes =
  Array.init (String.length s) (fun i -> Char.code s.[i])

(* or one can just use String.iter *)
String.iter
  (fun i ->
    (* do something with s.[i], the ith char of the string *)
    ()) s;;

(* The following function can be used to return a list of all unique
   keys in a hashtable *)

let keys h =
  let k = Hashtbl.fold (fun k v b -> k::b) h [] in
  (* filter out duplicates *)
  List.fold_left (fun b x -> if List.mem x b then b else x::b) [] k

(* and this function is a shorthand for adding a key, value pair
   to a hashtable *)

let ( <<+ ) h (k,v) = Hashtbl.add h k v

let seen = Hashtbl.create 13
let s = "an apple a day"
let array_of_chars = Array.init (String.length s) (fun i -> s.[i])
let () =
  Array.iter (fun x -> seen <<+ (x,1)) array_of_chars;
  print_string "unique chars are:\t";
  List.iter print_char (List.sort compare (keys seen));
  print_newline ()

(* or, without the unnecessary and inefficient step of converting the
   string into an array of chars *)
let seen = Hashtbl.create 13
let s = "an apple a day"
let () =
  String.iter (fun x -> seen <<+ (x,1)) s;
  print_string "unique chars are:\t";
  List.iter print_char (List.sort compare (keys seen));
  print_newline ()

(* To compute the simple 31-bit checksum of a string *)
let cksum s =
  let sum = ref 0 in
  String.iter (fun x -> sum := !sum + (Char.code x)) s;
  !sum
(*
# cksum "an apple a day";;
- : int = 1248
*)

(* to emulate the SysV 16-bit checksum, we will first write two routines

```

```

    sort of similar to Perl's (<>), that will return the contents of a
    file either as a list of strings or as a single string - not that the
    list of strings version throws away the \n at the end of each line *)

let slurp_to_list filename =
  let ic = open_in filename
  and l = ref [] in
  let rec loop () =
    let line = input_line ic in
    l := line::!l;
    loop () in
  try loop () with End_of_file -> close_in ic; List.rev !l

let slurp_to_string filename =
  let ic = open_in filename
  and buf = Buffer.create 4096 in
  let rec loop () =
    let line = input_line ic in
    Buffer.add_string buf line;
    Buffer.add_string buf "\n";
    loop () in
  try loop () with End_of_file -> close_in ic; Buffer.contents buf

let cksum16 fn =
  let addString sum s =
    let sm = ref sum in
    String.iter (fun c -> sm := !sm + (Char.code c)) (s ^ "\n");
    !sm mod 65537 (* 216 - 1 *)in
  List.fold_left addString 0 (slurp_to_list fn)

(* or *)
let cksum16 fn =
  let sum = ref 0
  and s = slurp_to_string fn in
  String.iter (fun c -> sum := (!sum + (Char.code c)) mod 65537) s;
  !sum

(* Note: slowcat as written is meant to be run from the command line,
   not in the toplevel *)

#!/usr/local/bin/ocaml
(* slowcat - emulate a s l o w line printer *)
(* usage: slowcat [-DELAY] [files ...] *)
#load "unix.cma";;

(* make sure you have the code for the slurp_to_string function in this
   file as well... *)

let _ =
  let delay, fs =
    try (float_of_string Sys.argv.(1)),2
    with Failure _ -> 1.,1 in
  let files = Array.sub Sys.argv fs (Array.length Sys.argv - fs) in

```

```

let print_file f =
  let s = slurp_to_string f in
  String.iter
    (fun c ->
      print_char c;
      ignore(Unix.select [] [] [] (0.005 *. delay))) s in
  Array.iter print_file files

```

## 1.6 Reversing a String by Word or Character

(\* To flip the characters of a string, we can use a for loop.  
 Note that this version does not destructively update the string. \*)

```

let reverse s =
  let len = String.length s - 1 in
  let s' = String.create (len + 1) in
  for i = 0 to len do
    s'.[i] <- s.[len - i]
  done;
  s'

```

(\* to modify the string in place, we can use the following function \*)

```

let reverse_in_place s =
  let len = String.length s - 1 in
  for i = 0 to (len + 1) / 2 - 1 do
    let t = s.[i] in
    s.[i] <- s.[len - i];
    s.[len - i] <- t
  done

```

(\* To reverse the words in a string, we can use String.concat,  
 Str.split and List.rev. Note that this requires us to load in the  
 Str module --

use '#load "str.cma"' in\* the toplevel, or be sure to include  
 str.cma in the list of object files when compiling your code.

E.g.:

```

ocamlc other options str.cma other files -or-
ocamlpt other options str.cmxa other files *)

```

```

let reverse_words s =
  String.concat " " (List.rev (Str.split (Str.regexp " ") s))

```

```

let is_palindrome s =
  s = reverse s

```

(\* We do need to do a bit more work that Perl to find the big  
 palindromes in /usr/share/dict/words ... \*)

```

let findBigPals () =
  let words = open_in "/usr/share/dict/words" in
  let rec loop () =
    let w = input_line words in
    if String.length w > 5 && w = reverse w then
      print_endline w;

```



```

    uncapitalize. *)

let big = String.uppercase little      (* "bo peep" -> "BO PEEP" *)
let little = String.lowercase big      (* "JOHN" -> "john" *)
let big = String.capitalize little     (* "bo" -> "Bo" *)
let little = String.uncapitalize big   (* "BoPeep" -> "boPeep" *)

(* Capitalize each word's first character, downcase the rest *)
let text = "thIS is a loNG liNE"
let text = String.capitalize (String.lowercase text)
let () = print_endline text

(*
This is a long line
*)

(* To do case insensitive comparisons *)
if String.uppercase a = String.uppercase b then
  print_endline "a and b are the same\n";;

let randcap fn =
  let s = slurp_to_string fn in
  for i = 0 to String.length s - 1 do
    if Random.int 100 < 20 then
      String.blit (String.capitalize (String.sub s i 1)) 0 s i 1
    done;
  print_string s

(*
# randcap "/etc/passwd";;

##
# User DataBase
#
# Note That this file is consulted when the system is running in
# single-user mode. At other times this information is handled by one
# or more of:
# lookupd DirectoryServices
# By default, lookupd gets information from NetInfo, so this file will
# not be consulted unless you have changed lookupd's configuration.
# This file is used while in single user mode.
#
# To use this file for normal authentication, you may enable it with
# /Applications/Utilities/Directory Access.
##

< ... snip ... >
*)

```

## 1.10 Interpolating Functions and Expressions Within Strings

```

(* Again, because of OCaml's type-safe nature, actual interpolation
cannot be done inside of strings -- one must use either string

```

```

concatenation or sprintf to get the results we're looking for. *)

let phrase = "I have " ^ (string_of_int (n+1)) ^ " guanacos."
let phrase = sprintf "I have %d guanacos." (n+1)

```

### 1.11 Indenting Here Documents

```

#load "str.cma";;
let var = Str.global_replace (Str.regexp "[\t ]+") "" "\
  your text
  goes here
"

```

### 1.12 Reformatting Paragraphs

```

(* We can emulate the Perl wrap function with the following function: *)
let wrap width s =
  let l = Str.split (Str.regexp " ") s in
  Format.pp_set_margin Format.str_formatter width;
  Format.pp_open_box Format.str_formatter 0;
  List.iter
    (fun x ->
      Format.pp_print_string Format.str_formatter x;
      Format.pp_print_break Format.str_formatter 1 0;) l;
  Format.flush_str_formatter ()

(*
# let st = "May I say how lovely you are looking today... this wrap-
ping has done wonders for your figure!\n";;
val st : string =
  "May I say how lovely you are looking today... this wrap-
ping has done wonders for your figure!\n"

# print_string (wrap 50 st);;
May I say how lovely you are looking today...
this wrapping has done wonders for your figure!

# print_string (wrap 30 st);;
May I say how lovely you are
looking today... this
wrapping has done wonders for
your figure!
*)

(* Note that this version doesn't allow you to specify an opening or
standard indentation (I am having trouble getting the Format module
to behave as I think it should...). However, if one only wants to
print spaces there instead of arbitrary line leaders, we can use the
following version *)

let wrap ?(lead=0) ?(indent=0) width s =
  let l = Str.split (Str.regexp " ") s in
  Format.pp_set_margin Format.str_formatter width;

```

```

Format.pp_open_box Format.str_formatter 0;
Format.pp_print_break Format.str_formatter lead indent;
List.iter
  (fun x ->
    Format.pp_print_string Format.str_formatter x;
    Format.pp_print_break Format.str_formatter 1 indent;) l;
Format.flush_str_formatter ()

(*
# print_string (wrap 20 st);;
May I say how
lovely you are
looking today...
this wrapping has
done wonders for
your figure!
- : unit = ()

# print_string (wrap ~lead:6 ~indent:2 20 st);;
  May I say how
  lovely you are
  looking today...
  this wrapping has
  done wonders for
  your figure!

# print_string (wrap ~lead:2 20 st);;
  May I say how
  lovely you are
  looking today...
  this wrapping has
  done wonders for
  your figure!
*)

```

### 1.13 Escaping Characters

```
(* The Str module is deistrubuted with the standard Ocaml compiler
suit but it is not automatically pulled in by the command line
interpreter or the compilers.
```

The "#load" line is only needed if you are running this in the command interpreter.

If you are using either of the ocaml compilers, you will need to remove the "#load" line and link in str.cmx in the final compile command. \*)

```
#load "str.cma";;

open Str

let escape charlist str =
```



```
(Array.of_list (parse_csv line))
```

## 1.16 Soundex Matching

```
let soundex =
  let code_1 = Char.code '1' in
  let code_A = Char.code 'A' in
  let code_Z = Char.code 'Z' in

  let trans = Array.make (code_Z - code_A + 1) 0 in
  let add_letters number letters =
    let add letter =
      trans.(Char.code letter - code_A) <- (number + code_1) in
    String.iter add letters in
  Array.iteri add_letters [| "BFPV"; "CGJKQSXZ"; "DT"; "L"; "MN"; "R" |];

  fun ?(length=4) s ->
    let slength = String.length s in
    let soundex = String.make length '0' in
    let rec loop i j last =
      if i < slength && j < length then begin
        let code = Char.code (Char.uppercase s.[i]) in
        if code >= code_A && code <= code_Z
        then (if j = 0
              then (soundex.[j] <- Char.chr code;
                    loop (i + 1) (j + 1) trans.(code - code_A))
              else (match trans.(code - code_A) with
                    | 0 -> loop (i + 1) j 0
                    | code when code <> last ->
                      soundex.[j] <- Char.chr code;
                      loop (i + 1) (j + 1) code
                    | _ -> loop (i + 1) j last))
        else loop (i + 1) j last
      end in
    loop 0 0 0;
    soundex

  (*-----*)

let code = soundex string
let codes = List.map soundex list

(*-----*)

#load "str.cma"
#load "unix.cma"

let () =
  print_string "Lookup user: ";
  let user = read_line () in
  if user <> "" then begin
    let name_code = soundex user in
    let regexp = Str.regexp ("\\([a-zA-Z_0-9]+\\)[^,]*[^a-zA-Z_0-9]+")
```

```

                                ^ "\\([a-zA-Z_0-9]+\\)" in
let passwd = open_in "/etc/passwd" in
try
  while true do
    let line = input_line passwd in
    let name = String.sub line 0 (String.index line ':') in
    let {Unix.pw_gecos=gecos} = Unix.getpwnam name in
    let (firstname, lastname) =
      if Str.string_match regexp gecoss 0
      then (Str.matched_group 1 gecoss, Str.matched_group 2 gecoss)
      else ("", "") in
    if (name_code = soundex name
        || name_code = soundex lastname
        || name_code = soundex firstname)
    then Printf.printf "%s: %s %s\n" name firstname lastname
    done
  with End_of_file ->
    close_in passwd
end

```

### 1.17 Program: fixstyle

```

(* fixstyle - switch first set of data strings to second set *)
#load "str.cma";;

```

```

let data = Hashtbl.create 0
let keys = ref []
let () =
  let (=>) key value =
    keys := key :: !keys;
    Hashtbl.replace data key value in
  (
    "analysed"      => "analyzed";
    "built-in"     => "builtin";
    "chastized"    => "chastised";
    "commandline" => "command-line";
    "de-allocate" => "deallocate";
    "dropin"       => "drop-in";
    "hardcode"     => "hard-code";
    "meta-data"    => "metadata";
    "multicharacter" => "multi-character";
    "multiway"     => "multi-way";
    "non-empty"    => "nonempty";
    "non-profit"   => "nonprofit";
    "non-trappable" => "nontrappable";
    "pre-define"   => "predefine";
    "preextend"    => "pre-extend";
    "re-compiling" => "recompiling";
    "reenter"     => "re-enter";
    "turnkey"     => "turn-key";
  )
let pattern_text =

```

```

    "\\(" ^ (String.concat "\\|" (List.map Str.quote !keys)) ^ "\\)"
let pattern = Str.regexp pattern_text

let args = ref (List.tl (Array.to_list Sys.argv))

let verbose =
  match !args with
  | "-v" :: rest -> args := rest; true
  | _ -> false

let () =
  if !args = []
  then (Printf.eprintf "%s: reading from stdin\n" Sys.argv.(0);
        args := ["-"])

let replace_all text line file =
  String.concat ""
    (List.map
     (function
      | Str.Text s -> s
      | Str.Delim s ->
        if verbose
        then Printf.eprintf "%s => %s at %s line %d.\n"
          s (Hashtbl.find data s) file line;
          Hashtbl.find data s)
     (Str.full_split pattern text))

let () =
  List.iter
    (fun file ->
     let in_channel =
       if file = "-"
       then stdin
       else open_in file in
     let line = ref 0 in
     try
       while true do
         let text = input_line in_channel in
         incr line;
         print_endline (replace_all text !line file)
       done
     with End_of_file ->
       close_in in_channel)
    !args

```

## 1.18 Program: psgrep

```

#!/usr/bin/ocaml
(* psgrep - print selected lines of ps output by
   compiling user queries into code *)
#load "unix.cma";;

(* Warning: In order to closely approximate the original recipe, this

```

example performs dynamic evaluation using the toplevel. This mechanism is undocumented and not type-safe. Use at your own risk.

The "psgrep" utility, defined below, can be used to filter the results of the command-line "ps" program. Here are some examples:

Processes whose command names start with "sh":

```
% psgrep 'String.sub command 0 2 = "sh"'
```

Processes running with a user ID below 10:

```
% psgrep 'uid < 10'
```

Login shells with active ttys:

```
% psgrep "command.[0] = '-'" 'tty <> "?"'
```

Processes running on pseudo-ttys:

```
% psgrep 'String.contains "pqrst" tty.[0]'
```

Non-superuser processes running detached:

```
% psgrep 'uid > 0 && tty = "?"'
```

Huge processes that aren't owned by the superuser:

```
% psgrep 'vsz > 50000' 'uid <> 0'
```

\*)

```
(* Eval recipe thanks to Clément Capel. *)
```

```
let () = Toploop.initialize_toplevel_env ()
```

```
let eval text = let lexbuf = (Lexing.from_string text) in
```

```
  let phrase = !Toploop.parse_toplevel_phrase lexbuf in
```

```
  ignore (Toploop.execute_phrase false Format.std_formatter phrase)
```

```
let get name = Obj.obj (Toploop.getvalue name)
```

```
let set name value = Toploop.setvalue name (Obj.repr value)
```

```
(* Type for "ps" results. *)
```

```
type ps =
```

```
  {f : int; uid : int; pid : int; ppid : int; pri : int; ni : string;
```

```
  vsz : int; rss : int; wchan : string; stat : string; tty : string;
```

```
  time : string; command : string}
```

```
(* Based on the GNU ps from Debian Linux. Other OSs will most likely  
require changes to this format. *)
```

```
let parse_ps_line line =
```

```
  Scanf.sscanf line "%d %d %d %d %d %s %d %d %6s %4s %10s %4s %s@\\000"
```

```
  (fun f uid pid ppid pri ni vsz rss wchan stat tty time command ->
```

```
    {f=f; uid=uid; pid=pid; ppid=ppid; pri=pri; ni=ni;
```

```
    vsz=vsz; rss=rss; wchan=wchan; stat=stat; tty=tty;
```

```
    time=time; command=command})
```

```

let eval_predicate ps pred =
  (* Use "eval" to initialize each variable's name and type,
     then use "set" to set a value. *)
  eval "let f = 0;;";          set "f" ps.f;
  eval "let uid = 0;;";       set "uid" ps.uid;
  eval "let pid = 0;;";       set "pid" ps.pid;
  eval "let ppid = 0;;";      set "ppid" ps.ppid;
  eval "let pri = 0;;";       set "pri" ps.pri;
  eval "let ni = \"\"";       set "ni" ps.ni;
  eval "let vsz = 0;;";       set "vsz" ps.vsz;
  eval "let rss = 0;;";       set "rss" ps.rss;
  eval "let wchan = \"\"";    set "wchan" ps.wchan;
  eval "let stat = \"\"";     set "stat" ps.stat;
  eval "let tty = \"\"";      set "tty" ps.tty;
  eval "let time = \"\"";     set "time" ps.time;
  eval "let command = \"\"";  set "command" ps.command;
  (* Evaluate expression and return result as boolean. *)
  eval ("let result = (" ^ pred ^ ");");
  (get "result" : bool)

exception TypeError of string
exception SyntaxError of string

let preds = List.tl (Array.to_list Sys.argv)
let () =
  if preds = []
  then (Printf.eprintf "usage: %s criterion ...
    Each criterion is an OCaml expression involving:
    f uid pid ppid pri ni vsz rss wchan stat tty time command
    All criteria must be met for a line to be printed.
" Sys.argv.(0); exit 0)

let () =
  let proc = Unix.open_process_in "ps wwaxl" in
  try
    print_endline (input_line proc);
    while true do
      let line = input_line proc in
      let ps = parse_ps_line line in
      if List.for_all
        (fun pred ->
          try eval_predicate ps pred
            with e ->
              (* Convert exceptions to strings to avoid depending on
                 additional toplevel libraries. *)
              match Printexc.to_string e with
              | "Typecore.Error(_, _)" -> raise (TypeError pred)
              | "Syntaxerr.Error(_)"
              | "Lexer.Error(1, _)"
              | "Lexer.Error(_, _)" -> raise (SyntaxError pred)
              | "Misc.Fatal_error" -> failwith pred
              | _ -> raise e)

```

```
        preds
    then print_endline line
done
with
| End_of_file ->
    ignore (Unix.close_process_in proc)
| e ->
    ignore (Unix.close_process_in proc);
    raise e
```

## 2 Numbers

### 2.1 Checking Whether a String Is a Valid Number

```
(* Something like this must be done differently in OCaml because of its
   type-safety. Some of the tests will use regular expressions, but most
   won't. *)

let has_NonDigits s =
  try ignore (search_forward (regexp "[^0-9]") s); true
  with Not_found -> true

let is_NaturalNumber s =
  try let n = int_of_string s in n > 0 with Failure _ -> false

let is_Integer s =
  try ignore(int_of_string s); true with Failure _ -> false

let is_DecimalNumber s =
  try ignore(int_of_string s); true with Failure _ ->
    try let n = float_of_string s in (abs_float f) >= 1.
    with Failure _ -> false

let is_CFloat s =
  try ignore(float_of_string s); true
  with Failure _ -> false

let () =
  (* One of the above predicates can then be used as needed *)
  if predicate s then
    (* is a number *)
    ()
  else
    (* is not a number *)
    ()
```

### 2.2 Comparing Floating-Point Numbers

```
(* equalStr num1 num2 accuracy returns true if num1 and num2
   are equal to accuracy decimal places *)

(* done by converting to strings, a la the Perl example *)
let equalStr num1 num2 accuracy =
  let p x = sprintf "%. *f" accuracy x in
  (p num1) = (p num2)

(* Done in a more or less sane way, i.e. treating them as numbers *)
let equal num1 num2 accuracy =
  let chop x = floor (x *. (10. ** (float accuracy))) in
  (chop num1) = (chop num2);;

(*-----*)
```

```

let wage = 536
let week = 40 * wage
let () =
  Printf.printf "One week's wage is %.2f\n" ((float week) /. 100.)

```

## 2.3 Rounding Floating-Point Numbers

```

let rounded digits fl = float_of_string (sprintf "%.*f" digits fl);;

(*-----*)

let a = 0.255
let b = float_of_string (sprintf "%.2f" a)
let c = rounded 2 a

let () =
  printf "Unrounded %f\nRounded %f\nOther rounded %f\n"
    a b c;
  printf "Unrounded %f\nRounded %.2f\nOther rounded %f\n"
    a c (rounded 2 a)

(*
* Unrounded 0.255000
* Rounded 0.260000
* Other rounded 0.260000
* Unrounded 0.255000
* Rounded 0.26
* Other rounded 0.260000
*)

(*-----*)

(* To "round" to the nearest integer, use ceil, floor, or truncate.
   Note that truncate converts the float to an integer, so a conversion
   back to a float is necessary *)

let fs = [3.3; 3.5; 3.7; -. 3.3]
let () =
  printf "number\tint\tfloor\tceil\n";
  List.iter
    (fun x ->
      printf "%.1f\t%.1f\t%.1f\t%.1f\n"
        x (float (truncate x)) (floor x) (ceil x))
    fs

(*
* number    int    floor    ceil
* 3.3       3.0    3.0      4.0
* 3.5       3.0    3.0      4.0
* 3.7       3.0    3.0      4.0
* -3.3      -3.0   -4.0     -3.0
*)

```

```

(* Or if you really want an integer in column 2 *)
let () =
  printf "number\tint\tfloor\tceil\n";
  List.iter
    (fun x -> printf "%.1f\t%d\t%.1f\t%.1f\n"
      x (truncate x) (floor x) (ceil x))
    fs

(*
* number   int    floor  ceil
* 3.3      3      3.0    4.0
* 3.5      3      3.0    4.0
* 3.7      3      3.0    4.0
* -3.3     -3     -4.0   -3.0
*)

```

## 2.4 Converting Between Binary and Decimal

```

(* Two versions in each direction -- one to deal with decimal strings,
and the other to deal with decimal integers. Binary numbers will
always be strings. *)

let binStr_of_decInt i =
  let rec strip_bits i s =
    match i with
    | 0 -> s
    | _ -> strip_bits (i lsr 1) ((string_of_int (i land 0x01)) ^ s) in
  strip_bits i ""

let binStr_of_decStr i =
  let rec strip_bits i s =
    match i with
    | 0 -> s
    | _ -> strip_bits (i lsr 1) ((string_of_int (i land 0x01)) ^ s) in
  strip_bits (int_of_string i) ""

(* Of course if you have binStr_of_decInt already, it's easier to just
call binStr_of_decInt (int_of_string i) *)

(*-----*)

let decInt_of_binStr s =
  int_of_string ("0b" ^ s)

let decStr_of_binStr s =
  string_of_int (int_of_string ("0b" ^ s))

(*-----*)

let numInt = decInt_of_binStr "0110110" (* numInt = 54 *)
let numInt = decStr_of_binStr "0110110" (* numInt = "54" *)
let bin1 = binStr_of_decInt 54 (* bin1 = "110110" *)
let bin2 = binStr_of_decStr "54" (* bin2 = "110110" *)

```

## 2.5 Operating on a Series of Integers

```
(* The boring way is to use a for loop... *)
for i = low to high do
  (* Do your stuff *)
  (* Note, if what you want to do in the loop does not have have type
    unit, you need to wrap it with ignore, e.g. ignore (2 * i) *)
done

(* Or you skip the syntactic sugar and write it recursively yourself *)
let rec loop low high f =
  if low > high then
    ()
  else
    begin
      ignore (f low);
      loop (succ low) high f
    end

(* and now with stepsize different from 1 *)
let rec loopStep low high step f =
  if low > high then
    ()
  else
    begin
      ignore (f low);
      loopStep (low + step) high f
    end

(* Or, if you don't mind wasting space, you can use the useful iter
  functions *)

(* Array based *)
let makeArraySequence lo hi =
  Array.init (hi - lo + 1) (fun i -> i + lo)
let () =
  Array.iter ( your function here ) (makeArraySequence lo hi)

(* List based *)
let makeListSequence lo hi =
  let rec msHelper lo hi l =
    match (a - b) with
    0 -> b::l
    | _ -> msHelper a (b-1) (b::l) in
  msHelper lo hi []
let () =
  List.iter ( your function here ) (makeListSequence lo hi)

(*-----*)

printf "Infancy is: ";
for i = 0 to 2 do
  printf "%d " i
```

```

done;;

print_newline();;

printf "Toddling is: ";
loop 3 4 (fun i -> printf "%d " i);;

print_newline ();;

printf "Childhood is: ";
Array.iter (fun i -> printf "%d " i) (makeArraySequence 5 12);;

print_newline();;

(*
  Infancy is: 0 1 2
  Toddling is: 3 4
  Childhood is: 5 6 7 8 9 10 11 12
*)

```

## 2.6 Working with Roman Numerals

(\* Based on Groovy version by Paul King. \*)

```

let roman_map =
  [1000, "M"; 900, "CM"; 500, "D"; 400, "CD"; 100, "C"; 90, "XC";
   50, "L"; 40, "XL"; 10, "X"; 9, "IX"; 5, "V"; 4, "IV"; 1, "I"]

let roman arabic =
  let rec loop remains text map =
    match map with
    | (key, value) :: rest ->
      if remains >= key
      then loop (remains - key) (text ^ value) map
      else loop remains text rest
    | [] -> text in
  loop arabic "" roman_map

let arabic roman =
  let rec loop text sum map =
    match map with
    | (key, value) :: rest ->
      if (String.length text >= String.length value
          && String.sub text 0 (String.length value) = value)
      then (loop
            (String.sub
              text
              (String.length value)
              (String.length text - String.length value))
            (sum + key)
            map)
      else loop text sum rest
    | [] -> sum in

```

```

loop (String.uppercase roman) 0 roman_map

(*-----*)

(* Alternative version by Ken Wakita. *)
let roman arabic =
  let nstr s n = String.concat "" (Array.to_list (Array.make n s)) in
  snd (List.fold_left
        (fun (arabic, roman) (arab, rom) ->
          arabic mod arab, roman ^ (nstr rom (arabic / arab)))
        (arabic, ""))
        roman_map)

(*-----*)

let () =
  let roman_fifteen = roman 15 in
  Printf.printf "Roman for fifteen is %s\n" roman_fifteen;
  let arabic_fifteen = arabic roman_fifteen in
  Printf.printf "Converted back, %s is %d\n" roman_fifteen arabic_fifteen

(* Roman for fifteen is XV
   Converted back, XV is 15 *)

```

## 2.7 Generating Random Numbers

```

let random_int lo hi =
  (Random.int (hi - lo + 1)) + lo

let random_float lo hi =
  (Random.float (hi -. lo +. 1.)) +. lo

(*-----*)

let random_number = random_int 25 75 in
printf "%d\n" random_number

(*-----*)

let elem = arr.(Random.int (Array.length arr))

(*-----*)

let uc = Array.init 26 (fun i -> Char.chr (i+ (Char.code 'A')))
and lc = Array.init 26 (fun i -> Char.chr (i+ (Char.code 'a')))
and nums = Array.init 10 (fun i -> Char.chr (i + (Char.code '0')))
and puncs = [| '!'; '@'; '$'; '%'; '^'; '&'; '*' |]
let chars = Array.concat [uc; lc; nums; puncs]

(* to generate the random password as a char array *)
let password =
  Array.init 8 (fun i -> chars.(Random.int (Array.length chars)));;

```

```

(* to generate the random password as a string *)
let passString =
  let s = String.make 8 ' ' in
  for i=0 to 7 do
    s.[i] <- chars.(Random.int (Array.length chars))
  done;
  s

```

## 2.8 Generating Different Random Numbers

```

(* Seed the generator with an integer *)
Random.init 5;;

(* Seed the generator with an array of integers *)
Random.full_init [| 1; 2; 178653; -62 |];;

(* Automatically seed the generator in a system-dependant manner *)
Random.self_init ();;

```

## 2.9 Making Numbers Even More Random

```

(* This requires installation of the third party cryptokit library. *)
let prng = Cryptokit.Random.secure_rng
let buf = String.make 10 ' '
(* random_bytes buf pos len stores len random bytes in string buf,
   starting at position pos *)
prng#random_bytes buf 0 10;; (* buf now contains 10 random bytes *)

```

## 2.10 Generating Biased Random Numbers

```

(* Note that this will return just one of the numbers, as returning
   either one or the other would requires always constructing an array
   or a list -- this just returns a float *)

let gaussianRand () =
  let rec getW () =
    let u1 = 2. *. (Random.float 1.) -. 1.
      and u2 = 2. *. (Random.float 1.) -. 1. in
    let w = u1 *. u1 +. u2 *. u2 in
    if w >= 0. then w,u1,u2 else getW () in
  let w,u1,u2 = getW () in
  let w = sqrt((-2. *. (log w)) /. w) in
  let g2 = u1 *. w
    and g1 = u2 *. w in
  g1

(* Note that because of the way dist is used, it makes the most sense
   to return it as a sorted associative list rather than another hash
   table. *)
let weightToDist whash =
  let total = Hashtbl.fold (fun k v b -> b +. v) whash 0. in
  let dist = Hashtbl.fold (fun k v b -> (v,k)::b) whash [] in
  List.sort compare dist

```

```

let rec weightedRand dhash =
  let r = ref (Random.float 1.) in
  try
    let v,k = List.find (fun (v,k) -> r := !r -. v; !r < 0.) dhash in k
  with Not_found -> weightedRand dhash

let mean, dev = 25., 2. in
let salary = gaussianRand () *. sdev +. mean
let () = printf "You have been hired at $%.2f\n" salary

```

## 2.11 Doing Trigonometry in Degrees, not Radians

```

let pi = acos(-. 1.)
let degrees_of_radians r = 180. *. r /. pi
let radians_of_degrees d = d *. pi /. 180.

let sinDeg d = sin (radians_of_degrees d)
let cosDeg d = cos (radians_of_degrees d)

```

## 2.12 Calculating More Trigonometric Functions

```

(* cos, sin, tan, acos, asin, atan, sinh, cosh and tanh are all standard
   functions, but missing functions, such as secant can be constructed in
   the usual way... *)

```

```

let sec x = 1. /. (sin x)

```

## 2.13 Taking Logarithms

```

(* to take a natural log, use the log function *)
let log_e = log 100.

```

```

(* to take a log to base 10, use the log10 function *)
let log_10 = log10 100.

```

```

(* to take a log to an arbitrary base, use traditional identities *)
let logB base x = (log x) /. (log base)

```

## 2.14 Multiplying Matrices

```

let mmult m1 m2 =
  let dim m =
    Array.length m, Array.length m.(0) in
  let r1, c1 = dim m1
  and r2, c2 = dim m2 in
  if c1 <> r2
  then raise (Invalid_argument "Matrix dimensions don't match")
  else
    begin
      let dotP v1 v2 =
        let sum = ref 0. in
        for i = 0 to Array.length v1 - 1 do

```

```

        sum := !sum +. (v1.(i) *. v2.(i))
    done;
    !sum in
    let row m i = m.(i)
    and col m i = Array.init (Array.length m) (fun r -> m.(r).(i)) in
    let res = Array.make_matrix r1 c2 0. in
    for r = 0 to pred r1 do
        for c = 0 to pred c2 do
            res.(r).(c) <- dotP (row m1 r) (col m2 c)
        done
    done;
    res
end

```

## 2.15 Using Complex Numbers

```

(* c = a * b manually *)
type cplx = { real : float; imag : float; }
let c = {real = a.real *. b.real -. a.imag *. b.imag;
        imag = a.imag *. b.real +. b.imag *. a.real}

(*-----*)

(* c = a * b using the Complex module *)
open Complex
let c = Complex.mul a b

(* Note that we could have simply said let c = mul a b, but a later
   binding of a value to the name mul would render the complex mul
   invisible after that, Complex.mul is less ambiguous. *)

(*-----*)

let a = {real=3.; imag=5.}
let b = {real=2.; imag=(-. 2.);}
let c = {real = a.real *. b.real -. a.imag *. b.imag;
        imag = a.imag *. b.real +. b.imag *. a.real}
let () = printf "c = %f+%fi\n" c.real c.imag

(* c = 16.000000+4.000000i *)

let a = {re=3.; im=5.}
let b = {re=2.; im=(-. 2.);}
let c = mul a b
let () = printf "c = %f+%fi\n" c.re c.im

(* c = 16.000000+4.000000i *)

let d = {re=3.; im=4.}
let s = sqrt d in
let () = printf "sqrt(%.2f+%.2fi) = %.2f+%.2fi\n" d.re d.im s.re s.im

(* sqrt(3.00+4.00i) = 2.00+1.00i *)

```

## 2.16 Converting Between Octal and Hexadecimal

```
(* Since integers and strings are very different things in OCaml, we will
   represent both octal and hexadecimal values as strings *)
let oct_of_hex h =
  Printf.sprintf "%0o" (int_of_string ("0x" ^ h))
let hex_of_oct o =
  Printf.sprintf "%0x" (int_of_string ("0o" ^ o))

(* One small problem is that OCaml integers are 31 (or 63) bit values,
   if you need something larger, you can use the following for a full
   32 bits: *)
let oct_of_hex32 h =
  Printf.sprintf "%01o" (Int32.of_string ("0x" ^ h))
let hex_of_oct32 o =
  Printf.sprintf "%01x" (Int32.of_string ("0o" ^ o))

(* Or this for 64 bits: *)
let oct_of_hex64 h =
  Printf.sprintf "%0Lo" (Int64.of_string ("0x" ^ h))
let hex_of_oct64 o =
  Printf.sprintf "%0Lx" (Int64.of_string ("0o" ^ o))

(* For anything else you have to roll your own *)
let chopn n s =
  (* Chops strings into list of n byte substrings *)
  match s with
  | "" -> [""] (* avoids wierd edge case *)
  | _ ->
    let ex = (String.length s) mod n in
    let ss = if ex = 0 then s else ((String.make (n-ex) '0') ^ s) in
    let rec schopn x s l =
      match x with
      | 0 -> (String.sub s 0 n)::l
      | _ -> schopn (x-n) s ((String.sub s x n)::l) in
    schopn (String.length ss - n) ss []

let long_oct_of_hex h =
  let choppedH = chopn 6 h in
  let f x = int_of_string ("0x" ^ x) in
  String.concat ""
    (List.map (fun x -> Printf.sprintf "%08o" (f x)) choppedH)

let long_hex_of_oct o =
  let choppedO = chopn 8 o in
  let f x = int_of_string ("0o" ^ x) in
  String.concat ""
    (List.map (fun x -> Printf.sprintf "%06x" (f x)) choppedO)

(*-----*)

(* Since octal, hex and decimal are all the same internally, we don't
   need to do any explicit conversion *)
```

```

let () = printf "Gimme a number in decimal, octal, or hex: "
let num = read_int ()
let () = printf "%d %x %o\n" num num num

(*-----*)

let () = printf "Enter file permission in octal: "
let permissions =
  try read_int ()
  with Failure message -> failwith "Exiting...\n"
let () = printf "The decimal value is %d\n" permissions

```

## 2.17 Putting Commas in Numbers

```

(* This example requires the PCRE library, available at:
   http://www.ocaml.info/home/ocaml_sources.html#pcre-ocaml *)
#directory "+pcre";;
#load "pcre.cma";;

let rev_string s =
  let s' = String.copy s in
  let i = ref (String.length s - 1) in
  String.iter (fun c -> s'.[!i] <- c; decr i) s;
  s'

let commify s =
  rev_string
  (Pcre.replace ~pat:"(\\d\\d\\d)(?=\\d)(?!\\d*\\.)" ~templ:"$1,"
   (rev_string s))

(*-----*)

(* more reasonable web counter :-) *)
let () =
  Random.self_init ();
  let hits = Random.int32 21474836471 in
  Printf.printf "Your web page received %s accesses last month.\n"
    (commify (Int32.to_string hits))
(* Your web page received 1,670,658,439 accesses last month. *)

```

## 2.18 Printing Correct Plurals

```

(* Hardcoded examples can be done as follows: *)
let () =
  Printf.printf "It took %d hour%s\n" n (if n <> 1 then "s" else "");
  Printf.printf "It took %d centur%s\n" n (if n <> 1 then "ies" else "y")

(* For a more general solution, first define the rules.
   Note: the OS needs to support dynamic loading of C
   libraries for this. *)
#load "str.cma";;

let rules =

```

```

List.map (fun x -> (Str.regexp (fst x)),(snd x))
  ["\\([psc]h\\)$\\|z$", "\\0es";
   "\\(ff\\)$\\|\\(ey\\)$", "\\0s";
   "f$", "ves";
   "y$", "ies";
   "ix$", "ices";
   "ius$", "ii";
   "[sx]$", "\\0es";
   "non", "na"];;

let f w x =
  ignore(Str.search_forward (fst x) w 0);
  Str.replace_first (fst x) (snd x) w

let rec exn_map ex fn1 fn2 l =
  match l with
  | [] -> fn2
  | h::t -> try (fn1 h) with ex -> exn_map ex fn1 fn2 t

let pluralize x = (* "wish" in *)
  exn_map Not_found (f x) (x ^ "s") rules

(* Note: This next example doesn't work on the odd cases *)
let nouns = ["fish"; "fly"; "ox"; "species"; "genus"; "phylum";
             "cherub"; "radius"; "jockey"; "index"; "matrix"; "mythos";
             "phenomenon"; "formula"]

let () =
  List.iter (fun x -> printf "One %s, two %s\n" x (pluralize x)) nouns

```

## 2.19 Program: Calculating Prime Factors

```

(* Note: the OS needs to support dynamic loading of C libraries for this,
   otherwise you will need to link the nums library with the code at
   compile time. *)
#load "nums.cma";;

open Big_int

let cmd = [|"bigfact"; "8"; "9"; "96"; "2178";
           "23932200000000000000000000";
           "2500000000000000000000000000"; "17"|]

(* This will raise an exception if a nonnumeric string is in the
   argument list. *)
let argList =
  Array.map big_int_of_string (Array.sub cmd 1 ((Array.length cmd) - 1))

let factorize num =
  let two = big_int_of_int 2 and four = big_int_of_int 4 in
  let rec genFactors (i,sqi) n fList =
    if eq_big_int n unit_big_int then fList else
    if lt_big_int n sqi then ((n,1)::fList) else
    let newn = ref n and fcount = ref 0 in

```

```

while (eq_big_int (mod_big_int !newn i) zero_big_int) do
  newn := div_big_int !newn i;
  fcount := !fcount + 1;
done;
let nexti,nextsqi =
  if eq_big_int i two then
    (add_big_int i unit_big_int),
    (add_big_int sqi (add_big_int (mult_big_int i two)
                                  unit_big_int))
  else
    (add_big_int i two),
    (add_big_int sqi (add_big_int (mult_big_int i four) two)) in
genFactors (nexti,nextsqi) !newn (if !fcount = 0 then fList else
                                   ((i,!fcount)::fList)) in

genFactors (two,four) num []

let _ =
  Array.iter
  (fun n ->
    let l = factorize n in
    match l with
    | [(x,1)] -> printf "%s\tPrime!\n" (string_of_big_int x)
    | _ ->
      printf "%s\t" (string_of_big_int n);
      List.iter
        (fun (x,count) -> let sx = string_of_big_int x in
                           if count = 1 then printf "%s " sx
                           else printf "%s**%d " sx count)
        (List.rev l);
      print_newline()) argList

```

### 3 Dates and Times

```
(* The unix module acts as a thin wrapper around the standard C
   Posix API. It comes standard with the Ocaml compiler but is
   not automatically linked.
```

```
If you are not using the command line interpreter, delete the
the "#load" line *)
```

```
#load "unix.cma";;
open Unix

let t = Unix.localtime (Unix.time ())
let () = Printf.printf "Today is day %d of the current year.\n" t.tm_yday
```

#### 3.1 Finding Today's Date

```
(* Finding today's date *)

let (day, month, year) = (t.tm_mday, t.tm_mon, t.tm_year)
let () =
  Printf.printf "The current date is %04d-%02d-%02d\n"
    (1900 + year) (month + 1) day
```

#### 3.2 Converting DMYHMS to Epoch Seconds

```
(* Converting DMYHMS to Epoch Seconds
   Again, use the Unix module. *)

(* For the local timezone *)
let ttup = mktime (localtime (time ()))
let () = Printf.printf "Epoch Seconds (local): %.0f\n" (fst ttup)

(* For UTC *)
let ttup = mktime (gmtime (time ()))
let () = Printf.printf "Epoch Seconds (UTC): %.0f\n" (fst ttup)
```

#### 3.3 Converting Epoch Seconds to DMYHMS

```
#load "unix.cma";;

let time = Unix.time ()

let {Unix.tm_sec=seconds; tm_min=minutes; tm_hour=hours;
     tm_mday=day_of_month; tm_mon=month; tm_year=year;
     tm_wday=wday; tm_yday=yday; tm_isdst=isdst} =
  Unix.localtime time

let () =
  Printf.printf "Dateline: %02d:%02d:%02d-%04d/%02d/%02d\n"
    hours minutes seconds (year + 1900) (month + 1) day_of_month
```

### 3.4 Adding to or Subtracting from a Date

```
let birthtime = 96176750. (* 18/Jan/1973, 3:45:50 am *)
let interval = 5. +. (* 5 seconds *)
                17. *. 60. +. (* 17 minutes *)
                2. *. 60. *. 60. +. (* 2 hours *)
                55. *. 60. *. 60. *. 24. (* and 55 days *)
let then' = birthtime +. interval
let () =
  (* format_time is defined in section 3.8. *)
  Printf.printf "Then is %s\n" (format_time then');
  (* Then is Tue Mar 13 23:02:55 1973 *)
```

### 3.5 Difference of Two Dates

```
let bree = 361535725. (* 16 Jun 1981, 4:35:25 *)
let nat = 96201950. (* 18 Jan 1973, 3:45:50 *)

let difference = bree -. nat
let () =
  Printf.printf "There were %.f seconds between Nat and Bree\n"
    difference
  (* There were 265333775 seconds between Nat and Bree *)

let seconds = mod_float difference 60.
let difference = (difference -. seconds) /. 60.
let minutes = mod_float difference 60.
let difference = (difference -. minutes) /. 60.
let hours = mod_float difference 24.
let difference = (difference -. hours) /. 24.
let days = mod_float difference 7.
let weeks = (difference -. days) /. 7.

let () =
  Printf.printf "(%.f weeks, %.f days, %.f:%.f:%.f)\n"
    weeks days hours minutes seconds
  (* (438 weeks, 4 days, 23:49:35) *)
```

### 3.6 Day in a Week/Month/Year or Week Number

```
#load "unix.cma";;

let {Unix.tm_mday=monthday; tm_wday=weekday; tm_yday=yearday} =
  Unix.localtime date
let weeknum = yearday / 7 + 1
```

### 3.7 Parsing Dates and Times from Strings

```
#load "unix.cma";;

let epoch_seconds date =
  Scanf.sscanf date "%04d-%02d-%02d"
  (fun yyyy mm dd ->
```

```

        fst (Unix.mkttime {Unix.tm_sec=0; tm_min=0; tm_hour=0;
                          tm_mday=dd; tm_mon=mm-1; tm_year=yyyy-1900;
                          tm_wday=0; tm_yday=0; tm_isdst=false}))

let () =
  while true do
    let line = read_line () in
    try
      let date = epoch_seconds line in
      let {Unix.tm_mday=day; tm_mon=month; tm_year=year} =
        Unix.localtime date in
      let month = month + 1 in
      let year = year + 1900 in
      Printf.printf "Date was %d/%d/%d\n" month day year
    with
      | Scanf.Scan_failure _
      | End_of_file
      | Unix.Unix_error (Unix.ERANGE, "mkttime", _) ->
        Printf.printf "Bad date string: %s\n" line
  done

```

### 3.8 Printing a Date

```

#load "unix.cma";;

open Unix
open Printf

let days = [| "Sun"; "Mon"; "Tue"; "Wed"; "Thu"; "Fri"; "Sat" |]
let months = [| "Jan"; "Feb"; "Mar"; "Apr"; "May"; "Jun";
                "Jul"; "Aug"; "Sep"; "Oct"; "Nov"; "Dec" |]

let format_time time =
  let tm = localtime time in
  sprintf "%s %s %2d %02d:%02d:%02d %04d"
    days.(tm.tm_wday)
    months.(tm.tm_mon)
    tm.tm_mday
    tm.tm_hour
    tm.tm_min
    tm.tm_sec
    (tm.tm_year + 1900)

let time = fst (Unix.mktime {tm_sec=50; tm_min=45; tm_hour=3;
                             tm_mday=18; tm_mon=0; tm_year=73;
                             tm_wday=0; tm_yday=0; tm_isdst=false})
let () = printf "format_time gives: %s\n" (format_time time)

```

### 3.9 High-Resolution Timers

```

#load "unix.cma";;

let t0 = Unix.gettimeofday ()

```

```

let () = print_string "Press return when ready: "; ignore (read_line ())
let t1 = Unix.gettimeofday ()
let () = Printf.printf "You took %f seconds.\n" (t1 -. t0)

(*-----*)

let size = 500 in
let number_of_times = 100 in
let total_time = ref 0. in

for i = 1 to number_of_times do
  let array = Array.init size (fun _ -> Random.bits()) in

  let before = Unix.gettimeofday() in
  Array.stable_sort compare array ;
  let time = Unix.gettimeofday() -. before in
  total_time := !total_time +. time
done ;

Printf.printf "On average, sorting %d random numbers takes %.5f seconds\n" size (!total_time /. float number_of_times)

```

### 3.10 Short Sleeps

```

let usleep time =
  ignore (Unix.select [] [] [] time)

let () =
  while true do
    usleep 0.25;
    print_newline ();
  done

```

### 3.11 Program: hopdelta

```

#!/usr/bin/ocaml
(* hopdelta - feed mail header, produce lines
   showing delay at each hop. *)
#load "str.cma";;
#load "unix.cma";;

(* Modify this function to tweak the format of results. *)
let print_result sender recipient time delta =
  Printf.printf "%-30s %-30s %-20s  %s\n"
    sender recipient time delta

(* Produce a stream of lines from an input channel. *)
let line_stream_of_channel channel =
  Stream.from
    (fun _ -> try Some (input_line channel) with End_of_file -> None)

(* Turn a stream of lines into a stream of paragraphs, where each
   paragraph is a stream of lines. Paragraphs are delimited by one

```

```

    or more empty lines. *)
let paragraphs lines =
  let rec next para_lines i =
    match Stream.peek lines, para_lines with
    | None, [] -> None
    | Some "", [] -> Stream.junk lines; next para_lines i
    | Some "", _ | None, _ -> Some (Stream.of_list (List.rev para_lines))
    | Some line, _ -> Stream.junk lines; next (line :: para_lines) i in
  Stream.from (next [])

(* Find blocks of email headers in a stream of paragraphs. Headers
   are all assumed to have a first line starting with "From" and
   containing a '@' character. This is not very robust. *)
let header_blocks paras =
  let rec next i =
    match Stream.peek paras with
    | Some lines ->
      if (match Stream.peek lines with
          | Some line ->
            (String.length line >= 5
             && (String.sub line 0 5 = "From ")
             && (String.contains line '@'))
          | None -> false)
      then Some (Stream.next paras)
      else (Stream.junk paras; next i)
    | None -> None in
  Stream.from next

(* Pattern to detect continuation lines. *)
let continuation_regexp = Str.regexp "^[\t ]+"

(* Transform a stream of lines such that continuation lines are joined
   with previous lines by a single space. *)
let join_continuations lines =
  let rec continuations () =
    match Stream.peek lines with
    | Some line ->
      let found = ref false in
      let trimmed =
        Str.substitute_first
          continuation_regexp
          (fun _ -> found := true; "")
          line in
      if !found
      then (Stream.junk lines; " " ^ trimmed ^ continuations ())
      else ""
    | None -> "" in
  let rec next i =
    match Stream.peek lines with
    | Some line ->
      Stream.junk lines;
      Some (line ^ continuations ())
    | None -> None in

```

```

Stream.from next

(* A type for headers, where "from" contains the text of the "From"
   line, and the rest of the headers are parsed into a (key, value)
   list called "params". *)
type header = { from : string;
                params : (string * string) list }

(* Given a stream of header blocks, produce a stream of values of the
   above "header" type. *)
let headers blocks =
  let parse_from line =
    String.sub line 5 (String.length line - 5) in
  let parse_param params line =
    try
      let index = String.index line ':' in
      let key = String.sub line 0 index in
      let value =
        if String.length line > index + 2
        then
          String.sub
            line
            (index + 2)
            (String.length line - index - 2)
        else "" in
      params := (key, value) :: !params
    with
    | Not_found
    | Invalid_argument "String.sub" ->
      Printf.eprintf "Unable to parse header: %s\n" line;
      () in
  let rec next i =
    try
      let lines = Stream.next blocks in
      let lines = join_continuations lines in
      let from = parse_from (Stream.next lines) in
      let params = ref [] in
      Stream.iter (parse_param params) lines;
      Some { from = from; params = List.rev !params }
    with Stream.Failure ->
      None in
  Stream.from next

(* Combine the above stream transformers to produce a function from
   input channels to streams of headers. *)
let header_stream_of_channel channel =
  headers
    (header_blocks
      (paragraphs
        (line_stream_of_channel channel)))

(* Association list mapping month abbreviations to 0-based month
   numbers as required by Unix.mktime. *)

```

```

let months =
  ["Jan", 0; "Feb", 1; "Mar", 2; "Apr", 3; "May", 4; "Jun", 5;
   "Jul", 6; "Aug", 7; "Sep", 8; "Oct", 9; "Nov", 10; "Dec", 11]

(* Turn a time zone into an offset in minutes. Not exhaustive. *)
let parse_tz = function
  | "" | "Z" | "GMT" | "UTC" | "UT" -> 0
  | "PST" -> -480
  | "MST" | "PDT" -> -420
  | "CST" | "MDT" -> -360
  | "EST" | "CDT" -> -300
  | "EDT" -> -240
  | string ->
    Scanf.sscanf string "%c%02d%_[:]%02d"
      (fun sign hour min ->
        min + hour * (if sign = '-' then -60 else 60))

(* List of date-parsing functions from strings to epoch seconds. *)
let date_parsers =
  [
    (fun string ->
      Scanf.sscanf string "%d %s %d %d:%d:%d %s"
        (fun mday mon year hour min sec tz ->
          let mon = List.assoc mon months in
            fst (Unix.mktime
              {Unix.tm_sec=sec; tm_min=min; tm_hour=hour;
               tm_mday=mday; tm_mon=mon; tm_year=year-1900;
               tm_wday=0; tm_yday=0; tm_isdst=false})
            -. (float (parse_tz tz) *. 60.0)));
    (fun string ->
      Scanf.sscanf string "%3s, %d %s %4d %d:%d:%d %s"
        (fun wday mday mon year hour min sec tz ->
          let mon = List.assoc mon months in
            fst (Unix.mktime
              {Unix.tm_sec=sec; tm_min=min; tm_hour=hour;
               tm_mday=mday; tm_mon=mon; tm_year=year-1900;
               tm_wday=0; tm_yday=0; tm_isdst=false})
            -. (float (parse_tz tz) *. 60.0)));
    (fun string ->
      Scanf.sscanf string "%3s, %d %s %2d %d:%d:%d %s"
        (fun wday mday mon year hour min sec tz ->
          let mon = List.assoc mon months in
            fst (Unix.mktime
              {Unix.tm_sec=sec; tm_min=min; tm_hour=hour;
               tm_mday=mday; tm_mon=mon; tm_year=year;
               tm_wday=0; tm_yday=0; tm_isdst=false})
            -. (float (parse_tz tz) *. 60.0)));
  ]

(* Tries each of the above date parsers, one at a time, until one
   of them doesn't throw an exception. If they all fail, returns
   a value of 0.0. *)
let getdate string =

```

```

let result = ref 0.0 in
let parsers = ref date_parsers in
while !result = 0.0 && !parsers <> [] do
  let parse = List.hd !parsers in
  parsers := List.tl !parsers;
  try result := parse string with _ -> ()
done;
!result

(* Formats a date given in epoch seconds for display. *)
let fmtdate epoch =
  let tm = Unix.localtime epoch in
  Printf.sprintf "%02d:%02d:%02d %04d/%02d/%02d"
    tm.Unix.tm_hour tm.Unix.tm_min tm.Unix.tm_sec
    (tm.Unix.tm_year + 1900) (tm.Unix.tm_mon + 1) tm.Unix.tm_mday

(* Formats the difference between two epoch times for display. *)
let fmsdelta delta =
  let sign = if delta < 0.0 then '-' else ' ' in
  let delta = abs_float delta in
  let seconds = mod_float delta 60. in
  let delta = (delta -. seconds) /. 60. in
  let minutes = mod_float delta 60. in
  let delta = (delta -. minutes) /. 60. in
  let hours = mod_float delta 24. in
  Printf.sprintf "%c%02.f:%02.f:%02.f" sign hours minutes seconds

(* Process the header for a single email. *)
let process_header header =
  let start_from =
    try List.assoc "From" header.params
    with Not_found -> header.from in
  let start_from =
    Str.replace_first
      (Str.regexp ".*@\\([^\>]*\\).*)" "\\1" start_from in
  let start_date =
    try List.assoc "Date" header.params
    with Not_found -> "" in
  let start_date =
    Str.replace_first
      (Str.regexp " +(.*)" "" start_date in
  let then' = ref (getdate start_date) in
  print_result "Sender" "Recipient" "Time" "Delta";
  print_result "Start" start_from (fmtdate !then') "";
  let prevfrom = ref start_from in
  List.iter
    (fun (key, value) ->
      if key = "Received"
      then
        begin
          let when' =
              Str.replace_first
                (Str.regexp ".*; +\\(.*\\)$") "\\1" value in

```

```

let when' =
  Str.replace_first
    (Str.regexp " +(.*$)" "" when' in
let from' =
  try
    ignore (Str.search_forward
      (Str.regexp "from +\\([^\ ]+\\)") value 0);
    Str.matched_group 1 value
  with Not_found ->
  try
    ignore (Str.search_forward
      (Str.regexp "(\\([^\ ]*\\)") value 0);
    Str.matched_group 1 value
  with Not_found -> "" in
let from' = Str.replace_first (Str.regexp ")$") "" from' in
let by' =
  try
    ignore (Str.search_forward
      (Str.regexp "by +\\([^\ ]+\\. [^\ ]+\\)") value 0);
    Str.matched_group 1 value
  with Not_found -> "" in
let now = getdate when' in
let delta = now -. !then' in
print_result
  (if !prevfrom <> "" then !prevfrom else from')
  by'
  (fmtdate now)
  (fmdelta delta);
then' := now;
prevfrom := by';
end)
(List.rev header.params);
print_newline ();
flush stdout

(* Process all emails from standard input. *)
let () =
  Stream.iter process_header (header_stream_of_channel stdin)

```

## 4 Arrays

```
let nested = ["this"; "that"; "the"; "other"] (* string list *)

(* There is no such non-homogeneous list.
   You can do things with tuples: *)
let nested = ("this", "that", ["the"; "other"])
(* string * string * string list *)

(*-----*)

let tune = ["The"; "Star-Spangled"; "Banner"]
```

### 4.1 Specifying a List In Your Program

```
(* Note that Perl sort of munges OCaml lists and arrays into a single
   data structure. In OCaml, they are two distinct data structures, and
   one needs to learn when it is best to use lists vs. arrays. *)

(* To initialize a list *)
let l = ["quick"; "brown"; "fox"]

(* To initialize an array *)
let a = [|"quick"; "brown"; "fox"|]

(*-----*)

let words s = Str.split (Str.regexp "[ \\t]+") s
let l = words "Why are you teasing me?"

(*-----*)

let str = " The boy stood on the burning deck,
          It was as hot as glass.
          "
let f l =
  let sep = Str.regexp "[ \\t\\n]*\\(\\.+\\)" in
  List.map (fun s ->
    if (Str.string_match sep s 0) then
      Str.matched_group 1 s
    else "") l
f (Str.split (Str.regexp_string "\\n") str);;
(*
 * - : string list =
 * ["The boy stood on the burning deck,"; "It was as hot as glass."]
 *)

let data = open_in "mydatafile"
let bigarray = readlines data
```

### 4.2 Printing a List with Commas

```
let commify_series l =
```

```

let rec sepChar l =
  match l with
  | [] -> ", "
  | h :: t ->
    if String.contains h ',' then "; " else sepChar t in
  match l with
  | [] -> ""
  | h :: [] -> h
  | h1 :: h2 :: [] -> h1 ^ " and " ^ h2
  | _ ->
    let l' =
      let last :: rest = List.rev l in
      (List.rev (("and " ^ last) :: rest)) in
    String.concat (sepChar l) l'

let lists =
[
  [ "just one thing" ];
  [ "Mutt"; "Jeff" ];
  [ "Peter"; "Paul"; "Mary" ];
  [ "To our parents"; "Mother Theresa"; "God" ];
  [ "pastrami"; "ham and cheese"; "peanut butter and jelly"; "tuna" ];
  [ "recycle tired, old phrases"; "ponder big, happy thoughts" ];
  [ "recycle tired, old phrases";
    "ponder big, happy thoughts";
    "sleep and dream peacefully" ]
]

let () =
  List.iter
    (fun x -> printf "The list is: %s.\n" (commify_series x))
    lists

(*
The list is: just one thing.
The list is: Mutt and Jeff.
The list is: Peter, Paul, and Mary.
The list is: To our parents, Mother Theresa, and God.
The list is: pastrami, ham and cheese, peanut butter and jelly, and tuna.
The list is: recycle tired, old phrases and ponder big, happy thoughts.
The list is: recycle tired, old phrases; pon-
der big, happy thoughts; and sleep and dream peacefully.
*)

(* Note that if you are actually using arrays instead of
lists, you can either reuse the above code by calling
"commify_series (Array.to_list a)", or you can use the
following solution (which won't work with lists, but is
probably more efficient). *)

let commify_array a =
  let len = Array.length a in
  let rec sepChar a =

```

```

try
  for i=0 to len - 1 do
    if String.contains a.(i) ',' then raise Not_found
  done;
  ", "
with Not_found -> "; " in
match len with
| 0 -> ""
| 1 -> a.(0)
| 2 -> a.(0) ^ " and " ^ a.(1)
| _ ->
  let buf = Buffer.create 10
  and sep = sepChar a in
  for i = 0 to len - 2 do
    Buffer.add_string buf a.(i);
    Buffer.add_string buf sep;
  done;
  Buffer.add_string buf "and ";
  Buffer.add_string buf a.(len - 1);
  Buffer.contents buf

let arrays =
  [
    [ "just one thing" ];
    [ "Mutt"; "Jeff" ];
    [ "Peter"; "Paul"; "Mary" ];
    [ "To our parents"; "Mother Theresa"; "God" ];
    [ "pastrami"; "ham and cheese"; "peanut butter and jelly"; "tuna" ];
    [ "recycle tired, old phrases"; "ponder big, happy thoughts" ];
    [ "recycle tired, old phrases";
      "ponder big, happy thoughts";
      "sleep and dream peacefully" ]
  ]

let () =
  Array.iter
    (fun x -> printf "The list is: %s.\n" (commify_array x))
    arrays

```

### 4.3 Changing Array Size

(\* OK, OCaml just doesn't work with arrays the same way tha Perl does. In Ocaml, Arrays are immutable in their shape, while containing mutable contents. You can simulate this example as shown below (which only works for string arrays), or you can get resizeable arrays from a library such as extlib  
<<http://ocaml-lib.sourceforge.net/>> \*)

```

let what_about_that_array a =
  let len = Array.length a in
  printf "The array now has %d elements.\n" len;
  printf "The index of the last element is %d.\n"
    (if len=0 then 0 else len-1);

```

```

printf "Element 3 is \"%s\".\n" a.(3)

let resizeArray a s =
  (* begin stupid hack to work like the Perl example *)
  let s = s + 1 in
  (* end stupid hack to work like the Perl example *)
  assert (s >= 0);
  let len = Array.length a in
  if s = len then a else
    if s < len then
      Array.sub a 0 s
    else
      Array.append a (Array.make (s - len) "")

let people = ["Crosby"; "Stills"; "Nash"; "Young"]
let () = what_about_that_array people

(*
The array now has 4 elements.
The index of the last element is 3.
Element 3 is "Young".
*)

let people = resizeArray people 2
let () = what_about_that_array people

(*
The array now has 3 elements.
The index of the last element is 2.
Exception: Invalid_argument "index out of bounds".
*)

let people = resizeArray people 10000
let () = what_about_that_array people

(*
The array now has 10001 elements.
The index of the last element is 10000.
Element 3 is "".
*)

```

#### 4.4 Doing Something with Every Element in a List

```

Array.iter complain bad_users;;
(* Or for lists *)
List.iter complain bad_users;;

(* For the hashtable example, we'd iterate over the table itself *)
Hashtbl.iter (fun k v -> printf "%s=%s\n" k v) h;;

(* Of course if you want to iterate over the keys in lexicographic order,
then you'll need to build a list of keys, sort it, then iterate over
that *)

```

```

List.iter
  (fun x -> printf "%s=%s\n" x (Hashtbl.find env x))
  (List.sort compare (Hashtbl.fold (fun k v b -> k::b) env []));;

Array.iter
  (fun x -> if get_usage x > max_quota then complain x)
  all_users;;

(* or for lists of users *)
List.iter
  (fun x -> if get_usage x > max_quota then complain x)
  all_users;;

(* For this example, we're going to assume that the output of the who
   command is contained in the list named who, with one line of output
   per list element.

   This example requires the use of the Str module which is not loaded
   or linked by default (but is part of the standard library), at the
   toplevel, use the directive: #load "str.cma" *)

List.iter
  (fun x ->
    try
      ignore (Str.search_forward (Str.quote "tchrist") x 0);
      print_endline x;
    with Not_found -> ()) who

(* To iterate over all lines read in from some channel we would do the
   following *)

let iter_channel f ic =
  try
    while true do
      f (input_line ic)
    done
  with Not_found -> ()

(* and the example would then be written as *)
iter_channel
  (fun s ->
    let reverse s = 'let len = String.length s in
    let s' = String.create len in
    for i = 0 to len - 1 do
      s'.[len-i-1] <- s.[i]
    done;
    s' in
  (* assuming we have written a chomp workalike *)
  let s = chomp s in
  List.iter
    (fun x -> print_endline (reverse x))
    (Str.split (Str.regexp "[\t]+" s)) fh

```

```

(* In OCaml the iterator variable also is an alias for the current
   element, however, because of the functional nature of OCaml, unless
   the elements of the array are references, the only way to change them
   is by resetting the value of the array to something new -- this is
   best done using iteri *)

let a = [|1; 2; 3|];;
Array.iteri (fun i x -> a.(i) <- x-1) a;;

(* or, with references *)

let a = [| ref 1; ref 2; ref 3 |];;
Array.iter (fun x -> x := !x - 1) a;;

(* You can, of course, use map to create a new array with the desired
   contents as well *)
let a = [| 0.5; 3. |];;
let b = [|0.; 1. |];;
Array.iter (printf "%f ")
  (Array.map (( *. ) 7.) (Array.append a b));;

let strip s =
  Str.replace_first (Str.regexp "[\t\n]") ""
    (Str.replace_first (Str.regexp "[\t\n$]") "" s)

let sc, ar, h =
  strip sc,
  Array.map strip ar,
  (Hashtbl.iter (fun k v -> Hashtbl.replace h k (strip v)) h; h)

(* of course, the Hashtbl.replace already destructively updates the old
   hashtable... *)

```

## 4.5 Iterating Over an Array by Reference

```

(* iterate over elements of array in arrayref *)

Array.iter (fun x -> (* do something with x *)) !arrayref;;

for i = 0 to Array.length !arrayref - 1 do
  (* do something with !arrayref.(i) *)
done

let fruits = [| "Apple"; "Blackberry" |];;
let fruit_ref = ref fruits;;
Array.iter (printf "%s tastes good in a pie.\n") !fruit_ref;;

for i = 0 to Array.length !fruit_ref - 1 do
  printf "%s tastes good in a pie.\n" !fruit_ref.(i)
done;;

Hashtbl.add namelist "felines" (ref rogue_cats);;

```

```

Array.iter (printf "%s purrs hypnotically.\n")
  !(Hashtbl.find namelist "felines");;
print_endline "--More--\nYou are controlled.;;

for i=0 to Array.length !(Hashtbl.find namelist "felines") - 1 do
  printf "%s purrs hypnotically.\n"
    !(Hashtbl.find namelist "felines").(i)
done;;

```

## 4.6 Extracting Unique Elements from a List

```
(* For lists, the most "natural" way to do this is by walking the list
and looking for duplicates of each item *)
```

```

let rec uniquesOnly l =
  let rec contains x l =
    match l with
    | [] -> false
    | h::t -> if x = h then true else contains x t in
  match l with
  | [] -> []
  | h::t -> if contains h t then uniquesOnly t else h::(uniquesOnly t)

```

```
(* if you have a lot of duplicates, it might be better to use
List.filter *)
```

```

let rec uniquesOnly l =
  match l with
  | [] -> []
  | h::t -> h::(uniquesOnly (List.filter ((<>) h) t))

```

```
(* Or, for lists or arrays, you can use a hashtable *)
```

```
(* Straightforward *)
```

```

let uniquesOnly l =
  let seen = Hashtbl.create 17
  and uniq = ref [] in
  List.iter
    (fun x ->
      if not (Hashtbl.mem seen x) then
        (Hashtbl.add seen x 1; uniq := (x::!uniq)))
    l;
  !uniq

```

```
(* Or more likely *)
```

```

let uniquesOnly l =
  let seen = Hashtbl.create 17 in
  List.iter (fun x -> Hashtbl.replace seen x 1) l;
  Hashtbl.fold (fun k v b -> k::b) seen []

```

```
(* To apply a user function to each unique element of a list,
one would likely do something like *)
```

```

let userUnique f l =
  List.map f (uniquesOnly l)

```

```

(* Generate a list of users logged in, removing duplicates.
   Note that this example requires linking with the Unix and Str
   libraries. *)
let who () =
  let w = Unix.open_process_in "who"
  and l = ref [] in
  try
    while true do
      l := (input_line w) :: !l
    done;
    !l
  with End_of_file -> !l

let ucnt = Hashtbl.create 17;;
List.iter
  (fun x ->
    Hashtbl.replace ucnt
      (Str.replace_first (Str.regexp "[ \\t].*$") "" x) 1)
  (who ());;
let users = Hashtbl.fold (fun k v b -> k::b) ucnt [];;

printf "users logged in: %s";;
List.iter (printf "%s ") users;;

```

#### 4.7 Finding Elements in One Array but Not Another

```

(* using hashtables, like the cookbook *)
let arrayDiff a b =
  let seen = Hashtbl.create 17
  and l = ref [] in
  Array.iter (fun x -> Hashtbl.add seen x 1) b;
  Array.iter (fun x -> if not (Hashtbl.mem seen x) then l := x::!l) a;
  Array.of_list !l;;

```

#### 4.8 Computing Union, Intersection, or Difference of Unique Lists

```

let a = [ 1;3;5;6;7;8 ];;
let b = [ 2;3;5;7;9 ];;

let union = Hashtbl.create 13
and isect = Hashtbl.create 13
and diff = Hashtbl.create 13;;

(* simple solution for union and intersection *)
List.iter (fun x -> Hashtbl.add union x 1) a;;
List.iter
  (fun x ->
    Hashtbl.add
      (if Hashtbl.mem union x then isect else union) x 1) b;;
let u = Hashtbl.fold (fun k v b -> k::b) union []
and i = Hashtbl.fold (fun k v b -> k::b) isect [];;

```

```

(* Union, intersection, and symmetric difference *)
let hincr h x =
  let v = try Hashtbl.find h x with Not_found -> 0 in
  Hashtbl.replace h x (v+1);;

let count = Hashtbl.create 13;;
List.iter (fun x -> Hashtbl.add count x 1) a;;
List.iter (hincr count) b;;
let u,i,d =
  let u = Hashtbl.fold (fun k v b -> (k,v)::b) count [] in
  let i,d = List.partition(fun x -> snd x = 2) u in
  let vo l = List.map fst l in
  (vo u),(vo i),(vo d);;

```

## 4.9 Appending One Array to Another

```

(* For lists, use the @ operator for two lists, or List.concat for a
   list of lists, for arrays, use Array.append for two arrays, or
   Array.concat for a list of arrays. *)

```

```

let list1 = list1 @ list2
let array1 = Array.append array1 array2

let members = [| "Time"; "Flies" |]
let initiates = [| "An"; "Arrow" |]
let members = Array.append members initiates

```

```

(* It is easiest to write a splice workalike and then just use the new
   function much like in Perl *)

```

```

let splice ?length ?list arr off =
  let len = Array.length arr in
  let off = if off < 0 then len + off else off in
  let l,back =
    match length with
    | None -> (len - off),[|]|
    | Some l ->
      l,
      (let boff = off + l in
       try Array.sub arr boff (len - boff)
       with Invalid_argument _ -> [|]|) in
  let front = Array.sub arr 0 off
  and mid =
    match list with
    | None -> [|]|
    | Some a -> a
  and sp = Array.sub arr off l in
  sp, Array.concat [front;mid;back]

let _,members =
  splice members 2 ~length:0 ~list:(Array.append [|"Like"|] initiates);;
Array.iter (printf "%s ") members; print_newline ();;

```

```

let _,members = splice members 0 ~length:1 ~list:["Fruit"];
let _,members = splice members (-2) ~length:2 ~list:["A"; "Banana"];
Array.iter (printf "%s ") members; print_newline ();

```

#### 4.10 Reversing an Array

```

(* To reverse a list, use List.rev *)
let reversed = List.rev l

(* For an array, it is probably easiest to use Array.init *)

let revArray a =
  let len = Array.length a - 1 in
  Array.init len+1 (fun i -> a.(len - i))

let reversed = revArray a

(* Or one can use a for loop *)
for i = Array.length a - 1 downto 0 do
  (* Do something to a.(i) *)
done;;

```

#### 4.11 Processing Multiple Elements of an Array

```

(* To remove multiple elements from an array at once, one can use the
splice function from section 4.9 *)

(* Remove n elements from the front of arr *)
front,arr = splice arr 0 ~length:n;;
rear,arr = splice arr (-n);;

(* this can also be wrapped as an explicit function *)

let shift2 a = splice a 0 ~length:2
let pop2 a = splice a (-2)

(* This lets you do something like Perl's hinkey pattern matching *)
let friends = ["Peter"; "Paul"; "Mary"; "Jim"; "Tim" ]
let [|this; that|],friends = shift2 friends

let beverages = ["Dew"; "Jolt"; "Cola"; "Sprite"; "Fresca"]
let pair,beverages = pop2 beverages

```

#### 4.12 Finding the First List Element That Passes a Test

```

(* To find the first element in a list that satisfies some predicate,
just use the List.find function to return an 'a option *)

match
  (try Some (List.find (fun x -> x > 10) l)
   with Not_found -> None)
with
  | None -> (* unfound *)

```

```

    | Some x -> (* Do something with x *);;

(* Note that this is a very general form, and can be shortened in
some cases *)
let pf l =
  try
    printf "hah! Found %d!\n" (List.find (fun x -> x > 10) l)
  with
    Not_found -> "Sorry charly!\n"

(*
# pf [1;2;3;4;5;6];;
Sorry charly!

# pf [1;2;3;50;100];;
Hah! Found 50!
*)

(* To return the index of a matching element in an array, we can use
exceptions to short circuit the search *)

exception Found of int

let findi pred arr =
  Array.iteri (fun i x -> if pred x then raise (Found i)) arr;
  raise Not_found

let f arr =
  try
    findi (fun x -> x > 10) arr
  with
    | Found i -> printf "element %d is a big element - %d\n" i arr.(i)
    | Not_found -> printf "Only small values here!\n"

(*
# f [|1; 2; 3; 4; 5; 6|];;
Only small values here!

# f [|1; 2; 3; 4; 5; 60; 8; 9; 100|];;
element 5 is a big element - 60
*)

let highest_engineer =
  List.find (fun x -> x#category = "engineer") employees in
  printf "Highest paid engineer is: %s\n" highest_engineer#name

```

#### 4.13 Finding All Elements in an Array Matching Certain Criteria

```

(* To find all elements of a list that satisfy a certain predicate,
just use the List.find_all function. *)
let matching = List.find_all ( (* predicate *) ) l

(* For an array, it's likely easiest to convert the original array to a

```

```

    list, use List.find_all, and convert that list into an array. *)
let matching =
  Array.ofList (List.find_all ( (*predicate *) ) (Array.to_list a))

(* The next example requires use of the Str library, which must be
   linked in. In the toplevel environment use '#load "str.cma"'. *)

let bigs = List.find_all (fun x -> x > 1000000) nums
let pigs = List.find_all (fun x -> (Hashtbl.find users x) > 1e7)
           (Hashtbl.fold (fun k v b -> k::b) users [])

let matching =
  List.find_all
    (fun x -> Str.string_match (Str.regexp "gnat") x 0) (who ())

let engineers =
  List.find_all (fun x -> x#position = "Engineer") employees

let secondary_assistance =
  List.find_all
    (fun x -> x#income >= 26000 && x#income < 30000) applicants

```

#### 4.14 Sorting an Array Numerically

```

(* OCaml is smart enough to figure out if a list is full of numbers or
 * non-numbers, so the polymorphic compare function works just fine *)
let sorted = List.sort compare unsorted

(* note that Array.sort sorts the given array in place, so unexpected
   results can occur, e.g. *)
let sorted = Array.sort compare unsorted;;
(* results in unsorted referring to the now sorted array, and sorted
   referring to something of type unit *)

(* pids is an unsorted list of process IDs *)
let () =
  List.iter (printf "%d\n") (List.sort compare pids);
  print_endline "Select a process ID to kill:";
  let pid = read_int () in
  Unix.kill pid Sys.sigterm;
  Unix.sleep 2;
  Unix.kill pid Sys.sigterm

let descending = List.sort (fun x y -> compare y x) unsorted

```

#### 4.15 Sorting a List by Computable Field

```

(* Since compare orders tuples by first comparing the first slot, then
   if they were equal, comparing the second slot, and so on, we can sort
   by computable fields as follows: *)

let sorted =
  List.map snd

```

```

(List.sort compare (List.map (fun x-> (compute x),x) unsorted))

let ordered = List.sort (fun x y -> compare x#name y#name) employees

let () =
  List.iter (fun x -> printf "%s earns $%2f\n" x#name x#salary)
    (List.sort (fun x y -> compare x#name y#name) employees)

let sorted_employees =
  List.map snd
    (List.sort compare (List.map (fun x-> (compute x),x) unsorted)) in
  List.iter
    (fun x -> printf "%s earns $%2f\n" x#name x#salary)
      sorted_employees;
  List.iter
    (fun x ->
      if Hashtbl.mem bonus x#ssn
      then printf "%s got a bonus!\n" x#name)
      sorted_employees

let sorted =
  List.sort
    (fun x y ->
      match compare x#name y#name with
      | 0 -> compare x#age y#age
      | c -> c)
    employees

(* Assuming we have a getpwent function that returns a value of type
   users, or throws an End_of_file exception when done (not sure what
   getpwent is supposed to do), then we can write: *)

let getUsers () =
  let l = ref [] in
  try
    while true do
      l := (getpwent ())::!l
    done
  with End_of_file -> !l

let () =
  List.iter
    (fun x -> print_endline x#name)
    (List.sort (fun x y -> compare x#name y#name) (getUsers ()))

let sorted = List.sort (fun x y -> compare x.[1] y.[1]) strings

let sorted =
  List.map snd
    (List.sort compare (List.map (fun x -> (String.length x),x) strings))

let sorted_fields =
  List.map snd

```

```

(List.sort compare
  (List.map
    (fun x ->
      (try
        ignore (Str.search_forward (Str.regexp "[0-9]+") x 0);
        int_of_string (Str.matched_string x)
        with Not_found -> max_int),x)
      strings))

let passwd () =
  let w = Unix.open_process_in "cat /etc/passwd"
  and l = ref [] in
  try
    while true do
      l := (input_line w) :: !l
    done;
    !l
  with End_of_file -> !l

(* For illustration purposes, we provide a function to return the
   (non-comment) contents of /etc/passwd *)
let passwd () =
  let w = Unix.open_process_in "cat /etc/passwd"
  and l = ref [] in
  try
    while true do
      l := (input_line w)::!l
    done;
    !l
  with End_of_file ->
    List.filter (fun x -> x.[0] <> '#') !l

let sortedPasswd =
  List.map (fun Some x -> snd x)
    (List.sort compare
      (List.filter (function Some x -> true | None -> false)
        (List.map
          (fun x ->
            match Str.split (Str.regexp ":") x with
            | name :: _ :: uid :: gid :: t ->
              Some ((gid, uid, name), x)
            | _ -> None)
            (passwd ())))))

```

## 4.16 Implementing a Circular List

```

(* To get a true circular list, one can use the let rec construct *)
let rec processes = 1 :: 2 :: 3 :: 4 :: 5 :: processes
let () =
  while true do
    let process::processes = process in
    printf "Handling process %d\n" process;
    Unix.sleep 2;

```

```

done

(* Or one can use these somewhat inefficient functions to simulate the
Perl examples *)

let popleft l =
  match l with
  | [] -> raise Not_found
  | h::t -> h,(t @ [h])

let popright l =
  match List.rev l with
  | [] -> raise Not_found
  | h::t -> h,(h::(List.rev t))

let processes = ref [1;2;3;4;5]
let () =
  while true do
    let process,np = popleft !processes in
    processes := np;
    printf "Handling process %d\n" process;
    flush_all ();
    Unix.sleep 1;
  done

```

#### 4.17 Randomizing an Array

```

let fisher_yates_shuffle a =
  for i = Array.length a - 1 downto 1 do
    let x = a.(i)
    and r = Random.int (i+1) in
    a.(i) <- a.(r);
    a.(r) <- x;
  done

```

#### 4.18 Program: words

```

(* Assuming we start with a list of all the data called data, and
assuming we already have the current number of screen columns in
a variable cols *)

let words data cols =
  let strippedData =
    Array.of_list
      (List.map (Str.replace_first (Str.regexp "[ \\t\\n]+$") "") data) in
  let maxlen =
    (Array.fold_left
      (fun m s -> max m (String.length s)) 0 strippedData) + 1 in
  let cols = if cols < maxlen then 1 else cols / maxlen in
  let rows = ((Array.length strippedData - 1) + cols)/cols in
  let bufs = Array.init rows (fun x -> Buffer.create (cols * maxlen)) in
  for i = 0 to Array.length strippedData - 1 do
    let dst = String.make maxlen ' '

```

```

    and src = strippedData.(i) in
    String.blit src 0 dst 0 (String.length src);
    Buffer.add_string bufs.(i mod rows) dst
done;
Array.iter (fun x -> print_endline (Buffer.contents x)) bufs;;

```

#### 4.19 Program: permute

(\* Note: This routine uses the splice routine written in section 4.9 \*)

```

let tsc_permute arr =
  if Array.length arr > 0 then print_endline "Perms:";
  let rec permute arr perms =
    match Array.length arr with
    | 0 -> Array.iter (printf "%s ") perms; print_newline ();
    | _ ->
      for i = 0 to Array.length arr - 1 do
        let v,ni = splice arr i ~length:1 in
        permute ni (Array.append v perms);
      done in
  permute arr [[]]

```

(\* Note: This example is going to permute the words of a given string - also, I don't feel like bringing in the BigInt module, so we will trim any array longer than 12 elements down to 12 before permuting \*)

```

let fact = Array.append [|Some 1|] (Array.make 11 None)
let rec factorial n =
  match fact.(n) with
  | Some f -> f
  | None -> let f = n*(factorial (n-1)) in fact.(n) <- Some f; f

```

```

let n2pat n len =
  let rec nh n i pat =
    if i > len+1 then pat
    else nh (n/i) (i+1) ((n mod i)::pat) in
  nh n 1 []

```

```

let pat2perm pat =
  let rec ph source pat perm =
    match pat with
    | [] -> perm
    | h::t ->
      let v,s = splice source h ~length:1 in
      ph s t (v.(0)::perm) in
  Array.of_list (ph (Array.init (List.length pat) (fun i -> i)) pat [])

```

```

let n2perm n len =
  pat2perm (n2pat n len)

```

```

let mjd_permute s =
  let arr =
    let arr = Array.of_list (Str.split (Str.regexp "[ \\t]+") s) in

```

```
try
  Array.sub arr 0 12
  with Invalid_argument _ -> arr in
let len = Array.length arr - 1 in
for i = 0 to factorial (len+1) do
  let perm = Array.map (fun i -> arr.(i)) (n2perm i len) in
  Array.iter (printf "%s ") perm; print_newline ();
done
```

## 5 Hashes

```
(* Build an hash table element by element *)
let age = Hashtbl.create 3 ;; (* 3 is the supposed average size for the
                             hash table *)

Hashtbl.replace age "Nat" 24 ;
Hashtbl.replace age "Jules" 25 ;
Hashtbl.replace age "Josh" 17 ;;

(*-----*)

let assoc_list2hashtbl assoc_list =
  let h = Hashtbl.create 0 in
  List.iter (fun (k,v) -> Hashtbl.replace h k v) assoc_list ;
  h

let food_color = assoc_list2hashtbl
  [ "Apple", "red" ;
    "Banana", "yellow" ;
    "Lemon", "yellow" ;
    "Carrot", "orange" ;
  ] ;;
```

### 5.1 Adding an Element to a Hash

```
(*-----*)
Hashtbl.replace tbl key value ;;
(*-----*)
(* food_color defined per the introduction *)
Hashtbl.replace food_color "Raspberry" "pink" ;;

let hashtbl_keys h = Hashtbl.fold (fun key _ l -> key :: l) h []
let hashtbl_values h = Hashtbl.fold (fun _ value l -> value :: l) h []
let hashtbl2assoc_list h = Hashtbl.fold (fun key value l -
> (key, value) :: l) h []
;;
print_string "Known_foods:\n" ;
Hashtbl.iter (fun food _ -> print_endline food) food_color ;
print_string "Known_foods:\n" ;
List.iter print_endline (hashtbl_keys food_color) ;;
(*
> Known_foods:
> Banana
> Raspberry
> Apple
> Carrot
> Lemon
*)
(*-----*)
```

## 5.2 Testing for the Presence of a Key in a Hash

```
(*-----*)
(* does %HASH have a value for $KEY ? *)
if (Hashtbl.mem hash key) then
  (* it exists *)
else
  (* id doesn't exists *)
  ;;
(*-----*)
(* food_color defined per the introduction *)
List.iter (fun name ->
  let kind = if Hashtbl.mem food_color name then "food" else "drink" in
  printf "%s is a %s.\n" name kind
) ["Banana"; "Martini"] ;;
(*
> Banana is a food.
> Martini is a drink.
*)
(*-----*)
(* there's no such thing called "undef", "nil" or "null" in Caml
   if you really want such a value, use type "option" as shown below *)
let age = assoc_list2hashtbl
  [ "Toddler", 3 ; "Unborn", 0 ] ;;
(*> val age : (string, int) Hashtbl.t = <abstr> *)

List.iter (fun thing ->
  printf "%s: %s\n" thing
    (try match Hashtbl.find age thing with
     | 0 -> "Exists"
     | _ -> "Exists NonNull"
     with Not_found -> "")
) ["Toddler" ; "Unborn" ; "Phantasm" ; "Relic" ]

let age = assoc_list2hashtbl
  [ "Toddler", Some 3 ; "Unborn", Some 0 ; "Phantasm", None ] ;;
(*> val age : (string, int option) Hashtbl.t = <abstr> *)

List.iter (fun thing ->
  printf "%s: %s\n" thing
    (try match Hashtbl.find age thing with
     | None -> "Exists"
     | Some 0 -> "Exists Defined"
     | Some _ -> "Exists Defined NonNull"
     with Not_found -> "")
) ["Toddler" ; "Unborn" ; "Phantasm" ; "Relic" ]
(*
> Toddler: Exists Defined NonNull
> Unborn: Exists Defined
> Phantasm: Exists
> Relic:
*)
(*-----*)
```

```

let size = Hashtbl.create 20 in
List.iter (fun f ->
  if not (Hashtbl.mem size f) then
    Hashtbl.replace size f (Unix.stat f).Unix.st_size;
) (readlines stdin);
(*-----*)
(* here is a more complete solu-
tion which does stat 2 times the same file (to
be mimic perl's version) *)
let size = Hashtbl.create 20 in
List.iter (fun f ->
  if not (Hashtbl.mem size f) then
    Hashtbl.replace size f (try Some (Unix.stat f).Unix.st_size with _ -
> None)
) (readlines stdin);

```

### 5.3 Deleting from a Hash

```

(*-----*)
(* remove $KEY and its value from %HASH *)
Hashtbl.remove hash key ;
(*-----*)
(* food_color as per Introduction *)
open Printf

let print_foods () =
  printf "Keys: %s\n" (String.concat " " (hashtbl_keys food_color)) ;
  printf "Values: %s\n" (String.concat " " (hashtbl_values food_color))
;;
print_string "Initially:\n";
print_foods ();

print_string "\nWith Banana deleted\n";
Hashtbl.remove food_color "Banana";
print_foods ()
;;
(*-----*)
Hashtbl.clear food_color ;;
(*-----*)

```

### 5.4 Traversing a Hash

```

(*-----*)
(* in this section consider opened the Printf module using: *)
open Printf;;

Hashtbl.iter
  (fun key value ->
    (*
      do something with key and value
    *)
  )
  hash

```

```

;;
(*-----*)
List.iter (fun key ->
  let value = Hashtbl.find hash key in
    (*
      do something with key and value
    *)
  ) (hashtbl_keys hash)
;;
(*-----*)
(* food_color as defined in the introduction *)
Hashtbl.iter (printf "%s is %s.\n") food_color;
(*
> Lemon is yellow.
> Apple is red.
> Carrot is orange.
> Banana is yellow.
*)
(* but beware of: *)
Hashtbl.iter (printf "food_color: %s is %s.\n") food_color;
(*
> food_color: Lemon is yellow.
> Apple is red.
> Carrot is orange.
> Banana is yellow.
*)
(* write this instead:
   (more on it at http://caml.inria.fr/ocaml/htmlman/manual055.html) *)
Hashtbl.iter (fun k v -> printf "food_color: %s is %s.\n" k v) food_color;
(*
> food_color: Lemon is yellow.
> food_color: Apple is red.
> food_color: Carrot is orange.
> food_color: Banana is yellow.
*)

List.iter (fun key ->
  let value = Hashtbl.find food_color key in
    printf "%s is %s.\n" key value
  ) (hashtbl_keys food_color) ;
(*
> Lemon is yellow.
> Apple is red.
> Carrot is orange.
> Banana is yellow.
*)

(*-----*)
List.iter
  (fun key ->
    printf "%s is %s.\n" key (Hashtbl.find food_color key)
  )
  (sort_ (hashtbl_keys food_color))

```

```

;;

(*
> Apple is red.
> Banana is yellow.
> Carrot is orange.
> Lemon is yellow.
*)

(*-----*)
(* Ocaml is safe in loop, so you can't reset the hash iterator as in
Perl and you don't risk infinite loops using, say, List.iter or
Hashtbl.iter, but if you really want to infinite loop on the first key
you get ... *)
List.iter
  (fun key ->
    while true do
      printf "Processing %s\n" key
    done
  )
  (hashtbl_keys food_color)
;;
(*-----*)
(* countfrom - count number of messages from each sender *)
let main () =
  let file =
    let files = ref [] in
    Arg.parse [] (fun file -> files := !files @ [file]) "";
  try
    open_in (List.hd !files)
  with Failure "hd" -> stdin
  in
  let from = Hashtbl.create 50 in
  let add_from address =
    let old_count =
      try Hashtbl.find from address
      with Not_found -> 0
    in
    let new_count = old_count + 1 in
    Hashtbl.replace from address new_count;
  in
  let extractfrom = Str.regexp "^From: \\.*\\" in

  iter_lines (fun line ->
    if (Str.string_match extractfrom line 0) then
      add_from (Str.matched_group 1 line)
    else ()
  ) file;
  Hashtbl.iter (printf "%s: %d\n") from
;;
main() ;

```

## 5.5 Printing a Hash

```
(*-----*)
(* note that OCaml does not have a native polymorphic print function, so
examples in this section work for hashes that map string keys to string
values *)
Hashtbl.iter (printf "%s => %s\n") hash ;
(*-----*)

(* map in ocaml maps a function on a list, rather that evaluate an
expression in turn on a list as Perl does *)
List.iter
  (fun key ->
    printf "%s => %s\n" key (Hashtbl.find hash key)
  )
  (hashtbl_keys hash) ;
(*-----*)

(* build a list from an hash table, note that this is possible only if
the type of key and value are the same *)
let hashtbl2list hash =
  Hashtbl.fold
    (fun key value init -> key :: value :: init)
    hash
    []
;;
List.iter (printf "%s ") (hashtbl2list hash) ;
(* or *)
print_endline (String.concat " " (hashtbl2list hash)) ;
```

## 5.6 Retrieving from a Hash in Insertion Order

```
(*-----*)
(* In OCaml one usually use association lists which really is a list of
(key,value). Note that insertion and lookup is O(n) (!!!) *)

(* initialization *)
let empty_food_color = []
let food_color =
  [ "Banana", "Yellow" ;
    "Apple", "Green" ;
    "Lemon", "Yellow" ;
  ]
(* adding *)
let food_color' = food_color @ [ "Carrot", "orange" ]
;;
(* output entries in insertion order *)
print_endline "In insertion order, the foods are:";
List.iter (printf "%s is colored %s.\n") food_color;
(*
> Banana is colored Yellow.
> Apple is colored Green.
> Lemon is colored Yellow.
```

```

*)
(* is it a key? *)
let has_food food = mem_assoc food food_color
(* remove a key *)
let remove_food food = remove_assoc food food_color
(* searching *)
let what_color food =
  try
    let color = assoc food food_color in
    printf "%s is colored %s.\n" food color
  with Not_found -> printf "i don't know the color of %s\n" food
;;

```

## 5.7 Hashes with Multiple Values Per Key

```

(*-----*)
let re = Str.regexp "^\[^\ ]*\)\ *\[^\ ]*\)" in
let lines = readlines (Unix.open_process_in "who") in
let ttys = filter_some (List.map (fun line ->
  if (Str.string_match re line 0) then
    Some(Str.matched_group 1 line, Str.matched_group 2 line)
  else None) lines) in
List.iter
  (fun user ->
    printf "%s: %s\n" user (String.concat " " (all_assoc user ttys))
  ) (sort_ (uniq (List.map fst ttys)))
;
(*-----*)
List.iter
  (fun user ->
    let ttylist = all_assoc user ttys in
    printf "%s: %d ttys.\n" user (List.length ttylist);
    List.iter
      (fun tty ->
        let uname =
          try
            let uid = (Unix.stat ("/dev/" ^ tty)).Unix.st_uid in
            (Unix.getpwuid uid).Unix.pw_name
          with Unix.Unix_error _ -> "(not available)"
        in
        printf "%s (owned by %s)\n" tty uname
      ) ttylist
  ) (sort_ (uniq (List.map fst ttys)))
(*-----*)

```

## 5.8 Inverting a Hash

```

(*-----*)

open Hashtbl

(* size of an hash, i.e. number of bindings *)
let hashtbl_size h = List.length (hashtbl_keys h);;

```

```

(* in OCaml does not exists a builtin function like "reverse", here is
an equivalent one: *)
let hashtbl_reverse h =
  assoc_list2hashtbl (List.combine (hashtbl_values h) (hashtbl_keys h))
(* or *)
let hashtbl_reverse h =
  assoc_list2hashtbl (List.map (fun (a,b) -> (b,a)) (hashtbl2assoc_list h))
;;
(* or *)
let hashtbl_reverse_multi h =
  let newhash = Hashtbl.create (hashtbl_size h) in
  List.iter
    (fun v -> add newhash (find h v) v)
    (hashtbl_keys h);
  newhash
(* note that the last implementation maintain also multiple binding
for the
same key, see Hashtbl.add in the standard OCaml library for more info *)

(*-----*)
(* example of hashtbl_reverse *)

let reverse = hashtbl_reverse lookup;;
(*-----*)
let surname = assoc_list2hashtbl ["Mickey", "Mantle"; "Babe", "Ruth"] in
let firstname = hashtbl_reverse surname in
print_endline (Hashtbl.find firstname "Mantle");;
(*
> Mickey
*)

(*-----*)
(* foodfind - find match for food or color *)

let given = Sys.argv.(1) in
let color = assoc_list2hashtbl
  ["Apple", "red";
   "Banana", "yellow";
   "Lemon", "yellow";
   "Carrot", "orange"] in
let food = hashtbl_reverse color in
(try
  printf "%s is a food with color %s.\n" given (Hashtbl.find color given);
with Not_found -> ());
(try
  printf "%s is a food with color %s.\n" (Hashtbl.find food given) given
with Not_found -> ());
;;
(*-----*)
(* food_color defined as previous *)

let foods_with_color = hashtbl_reverse food_color in

```

```
List.iter (printf "%s ") (Hashtbl.find_all foods_with_color "yellow");
print_endline "were yellow foods."
;;
(*-----*)
```

## 5.9 Sorting a Hash

```
(*-----*)

(* you may define your own compare function to be used in sorting *)
let keys = List.sort compare_function (hashtbl_keys hash) in
List.iter
  (fun key ->
    let value = Hashtbl.find hash key in
    (* do something with key and value *)
    ()
  )
  keys ;
(* or use this one if you want to compare not only on keys *)
Hashtbl.iter
  (fun (key, value) ->
    (* do something with key and value *)
    ()
  ) (List.sort compare_function (hashtbl2assoc_list hash)) ;
(*-----*)
List.iter
  (fun food ->
    printf "%s is %s.\n" food (Hashtbl.find food_color food)
  )
  (List.sort (hashtbl_keys food_color))
;;
(*-----*)
(* examples of "compare_function": *)

(* alphabetical sort on the hash value *)
let compare_function (_,color1) (_,color2) = compare color1 color2

(* length sort on the hash value *)
let compare_function (_,color1) (_,color2) = com-
pare (String.length color1) (String.length color2)

(*-----*)
```

## 5.10 Merging Hashes

```
(*-----*)
(* definition of merge function on hashes: *)
let hashtbl_merge h1 h2 = as-
soc_list2hashtbl (hashtbl2assoc_list h1 @ hashtbl2assoc_list h2)

(* usage: *)
let merged = hashtbl_merge a b;;
(*-----*)
```

```

let merged = Hashtbl.create 0 in
List.iter
  (Hashtbl.iter (fun k v -> Hashtbl.add merged k v))
  [a;b]
;;
(*-----*)
let drink_color = assoc_list2hashtbl
  ["Galliano", "yellow";
   "Mai Tai", "blue"]
;;

let ingested_color = hashtbl_merge drink_color food_color;;
(*-----*)
let substance_color = Hashtbl.create 0 in
List.iter
  (Hashtbl.iter (fun k v -> Hashtbl.add merged k v))
  [food_color; drink_color]
;;

```

### 5.11 Finding Common or Different Keys in Two Hashes

```

(*-----*)
let common =
  List.filter
    (fun key -> Hashtbl.mem hash2 key)
    (hashtbl_keys hash1)
;;
(* common now contains commne keys, note that a key may appear multiple
times in this list due tu multiple bindings allowed in Hashtbl
implementation *)

let this_not_that =
  List.filter
    (fun key -> not (Hashtbl.mem hash2 key))
    (hashtbl_keys hash1)
;;
(*-----*)
let citrus_color = assoc_list2hashtbl
  ["Lemon", "yellow";
   "Orange", "orange";
   "Lime", "green"]

in
let non_citrus = Hashtbl.create 3 in
List.filter
  (fun key -> not (Hashtbl.mem citrus_color key))
  (hashtbl_keys food_color)
;;
(*-----*)

```

### 5.12 Hashing References

```

(*-----*)

```

```

open Unix;;
open Printf;;

let filenames = ["/etc/printcap"; "/vmlinuz"; "/bin/cat"] in
let openfiles = Hashtbl.create 3 in
print_newline();
List.iter
  (fun fname ->
    printf "%s is %d bytes long.\n"
      fname
      (stat fname).st_size
  )
  filenames
;;

(*-----*)

```

### 5.13 Presizing a Hash

```

(*-----*)

(* presize hash to num elements *)
let hash = Hashtbl.create num;;
(* other examples of initial size on hashes *)
let hash = Hashtbl.create 512;;
let hash = Hashtbl.create 1000;;

(*-----*)

```

### 5.14 Finding the Most Common Anything

```

(*-----*)

(* size of an array named "a" *)
let count = Array.length a;;

(* size of a list named "l" *)
let count = List.length l;;

(*-----*)

```

### 5.15 Representing Relationships Between Data

```

(*-----*)

open Printf;;
open Hashtbl;;

let father = assoc_list2hashtbl
  [ "Cain", "Adam";
    "Abel", "Adam";
    "Seth", "Adam";
    "Enoch", "Cain";

```

```

    "Irad", "Enoch";
    "Mehujael", "Irad";
    "Methusael", "Mehujael";
    "Lamech", "Methusael";
    "Jabal", "Lamech";
    "Jubal", "Lamech";
    "Tubalcain", "Lamech";
    "Enos", "Seth"] ;;
(*-----*)
(* recursively print all parents of a given name *)
let rec parents s =
  printf "%s " s;
  if mem father s then
    parents (find father s)
  else
    printf "\n"
in
  iter_lines parents stdin
;;
(*-----*)
let children = hashtbl_reverse_multi father in
iter_lines
  (fun line ->
    List.iter (printf "%s ") (find_all children line);
    print_newline()
  )
  stdin;
;;
(*-----*)
(* build an hash that map filename to list of included file *)
open Hashtbl;;
open Str;;

let includes = create (List.length files);;
let includeRE = regexp "^#include <\\([a-zA-Z0-9.]+\\)>";;
let isincludeline l = string_match includeRE l 0;;
let getincludes fname =
  let includelines =
    List.filter isincludeline (readlines (open_in fname))
  in
  List.map (replace_first includeRE "\\1") includelines
;;
List.iter (fun fname -> add includes fname (getincludes fname)) files;;

(*-----*)
(* build a list of files that does not include system headers *)
let hasnoinclude fname = (find includes fname = []) in
List.filter hasnoinclude (uniq (hashtbl_keys includes));;

(*-----*)

```

## 5.16 Program: dutree

```
(*-----*)

#!/usr/bin/ocaml
(* dutree - print sorted indented rendition of du output *)
#load "str.cma";;
#load "unix.cma";;

let dirsize = Hashtbl.create 0
let kids = Hashtbl.create 0

(* run du, read in input, save sizes and kids *)
(* return last directory (file?) read *)
let input () =
  let last_name = ref "" in
  let last_push = ref None in
  let argv = "du" :: List.tl (Array.to_list Sys.argv) in
  let ch = Unix.open_process_in (String.concat " " argv) in
  begin
    try
      while true do
        let line = input_line ch in
        match Str.bounded_split (Str.regexp "[ \\t]+") line 2 with
        | [size; name] ->
          let size = int_of_string size in
          Hashtbl.replace dirsize name size;
          let parent =
            Str.replace_first (Str.regexp "[^/]+$") "" name in
          last_name := name;
          last_push :=
            Some (parent,
                  try Some (Hashtbl.find kids parent)
                    with Not_found -> None);
          Hashtbl.replace kids parent
            (name ::
             (try Hashtbl.find kids parent
              with Not_found -> []))
        | _ -> failwith line
      done
    with End_of_file ->
      ignore (Unix.close_process_in ch)
  end;
begin
  match !last_push with
  | None -> ()
  | Some (parent, None) ->
    Hashtbl.remove kids parent
  | Some (parent, Some previous) ->
    Hashtbl.replace kids parent previous
end;
!last_name
```

```

(* figure out how much is taken up in each directory *)
(* that isn't stored in subdirectories.  add a new *)
(* fake kid called "." containing that much. *)
let rec getdots root =
  let size = Hashtbl.find dirsize root in
  let cursize = ref size in
  if Hashtbl.mem kids root
  then
    begin
      List.iter
        (fun kid ->
          cursize := !cursize - Hashtbl.find dirsize kid;
          getdots kid)
          (Hashtbl.find kids root)
    end;
  if size <> !cursize
  then
    begin
      let dot = root ^ "/" in
      Hashtbl.replace dirsize dot !cursize;
      Hashtbl.replace kids root
        (dot ::
          (try Hashtbl.find kids root
            with Not_found -> []))
    end
  end

(* recursively output everything, *)
(* passing padding and number width in as well *)
(* on recursive calls *)
let rec output ?(prefix="") ?(width=0) root =
  let path = Str.replace_first (Str.regexp ".*/") "" root in
  let size = Hashtbl.find dirsize root in
  let line = Printf.sprintf "%*d %s" width size path in
  Printf.printf "%s%s\n" prefix line;
  let prefix =
    Str.global_replace (Str.regexp "[^|]") " "
    (Str.replace_first (Str.regexp "[0-9]" "") "| "
      (prefix ^ line)) in
  if Hashtbl.mem kids root
  then
    begin
      let kids = Hashtbl.find kids root in
      let kids =
        List.rev_map
          (fun kid -> (Hashtbl.find dirsize kid, kid)) kids in
      let kids = List.sort compare kids in
      let kids = List.rev_map (fun (_, kid) -> kid) kids in
      let width =
        String.length
          (string_of_int (Hashtbl.find dirsize (List.hd kids))) in
      List.iter (output ~prefix ~width) kids
    end
  end
end

```

```
let () =  
  let topdir = input () in  
  getdots topdir;  
  output topdir
```

## 6 Pattern Matching

```
(* We will use the Str library distributed with OCaml for regular expressions.
 * There are two ways to use the str library, building a top or passing it to ocaml.
 * Under Unix, you can create a new toplevel which has the Str module:
 *   $ ocamlmktop -o strttop str.cma
 *   $ ./strttop
 * Now you don't need to prefix the contents of the str module with Str.
 * The alternative is to pass str.cma as a parameter:
 *   $ ocaml str.cma
 * Now you may refer to the contents of the str module by using Str.
 * Under Windows, if you are using ocamlwin.exe you can simply load Str:
 *   # load "str.cma";;
 *)
(* Str.search_forward returns an int or throws an exception if the pattern isn't found.
 * In Perl, the =~ operator returns null. Since these two values have different
 * types in OCaml, we cannot copy this behaviour directly.
 * Instead, we return an impossible index, -1 using try ... with.
 * Another method would be to define an =~ operator and use that directly:
# let ( =~ ) s re = Str.string_match (Str.regexp re) s 0;;
val ( =~ ) : string -> string -> bool = <fun>
# "abc" =~ "a";;
- : bool = true
# "bc" =~ "a";;
- : bool = false
 * Don't underestimate the power of this. Many of the following examples could be
 * simplified by defining infix operators.
 *)
try Str.search_forward (Str.regexp pattern) string 0;
with Not_found -> -1;;

try Str.replace_first (Str.regexp pattern) replacement string;
with Not_found -> "";;
(*-----*)
try (Str.search_forward (Str.regexp "sheep") meadow 0) > -1;
with Not_found -> false;; (* true if meadow contains "sheep" *)

try not ((Str.search_forward (Str.regexp "sheep") meadow 0) > -1);
with Not_found -> true;; (* true if meadow doesn't contain "sheep" *)

let meadow =
  try Str.replace_first (Str.regexp "old") "new" meadow;
  with Not_found -> meadow;; (* Replace "old" with "new" in meadow *)
(*-----*)
try
  let temp = Str.search_forward (Str.regexp "\\bovines?\\b") meadow 0 in
  print_string "Here be sheep!";
with Not_found -> ();;
```

```

(*-----*)
let string = "good food" in
  try
    Str.replace_first (Str.regexp "o*") "e" string;
  with Not_found -> string;;
(*-----*)
(* There is no way to take command line parameters to ocaml that I know of.
 * You would first have to compile your OCaml program using ocamlc.
 *)
(*-----*)
let rec match_num s start=
  if String.length s > 0 then
    try
      let temp = Str.search_forward (Str.regexp "[0123456789]+") s start in
        print_string (String.concat "" ("Found number " :: Str.matched_string s :: ["\n"]));
        match_num s (temp + 1);
      with Not_found -> ();
    else
      ();;
(*-----*)
let rec match_group s start numbers=
  if String.length s > 0 then
    try
      let temp = (Str.search_forward (Str.regexp "[0123456789]+") s start) in
        let numbers = Str.matched_string s :: numbers in
          match_group s (temp + 1) numbers;
        with Not_found -> numbers;
    else
      numbers;;
(*-----*)
let (+=) s re =
  let result = ref [] in
  let offset = ref 0 in
  while ((String.length s) > !offset) do
    try
      offset := 1 + (Str.search_forward (Str.regexp re) s !offset);
      result := !result @ [Str.matched_string s] @ [];
    with Not_found -> ignore (offset := String.length s)
  done;
  result;;

let (=-) s re =
  let result = ref [] in
  let offset = ref 0 in
  while ((String.length s) > !offset) do
    try
      ignore (Str.search_forward (Str.regexp re) s !offset);
      offset := Str.match_end ();
      result := !result @ [Str.matched_string s] @ [];
    with Not_found -> ignore (offset := String.length s)
  done;
  result;;

```

```

let digits = "123456789";;
let yeslap = digits += "[1234567890][1234567890][1234567890]";;
let nonlap = digits -= "[1234567890][1234567890][1234567890]";;

print_string "Non-overlapping: ";
List.iter (fun v -> print_string (v ^ " ")) !nonlap;
print_string "\n";;
(* Non-overlapping: 123 456 789 *)

print_string "Overlapping: ";
List.iter (fun v -> print_string (v ^ " ")) !yeslap;
print_string "\n";;
(* Overlapping: 123 234 345 456 567 678 789 *)
(*-----*)
let index = ref 0;;
let string = "And little lambs eat ivy";;
try
  index := Str.search_forward (Str.regexp "l[^s]*s") string 0;
with Not_found -> ();;

print_string ("(" ^ (String.sub string 0 !index) ^ ")");
print_string ("(" ^ (Str.matched_string string) ^ ")");
print_string ("(" ^ (Str.string_after string 16) ^ ")\n");;
(* (And ) (little lambs) ( eat ivy) *)

```

## 6.1 Copying and Substituting Simultaneously

```

#load "str.cma";;

(* The Str module doesn't modify strings in place; you always get
   a copy when you perform a substitution. *)
let dst = Str.global_replace (Str.regexp "this") "that" src

(* Strip to basename. *)
let progname = Str.replace_first (Str.regexp "^.*/") "" Sys.argv.(0)

(* Make All Words Title-Cased. *)
let capword =
  Str.global_substitute
    (Str.regexp "\\b.")
    (fun s -> String.uppercase (Str.matched_string s))
  words

(* /usr/man/man3/foo.1 changes to /usr/man/cat3/foo.1 *)
let catpage =
  Str.replace_first (Str.regexp "man\\([0-9]\\)") "cat\\1" manpage

(* Copy and substitute on all strings in a list. *)
let bindirs = ["/usr/bin"; "/bin"; "/usr/local/bin"]
let libdirs =
  List.map (fun s -> Str.replace_first (Str.regexp "bin") "lib" s)
  bindirs

```

```
(* ["/usr/lib"; "/lib"; "/usr/local/lib"] *)
```

## 6.2 Matching Letters

```
(* Str can do a simple character range match, but it isn't very  
practical for matching alphabetic characters in general. *)
```

```
#load "str.cma";;  
let () =  
  if Str.string_match (Str.regexp "[A-Za-z]+$") var 0  
  then print_endline "var is purely alphabetic"
```

```
(* With Pcre, you can use UTF8 support and match characters with  
the letter property. *)
```

```
#directory "+pcre";;  
#load "pcre.cma";;  
let () =  
  if Pcre.pmatch ~rex:(Pcre.regexp ~flags:['UTF8] "\\pL+$") var  
  then print_endline "var is purely alphabetic"
```

## 6.3 Matching Words

```
#load "str.cma";;
```

```
(* Str's regexps lack a whitespace-matching pattern.  
Here is a substitute. *)
```

```
let whitespace_chars =  
  String.concat ""  
    (List.map (String.make 1)  
      [  
        Char.chr 9; (* HT *)  
        Char.chr 10; (* LF *)  
        Char.chr 11; (* VT *)  
        Char.chr 12; (* FF *)  
        Char.chr 13; (* CR *)  
        Char.chr 32; (* space *)  
      ])  
  )
```

```
let space = "[" ^ whitespace_chars ^ "]"  
let non_space = "[^" ^ whitespace_chars ^ "]"
```

```
(* as many non-whitespace characters as possible *)  
let regexp = Str.regexp (non_space ^ "+")
```

```
(* as many letters, apostrophes, and hyphens *)  
let regexp = Str.regexp "[A-Za-z'-]+"
```

```
(* usually best *)  
let regexp = Str.regexp "\\b\\([A-Za-z]+\\)\\b"
```

```
(* fails at ends or w/ punctuation *)  
let regexp = Str.regexp (space ^ "\\([A-Za-z]+\\)" ^ space)
```

## 6.4 Commenting Regular Expressions

```
#!/usr/bin/ocaml
(* rename - change all "foo.bar.com" style names in the input stream
   into "foo.bar.com [204.148.40.9]" (or whatever) instead *)

#directory "+pcre";;
#load "pcre.cma";;
#load "unix.cma";;

let regexp =
  Pcre.regexp ~flags:['EXTENDED] "
    (
      # capture the hostname in substring 1
      (?:
        # these parens for grouping only
        (?! [-_] )
        # lookahead for neither underscore nor dash
        [\\w-] +
        # hostname component
        \\.
        # and the domain dot
      ) +
      # now repeat that whole thing a bunch of times
      [A-Za-z]
      # next must be a letter
      [\\w-] +
      # now trailing domain part
    )
    # end of substring 1 capture
  "

let process line =
  print_endline
  (Pcre.substitute_substrings
   ~rex:regexp
   ~subst:(fun subs ->
     let name = Pcre.get_substring subs 1 in
     let addr =
       try
         Unix.string_of_inet_addr
         (Unix.gethostbyname name).Unix.h_addr_list.(0)
       with Not_found -> "???" in
     name ^ " [" ^ addr ^ "]"
   )
  line)

let () =
  try
    while true do
      let line = read_line () in
      process line
    done
  with End_of_file -> ()

(*-----*)

let vars = Hashtbl.create 0
let () =
  Hashtbl.replace vars "name" "Bob";
  Hashtbl.replace vars "flavor" "rhubarb"

let () =
```

```

print_endline
  (Pcre.substitute_substrings
   ~rex:(Pcre.regexp ~flags:['EXTENDED] "
         \\#           # a pound sign
         (\\w+)       # the variable name
         \\#           # another pound sign
   ")
   ~subst:(fun subs -> Hashtbl.find vars (Pcre.get_substring subs 1))
   "Hello, #name#, would you like some #flavor# pie?")

```

## 6.5 Finding the Nth Occurrence of a Match

```

#load "str.cma";;

let want = 3
let count = ref 0
let pond = "One fish two fish red fish blue fish"
let regexp = Str.regexp_case_fold "\\([a-z]+\\)[ ]+fish\\b"

exception Found of string
let () =
  let start = ref 0 in
  try
    while true do
      ignore (Str.search_forward regexp pond !start);
      start := !start + String.length (Str.matched_string pond);
      incr count;
      if !count = want then raise (Found (Str.matched_group 1 pond))
    done
  with
  | Found color ->
    Printf.printf "The third fish is a %s one.\n" color
  | Not_found ->
    Printf.printf "Only found %d fish!\n" !count

(* The third fish is a red one. *)

(*-----*)

let colors =
  let start = ref 0 in
  let fish = ref [] in
  begin
    try
      while true do
        ignore (Str.search_forward regexp pond !start);
        start := !start + (String.length (Str.matched_string pond));
        fish := (Str.matched_group 1 pond) :: !fish
      done;
      with Not_found -> ()
    end;
    Array.of_list (List.rev !fish)

```

```

let () =
  Printf.printf "The third fish in the pond is %s.\n" colors.(2)

(* The third fish in the pond is red. *)

(*-----*)

let evens =
  let colors' = ref [] in
  Array.iteri
    (fun i color -> if i mod 2 = 1 then colors' := color :: !colors')
    colors;
  List.rev !colors'
let () =
  Printf.printf "Even numbered fish are %s.\n" (String.concat " " evens)

(* Even numbered fish are two blue. *)

(*-----*)

let () =
  let count = ref 0 in
  print_endline
    (Str.global_substitute
     (Str.regexp_case_fold "\\b\\([a-z]+\\)\\([ ]+fish\\b\\)")
     (fun s ->
      incr count;
      if !count = 4
      then "sushi" ^ Str.matched_group 2 s
      else Str.matched_group 1 s ^ Str.matched_group 2 s)
     pond)

(* One fish two fish red fish sushi fish *)

(*-----*)

let pond = "One fish two fish red fish blue fish swim here."
let regexp = Str.regexp_case_fold "\\b\\([a-z]+\\)[ ]+fish\\b"
let colors =
  let rec loop start acc =
    try
      ignore (Str.search_forward regexp pond start);
      loop
        (start + String.length (Str.matched_string pond))
        (Str.matched_group 1 pond :: acc)
    with Not_found ->
      acc in
  loop 0 []
let color = List.hd colors
let () = Printf.printf "Last fish is %s.\n" color

(* Last fish is blue. *)

```

## 6.6 Matching Multiple Lines

```
#!/usr/bin/ocaml
(* killtags - very bad html tag killer *)
#load "str.cma";;
let regexp = Str.regexp "<[^>]*>"
let () =
  List.iter
    (fun filename ->
      let lines = ref [] in
      let in_channel = open_in filename in
      try
        begin
          try while true do lines := input_line in_channel :: !lines done
            with End_of_file -> ()
          end;
          let contents = String.concat "\n" (List.rev !lines) in
          print_endline
            (String.concat ""
              (List.map
                (function
                  | Str.Text s -> s
                  | _ -> ""))
                (Str.full_split regexp contents)));
          close_in in_channel
        with e ->
          close_in in_channel;
          raise e)
      (List.tl (Array.to_list Sys.argv))

  (*-----*)

#!/usr/bin/ocaml
(* headerfy - change certain chapter headers to html *)
#load "str.cma";;

let line_stream_of_channel channel =
  Stream.from
    (fun _ -> try Some (input_line channel) with End_of_file -> None)

let paragraph_stream_of_channel channel =
  let lines = line_stream_of_channel channel in
  let rec next para_lines i =
    match Stream.peek lines, para_lines with
    | None, [] -> None
    | Some "", [] -> Stream.junk lines; next para_lines i
    | Some "", _
    | None, _ -> Some (String.concat "\n" (List.rev para_lines))
    | Some line, _ -> Stream.junk lines; next (line :: para_lines) i in
  Stream.from (next [])

let regexp = Str.regexp "^Chapter[\\r\\n\\t ]+[0-9]+[\\r\\n\\t ]*:[^\\r\\n]*"
```

```

let headerfy chunk =
  String.concat ""
    (List.map
      (function
        | Str.Text s -> s
        | Str.Delim s -> "<H1>" ^ s ^ "</H1>")
      (Str.full_split regexp chunk))

let () =
  List.iter
    (fun filename ->
      let in_channel = open_in filename in
      try
        Stream.iter
          (fun para ->
            print_endline (headerfy para);
            print_newline ())
          (paragraph_stream_of_channel in_channel);
        close_in in_channel
      with e ->
        close_in in_channel;
        raise e)
    (List.tl (Array.to_list Sys.argv))

```

## 6.7 Reading Records with a Pattern Separator

```

#load "str.cma";;
let chunks =
  let lines = ref [] in
  begin
    try while true do lines := input_line stdin :: !lines done
      with End_of_file -> ()
    end;
  let contents = String.concat "\n" (List.rev !lines) in
  Str.full_split (Str.regexp "^\\.\\.\\.(Ch\\|Se\\|Ss\\)$") contents
let () =
  Printf.printf
    "I read %d chunks.\n"
    (List.length chunks)

```

## 6.8 Extracting a Range of Lines

```

#load "str.cma";;

(* Creates a stream that produces ranges of items from another stream.
   Production of items starts when when (start_test count item) returns
   true and stops when (finish_test count item) returns true. Multiple
   ranges will be produced if start_test returns true again. The count
   starts at 1. Ranges are inclusive; the item that causes finish_test
   to return true will be produced. *)
let stream_range start_test finish_test stream =
  let active = ref false in
  let count = ref 1 in

```

```

let rec next i =
  match Stream.peek stream with
  | None -> None
  | Some item ->
    if not !active then
      begin
        if start_test !count item
        then (active := true; next i)
        else (Stream.junk stream; incr count; next i)
      end
    else
      begin
        if finish_test !count item then active := false;
        Stream.junk stream;
        incr count;
        Some item
      end in
  Stream.from next

(* Creates a stream that produces items between a pair of indices.
   If start = 2 and finish = 4, items 2, 3, and 4 will be produced.
   The first item is number 1. *)
let stream_range_numbers start finish stream =
  stream_range
    (fun count _ -> count = start)
    (fun count _ -> count = finish)
  stream

(* Creates a stream that produces strings between a pair of regexps.
   The regexp will be tested using Str.string_match. *)
let stream_range_patterns start finish stream =
  stream_range
    (fun _ line -> Str.string_match start line 0)
    (fun _ line -> Str.string_match finish line 0)
  stream

(* Produce a stream of lines from an input channel. *)
let line_stream_of_channel channel =
  Stream.from
    (fun _ -> try Some (input_line channel) with End_of_file -> None)

(* Print lines 15 through 17 inclusive. *)
let () =
  Stream.iter
    print_endline
    (stream_range_numbers 15 17
     (line_stream_of_channel (open_in datafile)))

(* Print out all <XMP> .. </XMP> displays from HTML doc. *)
let () =
  Stream.iter
    print_endline
    (stream_range_patterns

```

```

(Str.regexp ".*<XMP>")
(Str.regexp ".*</XMP>")
(line_stream_of_channel stdin))

(*-----*)

let in_header = ref true
let in_body = ref false
let () =
  Stream.iter
    (fun line ->
      if !in_header && line = ""
      then (in_header := false; in_body := true)
      else
        begin
          (* do something with line *)
        end)
    (line_stream_of_channel stdin)

(*-----*)

module StringSet = Set.Make(String)
let seen = ref StringSet.empty
let email_regexp = Str.regexp "\\([^<>(),; \\t]+@[^<>(),; \\t]+\\)"
let () =
  Stream.iter
    (fun line ->
      List.iter
        (function
          | Str.Delim email ->
            if not (StringSet.mem email !seen)
            then
              begin
                seen := StringSet.add email !seen;
                print_endline email;
              end
          | _ -> ())
        (Str.full_split email_regexp line))
    (stream_range_patterns
      (Str.regexp "^From:[ \\t]")
      (Str.regexp "^$")
      (line_stream_of_channel stdin))

```

## 6.9 Matching Shell Globs as Regular Expressions

```

#load "str.cma";;

let regexp_string_of_glob s =
  let i, buffer = ref (-1), Buffer.create (String.length s + 8) in
  let read () =
    incr i;
    if !i < String.length s
    then Some s.[!i]

```

```

    else None in
let write = Buffer.add_string buffer in
let rec parse_glob () =
  match read () with
  | Some '*' -> write ".*"; parse_glob ()
  | Some '?' -> write "."; parse_glob ()
  | Some '[' -> parse_bracket ""
  | Some c -> write (Str.quote (String.make 1 c)); parse_glob ()
  | None -> ()
and parse_bracket text =
  match read () with
  | Some '!' when text = "" -> parse_bracket "^"
  | Some ']' -> write ("[" ^ text ^ "]"); parse_glob ()
  | Some c -> parse_bracket (text ^ (String.make 1 c))
  | None -> write (Str.quote ("[" ^ text)) in
write "^";
parse_glob ();
write "$";
Buffer.contents buffer

let regexp_of_glob s =
  Str.regexp (regexp_string_of_glob s)

let regexp_of_glob_case_fold s =
  Str.regexp_case_fold (regexp_string_of_glob s)

```

## 6.10 Speeding Up Interpolated Matches

```

#load "str.cma";;

let popstates = ["CO"; "ON"; "MI"; "WI"; "MN"]

(* Naive version: Compile a regexp each time it is needed. *)
let popgrep1 () =
  try
    begin
      while true do
        let line = input_line stdin in
        try
          List.iter
            (fun state ->
              if (Str.string_match
                  (Str.regexp (".*\\b" ^ (Str.quote state) ^ "\\b"))
                  line 0)
              then (print_endline line; raise Exit))
            popstates
        with Exit -> ()
      done
    end
  with End_of_file -> ()

(* First optimization: Compile the regexps in advance. *)
let popgrep2 () =

```

```

let popstate_regexps =
  List.map
    (fun state ->
      Str.regexp (".*\\b" ^ (Str.quote state) ^ "\\b"))
    popstates in
try
  begin
    while true do
      let line = input_line stdin in
      try
        List.iter
          (fun regexp ->
            if (Str.string_match regexp line 0)
            then (print_endline line; raise Exit))
          popstate_regexps
        with Exit -> ()
      done
    end
  with End_of_file -> ()

(* Second optimization: Build a single regexp for all states. *)
let popgrep3 () =
  let popstates_regexp =
    Str.regexp
      (".*\\b\\\\"
        ^ (String.concat "\\|" (List.map Str.quote popstates))
        ^ "\\|\\b") in
  try
    begin
      while true do
        let line = input_line stdin in
        if Str.string_match popstates_regexp line 0
        then print_endline line
      done
    end
  with End_of_file -> ()

(* Speed tests with a 15,000 line input file: *)
let () = popgrep1 ()      (* time: 13.670s *)
let () = popgrep2 ()      (* time: 0.264s *)
let () = popgrep3 ()      (* time: 0.123s *)

```

## 6.11 Testing for a Valid Pattern

```

#load "str.cma";;
let () =
  while true do
    print_string "Pattern? ";
    flush stdout;
    let pattern = input_line stdin in
    try ignore (Str.regexp pattern)
    with Failure message ->
      Printf.printf "INVALID PATTERN: %s\n" message

```

```

done

(*-----*)

let is_valid_pattern pattern =
  try ignore (Str.regexp pattern); true
  with Failure _ -> false

(*-----*)

#!/usr/bin/ocaml
(* paragrep - trivial paragraph grepper *)
#load "str.cma";;

let line_stream_of_channel channel =
  Stream.from
    (fun _ -> try Some (input_line channel) with End_of_file -> None)

let paragraph_stream_of_channel channel =
  let lines = line_stream_of_channel channel in
  let rec next para_lines i =
    match Stream.peek lines, para_lines with
    | None, [] -> None
    | Some "", [] -> Stream.junk lines; next para_lines i
    | Some "", _
    | None, _ -> Some (String.concat "\n" (List.rev para_lines))
    | Some line, _ -> Stream.junk lines; next (line :: para_lines) i in
  Stream.from (next [])

let paragrep pat files =
  let regexp =
    begin
      try Str.regexp pat
      with Failure msg ->
        Printf.eprintf "%s: Bad pattern %s: %s\n" Sys.argv.(0) pat msg;
        exit 1
    end in
  let count = ref 0 in
  List.iter
    (fun file ->
      let channel =
        if file = "-"
        then stdin
        else open_in file in
      try
        Stream.iter
          (fun para ->
            incr count;
            try
              ignore (Str.search_forward regexp para 0);
              Printf.printf "%s %d: %s\n\n" file !count para
            with Not_found -> ())
          (paragraph_stream_of_channel channel);
      end
    )
    files

```

```

        close_in channel
    with e ->
        close_in channel;
        raise e)
files

let () =
  match List.tl (Array.to_list Sys.argv) with
  | pat :: [] -> paragrep pat ["-"]
  | pat :: files -> paragrep pat files
  | [] -> Printf.eprintf "usage: %s pat [files]\n" Sys.argv.(0)

(*-----*)

let safe_pat = Str.quote pat

```

## 6.12 Honoring Locale Settings in Regular Expressions

(\* OCaml does not provide a way to change the locale, and PCRE does not appear to be sensitive to the default locale. Regardless, Str does not support locales, and PCRE only matches ASCII characters for `\w` and friends. This example instead demonstrates the use of PCRE's UTF-8 support to match words, and it does not use locales. \*)

```

#directory "+pcre";;
#load "pcre.cma";;

(* encoded as UTF-8 *)
let name = "andreas k\xc3\xb6nig"

(* the original regexp which is not Unicode-aware *)
let ascii_regexp = Pcre.regexp "\\b(\\w+)\\b"

(* a revised regexp which tests for Unicode letters and numbers *)
let utf8_regexp = Pcre.regexp ~flags:[`UTF8] "([\\pL\\pN]+)"

let () =
  List.iter
    (fun (enc, regexp) ->
      Printf.printf "%s names: %s\n" enc
        (String.concat " "
          (List.map
            String.capitalize
            (List.flatten
              (Array.to_list
                (Array.map
                  Array.to_list
                  (Pcre.extract_all
                    ~full_match:false
                    ~rex:regexp
                    name)))))))
      ["ASCII", ascii_regexp; "UTF-8", utf8_regexp]

```

```
(*
  ASCII names: Andreas K Nig
  UTF-8 names: Andreas König
*)
```

## 6.13 Approximate Matching

```
(* Calculates the Levenshtein, or edit distance, between two strings. *)
```

```
let levenshtein s t =
  let n = String.length s in
  let m = String.length t in
  match (m, n) with
  | (m, 0) -> m
  | (0, n) -> n
  | (m, n) ->
    let d = Array.init (m + 1) (fun x -> x) in
    let x = ref 0 in
    for i = 0 to n - 1 do
      let e = ref (i + 1) in
      for j = 0 to m - 1 do
        let cost = if s.[i] = t.[j] then 0 else 1 in
        x :=
          min
            (d.(j + 1) + 1)      (* insertion *)
            (min
              (!e + 1)          (* deletion *)
              (d.(j) + cost));  (* substitution *)
        d.(j) <- !e;
        e := !x
      done;
      d.(m) <- !x
    done;
    !x
```

```
(* Determines if two strings are an approximate match. *)
```

```
let amatch ?(percentage=20) s t =
  levenshtein s t * 100 / String.length s <= percentage
```

```
let () =
  let dict = open_in "/usr/dict/words" in
  try
    while true do
      let word = input_line dict in
      if amatch "ballast" word
      then print_endline word
    done
  with End_of_file -> close_in dict
```

```
(*
  ballast
  blast
*)
```

## 6.14 Matching from Where the Last Pattern Left Off

```
#directory "+pcre";;
#load "pcre.cma";;

let s = "12 345 hello 6 7world89 10"
let rex = Pcre.regexp "(\\d+)"

let () =
  let subs = ref (Pcre.exec ~rex s) in
  try
    while true do
      Printf.printf "Found %s\n" (Pcre.get_substring !subs 1);
      subs := Pcre.next_match ~rex !subs
    done
  with Not_found -> ()

(*-----*)

let () =
  let n = " 49 here" in
  let n = Pcre.replace ~pat:"\\G " ~templ:"0" n in
  print_endline n

(* 00049 here *)

(*-----*)

let s = "3,4,5,9,120"
let rex = Pcre.regexp "\\G,(?\\d+)"

let () =
  let subs = ref (Pcre.exec ~rex s) in
  try
    while true do
      Printf.printf "Found number %s\n" (Pcre.get_substring !subs 1);
      subs := Pcre.next_match ~rex !subs
    done
  with Not_found -> ()

(*-----*)

let s = "The year 1752 lost 10 days on the 3rd of September"

let rex = Pcre.regexp "(\\d+)"
let subs = ref (Pcre.exec ~rex s)

let () =
  try
    while true do
      Printf.printf "Found number %s\n" (Pcre.get_substring !subs 1);
      subs := Pcre.next_match ~rex !subs
    done
```

```

with Not_found -> ()

let () =
  let rex = Pcre.regexp "\\G(\\S+)" in
  subs := Pcre.next_match ~rex !subs;
  Printf.printf "Found %s after the last number.\n"
    (Pcre.get_substring !subs 1)

(*
  Found number 1752
  Found number 10
  Found number 3
  Found rd after the last number.
*)

(*-----*)

let () =
  match Pcre.get_substring_ofs !subs 1 with
  | (start, finish) ->
    Printf.printf
      "The position in 's' is %d..%d\n" start finish

(* The position in 's' is 35..37 *)

```

## 6.15 Greedy and Non-Greedy Matches

```

let s = "Even <TT>vi</TT> can edit <TT>troff</TT> effectively."

(* The Str library does not support non-greedy matches. In many cases,
   you can turn a non-greedy match into a greedy one, however: *)

#load "str.cma";;

let () = print_endline (Str.global_replace (Str.regexp "<.*>") "" s)
(* Even effectively. *)
let () = print_endline (Str.global_replace (Str.regexp "<[^>]*>") "" s)
(* Even vi can edit troff effectively. *)

(* If you need non-greedy matches, you'll want to use PCRE instead: *)

#directory "+pcre";;
#load "pcre.cma";;

let () = print_endline (Pcre.replace ~pat:"<.*?>" ~templ:"" s)
(* Even vi can edit troff effectively. *)

(* Non-greedy matches don't always work the way you expect: *)

let s = "<b><i>this</i> and <i>that</i> are impor-
tant</b> Oh, <b><i>me too!</i></b>"

let rex = Pcre.regexp "<b><i>(.*?)</i></b>"

```

```

let () = print_endline (Pcre.extract ~rex s).(1)
(* this<i> and <i>that</i> are important</b> Oh, <b><i>me too! *)

(* One solution is to use a non-grouping negative lookahead assertion: *)

let rex = Pcre.regexp "<b><i>((?:?!</b>|</i>).)*</i></b>"
let () = print_endline (Pcre.extract ~rex s).(1)
(* me too! *)

(* If performance is important, here is a faster technique: *)

let rex = Pcre.regexp ~flags:[`DOTALL; `EXTENDED] "
  <b><i>
  [^<]* # stuff not possibly bad, and not possibly the end.
  (?
  # at this point, we can have '<' if not part of something bad
  (?! </?[ib]> ) # what we can't have
  < # okay, so match the '<'
  [^<]* # and continue with more safe stuff
  ) *
  </i></b>
"

let () = print_endline (Pcre.extract ~rex s).(0)
(* <b><i>me too!</i></b> *)

```

## 6.16 Detecting Duplicate Words

```

#directory "+pcre";
#load "pcre.cma";

let line_stream_of_channel channel =
  Stream.from
    (fun _ -> try Some (input_line channel) with End_of_file -> None)

let paragraph_stream_of_channel channel =
  let lines = line_stream_of_channel channel in
  let rec next para_lines i =
    match Stream.peek lines, para_lines with
    | None, [] -> None
    | Some "", [] -> Stream.junk lines; next para_lines i
    | Some "", _
    | None, _ -> Some (String.concat "\n" (List.rev para_lines))
    | Some line, _ -> Stream.junk lines; next (line :: para_lines) i in
  Stream.from (next [])

let find_dup_words files =
  let rex = Pcre.regexp ~flags:[`CASELESS; `EXTENDED] "
    \\b # start at a word boundary (begin letters)
    (\\S+) # find chunk of non-whitespace
    \\b # until another word boundary (end letters)
    (
      \\s+ # separated by some whitespace

```

```

        \\1      # and that very same chunk again
        \\b      # until another word boundary
    ) +        # one or more sets of those
" in
let count = ref 0 in
List.iter
  (fun file ->
    let channel = if file = "-" then stdin else open_in file in
    try
      Stream.iter
        (fun para ->
          incr count;
          try
            let subs = ref (Pcre.exec ~rex para) in
            while true do
              Printf.printf "dup word '%s' at paragraph %d.\n"
                (Pcre.get_substring !subs 1)
                !count;
              flush stdout;
              subs := Pcre.next_match ~rex !subs;
            done
            with Not_found -> ()
          (paragraph_stream_of_channel channel);
        close_in channel
      with e ->
        close_in channel;
        raise e)
    files

let () =
  match List.tl (Array.to_list Sys.argv) with
  | [] -> find_dup_words ["-"]
  | files -> find_dup_words files

(*-----*)

(*
  This is a test
  test of the duplicate word finder.
*)

(* dup word 'test' at paragraph 1. *)

(*-----*)

let a = "nobody"
let b = "bodysnatcher"
let () =
  try
    let subs =
      Pcre.exec
        ~pat:"^(\\w+)(\\w+) \\2(\\w+)$"
        (a ^ " " ^ b) in

```

```

Printf.printf "%s overlaps in %s-%s-%s\n"
  (Pcre.get_substring subs 2)
  (Pcre.get_substring subs 1)
  (Pcre.get_substring subs 2)
  (Pcre.get_substring subs 3)
with Not_found ->
  ()

(* body overlaps in no-body-snatcher *)

(*-----*)

#!/usr/bin/ocaml
(* prime_pattern --
find prime factors of argument using pattern matching *)
#directory "+pcre";;
#load "pcre.cma";;

let arg = try int_of_string Sys.argv.(1) with _ -> 0
let n = ref (String.make arg 'o')
let rex = Pcre.regexp "(oo+?)\\1+$"
let templ = "o"
let () =
  try
    while true do
      let pat = Pcre.get_substring (Pcre.exec ~rex !n) 1 in
      Printf.printf "%d " (String.length pat);
      n := Pcre.replace ~pat ~templ !n
    done
  with Not_found ->
    Printf.printf "%d\n" (String.length !n)

(*-----*)

exception Found of (int * int * int)
let () =
  try
    match
      Pcre.extract
        ~full_match:false
        ~pat:"^(o*)\\1{11}(o*)\\2{14}(o*)\\3{15}$"
        (String.make 281 'o')
    with
      | [| x; y; z |] -> raise (Found
                               (String.length x,
                               String.length y,
                               String.length z))
      | _ -> raise Not_found
  with
    | Found (x, y, z) ->
      Printf.printf "One solution is: x=%d; y=%d; z=%d.\n"
        x y z
    | Not_found ->

```

```

    Printf.printf "No solution.\n"

(* One solution is: x=17; y=3; z=2. *)

(*-----*)

~pat:"^(o+)\{11}(o+)\{2{14}(o+)\{3{15}$"
(* One solution is: x=17; y=3; z=2. *)

~pat:"^(o*?)\{11}(o*)\{2{14}(o*)\{3{15}$"
(* One solution is: x=0; y=7; z=11. *)

~pat:"^(o+?)\{11}(o*)\{2{14}(o*)\{3{15}$"
(* One solution is: x=1; y=3; z=14. *)

```

## 6.17 Expressing AND, OR, and NOT in a Single Pattern

```

#directory "+pcre";
#load "pcre.cma";

let pat = input_line config_channel
let () = if Pcre.pmatch ~pat data then (* ... *) ()

(*-----*)

(* alpha OR beta *)
let regexp = Pcre.regexp "alpha|beta"

(* alpha AND beta *)
let regexp = Pcre.regexp ~flags:['DOTALL] "(?=.*alpha)(?=.*beta)"

(* alpha AND beta, no overlap *)
let regexp = Pcre.regexp ~flags:['DOTALL] "alpha.*beta|beta.*alpha"

(* NOT pat *)
let regexp = Pcre.regexp ~flags:['DOTALL] "(?!pat).*"

(* NOT bad BUT good *)
let regexp = Pcre.regexp ~flags:['DOTALL] "(?!bad).*good"

(*-----*)

let () =
  if not (Pcre.pmatch ~rex:regexp text)
  then something ()

(*-----*)

let () =
  if (Pcre.pmatch ~rex:regexp1 text) && (Pcre.pmatch ~rex:regexp2 text)
  then something ()

(*-----*)

```

```

let () =
  if (Pcre.pmatch ~rex:rexexp1 text) || (Pcre.pmatch ~rex:rexexp2 text)
  then something ()

(*-----*)

#!/usr/bin/ocaml
(* minigrep - trivial grep *)
#directory "+pcre";;
#load "pcre.cma";;

let line_stream_of_channel channel =
  Stream.from
    (fun _ -> try Some (input_line channel) with End_of_file -> None)

let minigrep pat files =
  let rex =
    try Pcre.regexp pat
    with Pcre.BadPattern (msg, _) ->
      Printf.eprintf "%s: Bad pattern %s: %s\n" Sys.argv.(0) pat msg;
      exit 1 in
  let process file =
    let channel = if file = "-" then stdin else open_in file in
    try
      Stream.iter
        (fun line -> if Pcre.pmatch ~rex line then print_endline line)
        (line_stream_of_channel channel);
      close_in channel
    with e ->
      close_in channel;
      raise e in
  List.iter process files

let () =
  match List.tl (Array.to_list Sys.argv) with
  | pat :: [] -> minigrep pat ["-"]
  | pat :: files -> minigrep pat files
  | [] -> Printf.eprintf "usage: %s pat [files]\n" Sys.argv.(0)

(*-----*)

let string = "labelled"

let () =
  Printf.printf "%b\n"
    (Pcre.pmatch
     ~rex:(Pcre.regexp ~flags:['DOTALL] "^(?=.*bell)(?=.*lab)")
     string)

let () =
  Printf.printf "%b\n"
    (Pcre.pmatch ~pat:"bell" string && Pcre.pmatch ~pat:"lab" string)

```

```

let () =
  if (Pcre.pmatch
      ~rex:(Pcre.regexp ~flags:['DOTALL; 'EXTENDED] "
        ^
        (?=
          .*
          bell
        )
        (?=
          .*
          lab
        )")
      string)
  then print_endline "Looks like Bell Labs might be in Murray Hill!"

let () =
  Printf.printf "%b\n"
    (Pcre.pmatch ~pat:"(?:^.*bell.*lab)|(?:^.*lab.*bell)" string)

let brand = "labelled"
let () =
  if (Pcre.pmatch
      ~rex:(Pcre.regexp ~flags:['DOTALL; 'EXTENDED] "
        (?:
          ^ .*?
          bell
          .*?
          lab
        )
        |
        (?:
          ^ .*?
          lab
          .*?
          bell
        )
      ") brand)
  then print_endline "Our brand has bell and lab separate."

let map = "a map of the world"
let () =
  Printf.printf "%b\n"
    (Pcre.pmatch
     ~rex:(Pcre.regexp ~flags:['DOTALL] "^((?!waldo).)*$")
     map)

let () =
  if (Pcre.pmatch
      ~rex:(Pcre.regexp ~flags:['DOTALL; 'EXTENDED] "
        ^
        (?:
          (?!

```

```

        waldo      # is he ahead of us now?
    )
    .              # is so, the negation failed
    ) *           # any character (cuzza /s)
    $             # repeat that grouping 0 or more
    # through the end of the string
") map)
then print_endline "There's no waldo here!"

(*-----*)

% w | minigrep '~(?!. *ttyp).*tchrist'

(*-----*)

Pcre.regexp ~flags:['EXTENDED] "
^                # anchored to the start
(?:            # zero-width look-ahead assertion
  .*           # any amount of anything (faster than .*)
  ttyp         # the string you don't want to find
)              # end look-ahead negation; rewind to start
.*            # any amount of anything (faster than .*)
tchrist       # now try to find Tom
"

(*-----*)

% w | grep tchrist | grep -v ttyp

(*-----*)

% grep -i 'pattern' files
% minigrep '(?i)pattern' files

```

## 6.18 Matching Multiple-Byte Characters

```

#load "str.cma";;

(* Regexp text for an EUC-JP character *)
let eucjp =
  (String.concat "\\|"
    (* EUC-JP encoding subcomponents: *)
    [
      (* ASCII/JIS-Roman (one-byte/character) *)
      "[\x00-\x7F]";

      (* half-width katakana (two bytes/char) *)
      "\x8E[\xA0-\xDF]";

      (* JIS X 0212-1990 (three bytes/char) *)
      "\x8F[\xA1-\xFE] [\xA1-\xFE]";

      (* JIS X 0208:1997 (two bytes/char) *)
      "[\xA1-\xFE] [\xA1-\xFE]";
    ]
  )

```

```

    ])

(* Match any number of EUC-JP characters preceding Tokyo *)
let regexp = Str.regexp ("\\(\\(\" ^ eu-
cjp ^ \"\\)*\\)\\(\\xC5\\xEC\\xB5\\xFE\\)")

(* Search from the beginning for a match *)
let () =
  if Str.string_match regexp string 0
  then print_endline "Found Tokyo"

(* Replace Tokyo with Osaka *)
let () =
  let buffer = Buffer.create (String.length string) in
  let start = ref 0 in
  while Str.string_match regexp string !start do
    Buffer.add_string buffer (Str.matched_group 1 string);
    Buffer.add_string buffer osaka; (* Assuming osaka is defined *)
    start := Str.match_end ();
  done;
  if !start < String.length string
  then Buffer.add_substring buffer string
    !start (String.length string - !start);
  print_endline (Buffer.contents buffer)

(* Split a multi-byte string into characters *)
let () =
  (* One character per list element *)
  let chars =
    Array.map
      (function
        | Str.Delim c -> c
        | Str.Text c -> failwith ("invalid char: " ^ c))
      (Array.of_list
        (Str.full_split
          (Str.regexp eucjp)
          string)) in
  let length = Array.length chars in
  for i = 0 to length - 1 do
    if String.length chars.(i) = 1 then
      begin
        (* Do something interesting with this one-byte character *)
      end
    else
      begin
        (* Do something interesting with this multi-byte character *)
      end
  done;
  (* Glue list back together *)
  let line = String.concat "" (Array.to_list chars) in
  print_endline line

(* Determine if an entire string is valid EUC-JP *)

```

```

let is_eucjp s =
  Str.string_match
    (Str.regexp ("\\(\" ^ eucjp ^ "\\)*$")) s 0

(* Assuming a similar string has been defined for Shift-JIS *)
let is_sjis s =
  Str.string_match
    (Str.regexp ("\\(\" ^ sjis ^ "\\)*$")) s 0

(* Convert from EUC-JP to Unicode, assuming a Hashtbl named
   euc2uni is defined with the appropriate character mappings *)
let () =
  let chars =
    Array.map
      (function
        | Str.Delim c -> c
        | Str.Text c -> failwith ("invalid char: " ^ c))
      (Array.of_list
        (Str.full_split
          (Str.regexp eucjp)
          string)) in
    let length = Array.length chars in
    for i = 0 to length - 1 do
      if Hashtbl.mem euc2uni chars.(i)
      then
        begin
          chars.(i) <- (Hashtbl.find euc2uni chars.(i))
        end
      else
        begin
          (* deal with unknown EUC->Unicode mapping here *)
        end
    end
  done;
  let line = String.concat "" (Array.to_list chars) in
  print_endline line

```

## 6.19 Matching a Valid Mail Address

```

#load "str.cma";;

(* Not foolproof, but works in most common cases. *)
let regexp =
  Str.regexp_case_fold
    "\\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\\.[A-Z][A-Z][A-Z]?[A-Z]?\\b"

let () =
  try
    while true do
      print_string "Email: ";
      flush stdout;
      let line = input_line stdin in
      try
        let start = ref 0 in

```

```

    while true do
      start := Str.search_forward regexp line !start;
      let string = Str.matched_string line in
      start := !start + String.length string;
      print_string "Found: ";
      print_endline string;
    done
  with Not_found -> ()
done
with End_of_file -> ()

```

## 6.20 Matching Abbreviations

```

#load "str.cma";;

let () =
  try
    while true do
      print_string "Action: ";
      flush stdout;
      let answer = input_line stdin in
      let regexp = Str.regexp_string_case_fold answer in
      if Str.string_match regexp "SEND" 0
      then print_endline "Action is send"
      else if Str.string_match regexp "STOP" 0
      then print_endline "Action is stop"
      else if Str.string_match regexp "ABORT" 0
      then print_endline "Action is abort"
      else if Str.string_match regexp "LIST" 0
      then print_endline "Action is list"
      else if Str.string_match regexp "EDIT" 0
      then print_endline "Action is edit"
    done
  with End_of_file -> ()

(*-----*)

(* assumes that invoke_editor, deliver_message, *)
(* file and pager are defined somewhere else. *)
let actions =
  [
    "edit", invoke_editor;
    "send", deliver_message;
    "list", (fun () -> ignore (Sys.command (pager ^ " " ^ file)));
    "abort", (fun () -> print_endline "See ya!"; exit 0);
  ]

let errors = ref 0

let () =
  try
    while true do
      print_string "Action: ";

```

```

flush stdout;
let answer = input_line stdin in
(* trim leading white space *)
let answer = Str.replace_first (Str.regexp "^[\t]+") "" answer in
(* trim trailing white space *)
let answer = Str.replace_first (Str.regexp "[\t]+$") "" answer in
let regexp = Str.regexp_string_case_fold answer in
let found = ref false in
List.iter
  (fun (action, handler) ->
    if Str.string_match regexp action 0
    then (found := true; handler ()))
  actions;
if not !found
then (incr errors; print_endline ("Unknown command: " ^ answer))
done
with End_of_file -> ()

```

## 6.21 Program: urlify

```

#!/usr/bin/ocaml
(* urlify - wrap HTML links around URL-like constructs *)
#directory "+pcre";;
#load "pcre.cma";;

let urls = "(http|telnet|gopher|file|wais|ftp)"
let ltrs = "\\w"
let gunk = "/#~:~?+=&%@!\\"-
let punc = ".:?\\"-
let any = ltrs ^ gunk ^ punc

let rex = Pcre.regexp ~flags:[`CASELESS; `EXTENDED]
(Printf.sprintf "
  \\b                # start at word boundary
  (                  # begin $1 {
    %s                # need resource and a colon
    [%s] +?          # followed by one or more
                    # of any valid character, but
                    # be conservative and take only
                    # what you need to....
  )                  # end $1 }
  (?=               # look-ahead non-consumptive assertion
  [%s]*             # either 0 or more punctuation
  [^%s]             # followed by a non-url char
  |                 # or else
  $                 # then end of the string
  )
" urls any punc any)

let templ = "<A HREF=\"\${1}\">\${1}</A>"

let () =
  try

```

```

while true do
  let line = input_line stdin in
  print_endline (Pcre.replace ~rex ~templ line)
done
with End_of_file ->
  ()

```

## 6.22 Program: tcgrep

```

% tcgrep -ril '^From: .*kate' ~/mail

(*-----*)

#!/usr/bin/ocaml
(* tcgrep: rewrite of tom christiansen's rewrite of grep *)
#load "unix.cma";

(* http://ocaml.info/home/ocaml_sources.html#pcre-ocaml *)
#directory "+pcre";
#load "pcre.cma";

(* http://alain.frisch.fr/soft.html#Getopt *)
#directory "+getopt";
#load "getopt.cma";

(* Initialize globals. *)
let me = Pcre.replace ~pat:".*/" Sys.argv.(0)
let matches = ref 0
let errors = ref 0
let grand_total = ref 0
let mult = ref false
let compress = [".z", "zcat"; ".gz", "zcat"; ".Z", "zcat"]

(* Prints usage and exits. *)
let usage () =
  Printf.eprintf "usage: %s [flags] [files]

Standard grep options:
  i  case insensitive
  n  number lines
  c  give count of lines matching
  C  ditto, but >1 match per line possible
  w  word boundaries only
  s  silent mode
  x  exact matches only
  v  invert search sense (lines that DON'T match)
  h  hide filenames
  e  expression (for exprs beginning with -)
  f  file with expressions
  l  list filenames matching

Specials:
  1  1 match per file

```

```

H   highlight matches
u   underline matches
r   recursive on directories or dot if none
t   process directories in 'ls -t' order
p   paragraph mode (default: line mode)
P   ditto, but specify separator, e.g. -P '%%'
a   all files, not just plain text files (not implemented)
q   quiet about failed file and dir opens
T   trace files as opened

```

May use a TCGREP environment variable to set default options.

```

" me;
  exit 255

(* Produces a stream of lines from an input channel. *)
let line_stream_of_channel channel =
  Stream.from
    (fun _ -> try Some (input_line channel) with End_of_file -> None)

(* Produces a stream of chunks from an input channel given a delimiter. *)
let delimited_stream_of_channel delim channel =
  let lines = line_stream_of_channel channel in
  let rec next para_lines i =
    match Stream.peek lines, para_lines with
    | None, [] -> None
    | Some delim', [] when delim' = delim ->
      Stream.junk lines; next para_lines i
    | Some delim', _ when delim' = delim ->
      Some (String.concat "\n" (List.rev para_lines))
    | None, _ ->
      Some (String.concat "\n" (List.rev para_lines))
    | Some line, _ -> Stream.junk lines; next (line :: para_lines) i in
  Stream.from (next [])

(* An empty delimiter corresponds to an empty line, so we can create
   a paragraph stream in terms of the previous function. *)
let paragraph_stream_of_channel = delimited_stream_of_channel ""

(* By default, the stream builder will produce lines. This can be changed
   by the -p and -P options. *)
let stream_of_channel = ref line_stream_of_channel

(* Type for command-line options and their values. *)
type opt = OBool of bool | OStr of string

(* Set an option. *)
let opt_set opt c =
  Hashtbl.replace opt c (OBool true)

(* Test an option. *)
let opt_test opt c =
  try
    match Hashtbl.find opt c with

```

```

    | OBool b -> b
    | OStr "" -> false
    | OStr _ -> true
with Not_found ->
  false

(* Convert an option to a string. *)
let opt_str opt c =
  try
    match Hashtbl.find opt c with
    | OBool b -> string_of_bool b
    | OStr s -> s
  with Not_found ->
    ""

(* Gets terminal escape characters. *)
let tput cap =
  let ch = Unix.open_process_in ("tput " ^ cap) in
  try
    let result = input_line ch in
    ignore (Unix.close_process_in ch);
    result
  with
    | End_of_file ->
      ignore (Unix.close_process_in ch);
      ""
    | e ->
      ignore (Unix.close_process_in ch);
      raise e

(* Splits a filename into its base and extension. *)
let splitext name =
  try
    let base = Filename.chop_extension name in
    let i = String.length base in
    let ext = String.sub name i (String.length name - i) in
    base, ext
  with Invalid_argument _ ->
    name, ""

(* Parses command-line arguments. *)
let parse_args () =
  let opt = Hashtbl.create 0 in
  let args = ref [] in

  let optstring = "incCwsxvhe:f:l1HurtpP:aqT" in
  let optstream = Stream.of_string optstring in

  (* Prepare options for Getopt. *)
  let opts =
    let str_setter c =
      (c, Getopt.nolong,
       None,

```

```

    Some (fun s -> Hashtbl.replace opt c (OStr s))) in
let int_setter c =
  (c, Getopt.nolong,
   Some (fun () -> Hashtbl.replace opt c (OBool true)),
   None) in
let rec loop acc =
  match Stream.peek optstream with
  | Some c ->
    (Stream.junk optstream;
     match Stream.peek optstream with
     | Some ':' ->
       Stream.junk optstream;
       loop (str_setter c :: acc)
     | _ ->
       loop (int_setter c :: acc))
  | None -> List.rev acc in
loop [] in

(* Handle TCGREP environment variable. *)
let cmdline = ref (List.tl (Array.to_list Sys.argv)) in
Array.iter
  (fun env ->
   if (String.length env > 7
       && String.sub env 0 7 = "TCGREP=")
   then
     begin
       let s = String.sub env 7 (String.length env - 7) in
       let s = if s.[0] <> '-' then "-" ^ s else s in
       cmdline := s :: !cmdline
     end)
  (Unix.environment ());
let cmdline = Array.of_list !cmdline in

(* Parse command-line options using Getopt. *)
begin
  try
    Getopt.parse
      opts (fun arg -> args := arg :: !args)
      cmdline 0 (Array.length cmdline - 1);
    args := List.rev !args
  with Getopt.Error e ->
    prerr_endline e;
    usage ()
end;

(* Read patterns from file or command line. *)
let patterns =
  if opt_test opt 'f'
  then
    begin
      let in_channel =
        try open_in (opt_str opt 'f')
        with e ->

```

```

        Printf.eprintf "%s: can't open %s: %s\n"
            me (opt_str opt 'f') (Printexc.to_string e);
        exit 255 in
    try
        let acc = ref [] in
        Stream.iter
            (fun pat -> acc := pat :: !acc)
            (line_stream_of_channel in_channel);
        close_in in_channel;
        List.rev !acc
    with e ->
        close_in in_channel;
        Printf.eprintf "%s: error reading %s: %s\n"
            me (opt_str opt 'f') (Printexc.to_string e);
        exit 255
    end
else if opt_test opt 'e'
then [opt_str opt 'e']
else [match !args with h :: t -> (args := t; h) | [] -> usage ()] in

(* Terminal escape characters for highlighting options. *)
let highlight =
    if opt_test opt 'H'
    then tput "sms0" ^ "$1" ^ tput "rms0"
    else if opt_test opt 'u'
    then tput "smul" ^ "$1" ^ tput "rmul"
    else "$1" in

(* Regular expression flags to use. *)
let flags = ref [] in
if opt_test opt 'i' then flags := 'CASELESS :: !flags;

(* Options for paragraph modes. *)
if opt_test opt 'p'
then stream_of_channel := paragraph_stream_of_channel;
if opt_test opt 'P'
then stream_of_channel := delimited_stream_of_channel (opt_str opt 'P');

(* Word boundary option. *)
let patterns =
    if opt_test opt 'w'
    then List.map (fun pat -> "\\b" ^ pat ^ "\\b") patterns
    else patterns in

(* Exact match option. *)
let patterns =
    if opt_test opt 'x'
    then List.map (fun pat -> "^" ^ pat ^ "$") patterns
    else patterns in

(* Options that imply other options. *)
if opt_test opt 'l' then opt_set opt '1';
if opt_test opt 'u' then opt_set opt 'H';

```

```

if opt_test opt 'C' then opt_set opt 'c';
if opt_test opt 'c' then opt_set opt 's';
if opt_test opt 's' && not (opt_test opt 'c') then opt_set opt '1';

(* Compile the regular expression patterns. *)
let rexes =
  List.map
    (fun pat ->
      try Pcre.regexp ~flags:!flags (" " ^ pat ^ " ")
      with Pcre.BadPattern (msg, _) ->
        Printf.eprintf "%s: bad pattern %s: %s\n" me pat msg;
        exit 255)
    patterns in

(* Increments the matches variable by the number of matches
(or non-matches) in the given line. *)
let count_matches line =
  if opt_test opt 'v'
  then fun rex ->
    (if not (Pcre.pmatch ~rex line) then incr matches)
  else if opt_test opt 'C'
  then fun rex ->
    (matches := !matches + (try Array.length (Pcre.extract_all ~rex line)
      with Not_found -> 0))
  else fun rex ->
    (if Pcre.pmatch ~rex line then incr matches) in

(* Counts matches in a line and returns the line with any
necessary highlighting. *)
let matcher line =
  List.iter (count_matches line) rexes;
  if opt_test opt 'H'
  then
    List.fold_left
      (fun line rex ->
        Pcre.replace ~rex ~templ:highlight line)
      line rexes
  else line in

(* List of files or directories to process. *)
let files =
  match !args with
  | [] -> if opt_test opt 'r' then ["."] else ["-"]
  | [arg] -> [arg]
  | args -> (mult := true; args) in

(* Overrides for options that affect the multiple-file flag. *)
if opt_test opt 'h' then mult := false;
if opt_test opt 'r' then mult := true;

(* Return the three values to be processed by matchfiles. *)
opt, matcher, files

```

```

(* Used to break out of loops and abort processing of the current file. *)
exception NextFile

(* Runs given matcher on a list of files using the specified options. *)
let rec matchfiles opt matcher files =
  (* Handles a single directory. *)
  let matchdir dir =
    if not (opt_test opt 'r')
    then
      begin
        if opt_test opt 'T'
        then (Printf.eprintf "%s: \"%s\" is a directory, but no -r given\n"
              me dir;
              flush stderr)
        end
      else
        begin
          let files =
            try Some (Sys.readdir dir)
            with e ->
              if not (opt_test opt 'q')
              then (Printf.eprintf "%s: can't readdir %s: %s\n"
                    me dir (Printexc.to_string e);
                    flush stderr);
                    incr errors;
                    None in
            match files with
            | Some files ->
              let by_mtime a b =
                compare
                  (Unix.stat (Filename.concat dir b)).Unix.st_mtime
                  (Unix.stat (Filename.concat dir a)).Unix.st_mtime in
              if opt_test opt 't'
              then Array.sort by_mtime files;
              matchfiles opt matcher
                (Array.to_list
                 (Array.map
                  (Filename.concat dir)
                  files))
            | None -> ()
          end in
        end in
    (* Handles a single file. *)
    let matchfile file =
      (* Keep a running total of matches for this file. *)
      let total = ref 0 in

      (* Keep track of the current line number. *)
      let line_num = ref 0 in

      (* Shadow close_in to properly close process channels for compressed
      files and avoid closing stdin. *)
      let process_open = ref false in

```

```

let close_in channel =
  if !process_open
  then ignore (Unix.close_process_in channel)
  else if channel != stdin then close_in channel in

(* Process a line (or paragraph, with -p or -P) of input. *)
let matchline line =
  incr line_num;
  matches := 0;
  let line = matcher line in
  if !matches > 0
  then
    begin
      total := !total + !matches;
      grand_total := !grand_total + !matches;
      if opt_test opt 'l'
      then (print_endline file; raise NextFile)
      else if not (opt_test opt 's')
      then (Printf.printf "%s%s%s%s\n"
              (if !mult
                then file ^ ":"
                else "")
              (if opt_test opt 'n'
                then (string_of_int !line_num
                     ^ (if opt_test opt 'p' || opt_test opt 'P'
                        then ":\n"
                        else ":"))
                else ""))
              line
              (if opt_test opt 'p' || opt_test opt 'P'
                then "\n" ^ String.make 20 '-'
                else ""));
      flush stdout);
      if opt_test opt '1' then raise NextFile
    end in

(* Get a channel for the file, starting a decompression process if
   necessary. If None, the file will be skipped. *)
let maybe_channel =
  if file = "-"
  then (if Unix.isatty Unix.stdin && not (opt_test opt 'q')
        then (Printf.eprintf "%s: reading from stdin\n" me;
              flush stderr);
        Some stdin)
  else if not (Sys.file_exists file)
  then (if not (opt_test opt 'q')
        then (Printf.eprintf "%s: file \"%s\" does not exist\n"
              me file;
              flush stderr);
        None)
  else if List.mem_assoc (snd (splitext file)) compress
  then (process_open := true;

```

```

try Some (Unix.open_process_in
          (List.assoc (snd (splitext file)) compress
                    ^ " < " ^ file))
with e ->
  if not (opt_test opt 'q')
  then (Printf.eprintf "%s: %s: %s\n" me file
        (Printexc.to_string e);
        flush stderr);
  incr errors;
  None)
else (try Some (open_in file)
      with e ->
        if not (opt_test opt 'q')
        then (Printf.eprintf "%s: %s: %s\n" me file
              (Printexc.to_string e);
              flush stderr);
        incr errors;
        None) in

(* Run matcher on the open channel, then close the channel. *)
match maybe_channel with
| None -> ()
| Some channel ->
  begin
  try
    if opt_test opt 'T'
    then (Printf.eprintf "%s: checking %s\n" me file;
          flush stderr);
    Stream.iter matchline (!stream_of_channel channel);
    close_in channel
  with
  | NextFile ->
    close_in channel
  | e ->
    close_in channel;
    raise e
  end;
  if opt_test opt 'c'
  then (Printf.printf "%s%d\n"
        (if !mult then file ^ ":" else "")
        !total;
        flush stdout) in

(* Handle each of the specified files and directories. *)
List.iter
(fun file ->
  if file = "-"
  then matchfile file
  else if try Sys.is_directory file with _ -> false
  then matchdir file
  else matchfile file)
files

```

```

(* Parse command line arguments, run matcher, and set result status. *)
let opt, matcher, files = parse_args ()
let () =
  matchfiles opt matcher files;
  if !errors > 0 then exit 2;
  if !grand_total > 0 then exit 0;
  exit 1

```

## 6.23 Regular Expression Grabbag

```

#directory "+pcre";;
#load "pcre.cma";;

(*-----*)

Pcre.pmatch
  ~rex:(Pcre.regexp
    ~flags:['CASELESS]
    "~m*(d?c{0,3}|c[dm])(l?x{0,3}|x[lc])(v?i{0,3}|i[vx])$")
  input

(*-----*)

Pcre.replace
  ~pat:"(\\S+)(\\s+)(\\S+)"
  ~templ:"$3$2$1"
  input

(*-----*)

Pcre.extract
  ~full_match:false
  ~pat:"(\\w+)\\s*=\\s*(.*)\\s*$"
  input

(*-----*)

Pcre.pmatch
  ~pat:". {80,}"
  input

(*-----*)

Pcre.extract
  ~full_match:false
  ~pat:"(\\d+)/ (\\d+)/ (\\d+) (\\d+): (\\d+): (\\d+)"
  input

(*-----*)

Pcre.replace
  ~pat:"/usr/bin"
  ~templ:"/usr/local/bin"

```

```

input

(*-----*)

Pcre.substitute_substrings
  ~pat:"%([0-9A-Fa-f][0-9A-Fa-f])"
  ~subst:(fun subs ->
    let c = Pcre.get_substring subs 1 in
    String.make 1 (Char.chr (int_of_string ("0x" ^ c))))
input

(*-----*)

Pcre.replace
  ~rex:(Pcre.regexp
    ~flags:['DOTALL; 'EXTENDED] "
    /\\" # Match the opening delimiter
    .*? # Match a minimal number of characters
    /\\" # Match the closing delimiter
    ")
input

(*-----*)

Pcre.replace ~pat:"^\\s+" input
Pcre.replace ~pat:"\\s+$" input

(*-----*)

Pcre.replace ~pat:"\\\\\\n" ~templ:"\\n" input

(*-----*)

Pcre.replace ~pat:"^.*:." input

(*-----*)

Pcre.extract
  ~full_match:false
  ~pat:"^([01]?\\d\\d|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d|2[0-4]\\d|25[0-5])\\.\\.([01]?\\d\\d|2[0-4]\\d|25[0-5])$"
input

(*-----*)

Pcre.replace ~pat:"^.*/" input

(*-----*)

let termcap = ":co#80:li#24:"
let cols =

```

```

try int_of_string (Pcre.extract ~pat:".co#(\d+):" termcap).(1)
with Not_found | Failure "int_of_string" -> 80

(*-----*)

let name =
  Pcre.replace
    ~pat:" /\S+/"
    ~templ:" "
    (" " ^ String.concat " " (Array.to_list Sys.argv))

(*-----*)

#load "unix.cma";;
let () =
  let ch = Unix.open_process_in "uname -a" in
  let os = input_line ch in
  ignore (Unix.close_process_in ch);
  if not (Pcre.pmatch ~rex:(Pcre.regexp ~flags:['CASELESS] "linux") os)
  then print_endline "This isn't Linux"

(*-----*)

Pcre.replace ~pat:"\n\s+" ~templ:" " input

(*-----*)

let nums =
  Array.map
    (fun group -> float_of_string group.(1))
    (Pcre.extract_all
      ~pat:"(\d+\.?\d*|\.\d+)"
      input)

(*-----*)

let capwords =
  Array.map
    (fun group -> group.(1))
    (Pcre.extract_all
      ~pat:"(\b[^\Wa-z0-9_]+\b)"
      input)

(*-----*)

let lowords =
  Array.map
    (fun group -> group.(1))
    (Pcre.extract_all
      ~pat:"(\b[^\WA-Z0-9_]+\b)"
      input)

(*-----*)

```

```

let icwords =
  Array.map
    (fun group -> group.(1))
    (Pcre.extract_all
      ~pat:"(\\b[^\Wa-z0-9_][^\WA-Z0-9_]*\\b)"
      input)

(*-----*)

let links =
  Array.map
    (fun group -> group.(1))
    (Pcre.extract_all
      ~rex:(Pcre.regexp
        ~flags:['DOTALL; 'CASELESS]
        "<A[^>+?HREF\\s*=\\s*[\"']?([^\"] >+?) [ '\"']?>")
      input)

(*-----*)

let initial =
  try (Pcre.extract ~pat:"^\\S+\\s+(\\S)\\S*\\s+\\S" input).(1)
  with Not_found -> ""

(*-----*)

Pcre.replace ~pat:"\"([^\"]*)\\\"" ~templ:"'$1'" input

(*-----*)

let sentences =
  Array.map
    (fun group -> group.(1))
    (Pcre.extract_all
      ~pat:"(\\S.*?\\pP)(?= |\\Z)"
      (Pcre.replace ~pat:" {3,}" ~templ:" "
        (Pcre.replace ~pat:"\\n" ~templ:" "
          (Pcre.replace ~pat:"(\\pP\\n)" ~templ:"$1 "
            input))))))

(*-----*)

Pcre.extract ~full_match:false ~pat:"(\\d{4})-(\\d\\d)-(\\d\\d)" input

(*-----*)

Pcre.pmatch
  ~pat:"^[01]?[- .]?((\\([2-9]\\d{2}\\)|[2-9]\\d{2})[- .]?\\d{3}[-
  .]?\\d{4})$"
  input

(*-----*)

```

```

Pcre.pmatch
  ~rex:(Pcre.regex
    ~flags:['CASELESS]
    "\\boh\\s+my\\s+gh?o(d(dess(es)?|s?)|odness|sh)\\b")
  input

(*-----*)

let lines =
  Array.map
    (fun group -> group.(1))
    (Pcre.extract_all
      ~pat:"([^\010\013]*) (\010\013?|\013\010?)"
      input)

```

## 7 File Access

```
#load "str.cma";;

(* Print all lines that contain the word "blue" in the input file
   /usr/local/widgets/data to stdout. *)
let () =
  let in_channel = open_in "/usr/local/widgets/data" in
  try
    while true do
      let line = input_line in_channel in
      try
        ignore (Str.search_forward (Str.regexp_string "blue") line 0);
        print_endline line
      with Not_found -> ()
    done
  with End_of_file ->
    close_in in_channel

(*-----*)

let () =
  let regexp = Str.regexp "[0-9]" in
  try
    while true do
      (* reads from stdin *)
      let line = input_line stdin in
      (* writes to stderr *)
      if not (Str.string_match regexp line 0)
      then prerr_endline "No digit found.";
      (* writes to stdout *)
      Printf.printf "Read: %s\n" line;
      flush stdout
    done
  with End_of_file ->
    close_out stdout

(*-----*)

(* Write to an output file the usual way. *)
let () =
  let logfile = open_out "/tmp/log" in
  output_string logfile "Countdown initiated...\n";
  close_out logfile;
  print_endline "You have 30 seconds to reach minimum safety distance."

(* Write to an output file using redirection. *)
#load "unix.cma";;
let () =
  let logfile = open_out "/tmp/log" in
  let old_descr = Unix.dup Unix.stdout in
  (* switch to logfile for output *)
  Unix.dup2 (Unix.descr_of_out_channel logfile) Unix.stdout;
```

```

print_endline "Countdown initiated...";
(* return to original output *)
Unix.dup2 old_descr Unix.stdout;
print_endline "You have 30 seconds to reach minimum safety distance."

```

## 7.1 Opening a File

```

(* open file "path" for reading only *)
let source =
  try open_in path
  with Sys_error msg -> failwith ("Couldn't read from " ^ msg)

(* open file "path" for writing only *)
let sink =
  try open_out path
  with Sys_error msg -> failwith ("Couldn't write to " ^ msg)

(*-----*)

#load "unix.cma";;

(* open file "path" for reading only *)
let source =
  try Unix.openfile path [Unix.O_RDONLY] 0o644
  with Unix.Unix_error (code, func, param) ->
    failwith (Printf.sprintf "Couldn't open %s for reading: %s"
      path (Unix.error_message code))

(* open file "path" for writing only *)
let sink =
  try Unix.openfile path [Unix.O_WRONLY; Unix.O_CREAT] 0o644
  with Unix.Unix_error (code, func, param) ->
    failwith (Printf.sprintf "Couldn't open %s for writing: %s"
      path (Unix.error_message code))

(*-----*)

(* open file "path" for reading and writing *)
let fh =
  try Unix.openfile filename [Unix.O_RDWR] 0o644
  with Unix.Unix_error (code, func, param) ->
    failwith (Printf.sprintf "Couldn't open %s for read and write: %s"
      filename (Unix.error_message code))

(*-----*)

(* open file "path" read only *)
let fh = open_in path
let fh = Unix.openfile path [Unix.O_RDONLY] 0o644

(*-----*)

(* open file "path" write only, create it if it does not exist *)

```

```

let fh = open_out path
let fh = Unix.openfile path [Unix.O_WRONLY; Unix.O_TRUNC; Unix.O_CREAT] 0o600

(*-----*)

(* open file "path" write only, fails if file exists *)
let fh = Unix.openfile path [Unix.O_WRONLY; Unix.O_EXCL; Unix.O_CREAT] 0o600

(*-----*)

(* open file "path" for appending *)
let fh =
  open_out_gen [Open_wronly; Open_append; Open_creat] 0o600 path
let fh =
  Unix.openfile path [Unix.O_WRONLY; Unix.O_APPEND; Unix.O_CREAT] 0o600

(*-----*)

(* open file "path" for appending only when file exists *)
let fh = Unix.openfile path [Unix.O_WRONLY; Unix.O_APPEND] 0o600

(*-----*)

(* open file "path" for reading and writing *)
let fh = Unix.openfile path [Unix.O_RDWR] 0o600

(*-----*)

(* open file "path" for reading and writing,
  create a new file if it does not exist *)
let fh = Unix.openfile path [Unix.O_RDWR; Unix.O_CREAT] 0o600

(*-----*)

(* open file "path" for reading and writing, fails if file exists *)
let fh = Unix.openfile path [Unix.O_RDWR; Unix.O_EXCL; Unix.O_CREAT] 0o600

```

## 7.2 Opening Files with Unusual Filenames

```
(* Nothing different needs to be done with OCaml *)
```

## 7.3 Expanding Tildes in Filenames

```

#load "str.cma";;
#load "unix.cma";;

let expanduser =
  let regexp = Str.regexp "~\\([~/]*\\)" in
  let replace s =
    match Str.matched_group 1 s with
    | "" ->
      (try Unix.getenv "HOME"
       with Not_found ->

```

```

        (try Unix.getenv "LOGDIR"
         with Not_found ->
          (Unix.getpwuid (Unix.getuid ())).Unix.pw_dir))
    | user -> (Unix.getpwnam user).Unix.pw_dir in
Str.substitute_first regexp replace

(*-----*)

~user
~user/blah
~
~/blah

```

## 7.4 Making Perl Report Filenames in Errors

```

#load "unix.cma";;

open Unix

(* Raises an exception on failure. *)
let file = openfile filename [ O_RDONLY ] 0o640

exception ErrString of string

let file =
  try openfile filename [ O_RDONLY ] 0o640
  with Unix_error (e, f, n) ->
    raise (ErrString
           (Printf.sprintf "Could not open %s for read: %s"
                            n (error_message e)))

```

## 7.5 Creating Temporary Files

```

(* Open a new temporary file for writing. Filename.open_temp_file
safeguards against race conditions and returns both the filename
and an output channel. *)
let name, out_channel = Filename.open_temp_file "prefix-" ".suffix"

(* Install an at_exit handler to remove the temporary file when this
program exits. *)
let () = at_exit (fun () -> Sys.remove name)

(*-----*)

#load "unix.cma";;

let () =
  (* Open a temporary file for reading and writing. *)
  let name = Filename.temp_file "prefix-" ".suffix" in
  let descr = Unix.openfile name [Unix.O_RDWR] 0o600 in

  (* Write ten lines of output. *)
  let out_channel = Unix.out_channel_of_descr descr in

```

```

for i = 1 to 10 do
  Printf.fprintf out_channel "%d\n" i
done;
flush out_channel;

(* Seek to the beginning and read the lines back in. *)
let in_channel = Unix.in_channel_of_descr descr in
seek_in in_channel 0;
print_endline "Tmp file has:";
let rec loop () =
  print_endline (input_line in_channel);
  loop () in
try loop() with End_of_file -> ();

(* Close the underlying file descriptor and remove the file. *)
Unix.close descr;
Sys.remove name

```

## 7.6 Storing Files Inside Your Program Text

```

#load "str.cma";;

let main data =
  List.iter
    (fun line ->
      (* process the line *)
      ())
    (Str.split (Str.regexp "\n") data)

let () = main "\
your data goes here
"

```

## 7.7 Writing a Filter

```

#load "str.cma";;

let parse_args () =
  match List.tl (Array.to_list Sys.argv) with
  | [] -> ["-"]
  | args -> args

let run_filter func args =
  List.iter
    (fun arg ->
      let in_channel =
        match arg with
        | "-" -> stdin
        | arg -> open_in arg in
      try
        begin
          try
            while true do

```

```

        func (input_line in_channel)
        done
        with End_of_file -> ()
        end;
        close_in in_channel
    with e ->
        close_in in_channel;
        raise e)
    args

let () =
    run_filter
    (fun line ->
        (* do something with the line *)
        ())
    (parse_args ())

(*-----*)

(* arg demo 1: Process optional -c flag *)
let chop_first = ref false
let args =
    match parse_args () with
    | "-c" :: rest -> chop_first := true; rest
    | args -> args

(* arg demo 2: Process optional -NUMBER flag *)
let columns = ref None
let args =
    match parse_args () with
    | arg :: rest
      when Str.string_match (Str.regexp "^-\\([0-9]+\\)$") arg 0 ->
        columns := Some (int_of_string (Str.matched_group 1 arg));
        rest
    | args -> args

(* arg demo 3: Process clustering -a, -i, -n, or -u flags *)
let append = ref false
let ignore_ints = ref false
let nostdout = ref false
let unbuffer = ref false
let args =
    let rec parse_flags = function
        | "" -> ()
        | s ->
            (match s.[0] with
             | 'a' -> append := true
             | 'i' -> ignore_ints := true
             | 'n' -> nostdout := true
             | 'u' -> unbuffer := true
             | _ ->
                Printf.eprintf "usage: %s [-ainu] [filenames] ...\n"
                    Sys.argv.(0);
            )
    in
    parse_flags args

```

```

        flush stderr;
        exit 255);
    parse_flags (String.sub s 1 (String.length s - 1)) in
List.rev
  (List.fold_left
   (fun acc ->
    function
      | "" -> acc
      | s when s.[0] = '-' ->
        parse_flags (String.sub s 1 (String.length s - 1));
        acc
      | arg -> arg :: acc)
   []
   (parse_args ()))

(*-----*)

(* findlogin - print all lines containing the string "login" *)

let () =
  run_filter
    (fun line ->
     if Str.string_match (Str.regexp ".*login.*") line 0
     then print_endline line)
    (parse_args ())

(*-----*)

(* lowercase - turn all lines into lowercase *)

let () =
  run_filter
    (fun line -> print_endline (String.lowercase line))
    (parse_args ())

(*-----*)

(* countchunks - count how many words are used *)

let chunks = ref 0
let () =
  run_filter
    (fun line ->
     if line <> "" && line.[0] == '#'
     then ()
     else chunks := !chunks
      + List.length (Str.split (Str.regexp "[ \\t]+") line))
    (parse_args ());
  Printf.printf "Found %d chunks\n" !chunks

```

## 7.8 Modifying a File in Place with Temporary File

```
(* Modify a file in place. *)
```

```

let modify func old new' =
  let old_in = open_in old in
  let new_out = open_out new' in
  begin
    try
      while true do
        let line = input_line old_in in
        func new_out line
      done
    with End_of_file -> ()
  end;
close_in old_in;
close_out new_out;
Sys.rename old (old ^ ".orig");
Sys.rename new' old

(* Insert lines at line 20. *)
let () =
  let count = ref 0 in
  modify
    (fun out line ->
      incr count;
      if !count = 20
      then (output_string out "Extra line 1\n";
            output_string out "Extra line 2\n");
      output_string out line;
      output_string out "\n")
  old new'

(* Delete lines 20..30. *)
let () =
  let count = ref 0 in
  modify
    (fun out line ->
      incr count;
      if !count < 20 || !count > 30
      then (output_string out line;
            output_string out "\n"))
  old new'

```

## 7.9 Modifying a File in Place with -i Switch

```
(* An equivalent of Perl's -i switch does not exist in OCaml. *)
```

## 7.10 Modifying a File in Place Without a Temporary File

```

#load "str.cma";;
#load "unix.cma";;

(* Modify a file in place. *)
let modify func file =
  let in' = open_in file in
  let lines = ref [] in

```

```

begin
  try
    while true do
      let line = input_line in' in
      lines := func line :: !lines
    done
    with End_of_file -> ()
  end;
  close_in in';
  let lines = List.rev !lines in
  let out = open_out file in
  List.iter
    (fun line ->
      output_string out line;
      output_string out "\n")
    lines;
  close_out out

(* Replace DATE with the current date. *)
let () =
  let tm = Unix.localtime (Unix.time ()) in
  let date = Printf.sprintf "%02d/%02d/%04d"
    (tm.Unix.tm_mon + 1)
    tm.Unix.tm_mday
    (tm.Unix.tm_year + 1900) in
  modify
    (Str.global_replace (Str.regexp "DATE") date)
  infile

```

## 7.11 Locking a File

```

#load "unix.cma";;

let descr = Unix.openfile path [Unix.O_RDWR] 0o664

let () =
  Unix.lockf descr Unix.F_LOCK 0;
  (* update file, then ... *)
  Unix.close descr

let () =
  try Unix.lockf descr Unix.F_TLOCK 0
  with Unix.Unix_error (error, _, _) ->
    Printf.eprintf
      "can't immediately write-lock the file (%s), blocking ...\n"
      (Unix.error_message error);
    flush stderr;
    Unix.lockf descr Unix.F_LOCK 0

(*-----*)

#load "unix.cma";;

```

```

let descr = Unix.openfile "numfile" [Unix.O_RDWR; Unix.O_CREAT] 0o664

let () =
  Unix.lockf descr Unix.F_LOCK 0;
  (* Now we have acquired the lock, it's safe for I/O *)
  let num =
    try int_of_string (input_line (Unix.in_channel_of_descr descr))
    with _ -> 0 in
  ignore (Unix.lseek descr 0 Unix.SEEK_SET);
  Unix.ftruncate descr 0;
  let out = Unix.out_channel_of_descr descr in
  output_string out (string_of_int (num + 1));
  output_string out "\n";
  flush out;
  Unix.close descr

```

## 7.12 Flushing Output

```

(* OCaml automatically flushes after calling these functions: *)
let () =
  print_endline "I get flushed.";
  print_newline (); (* Me too! *)
  prerr_endline "So do I.";
  prerr_newline () (* As do I. *)

(* The Printf functions allow a format specifier of "%!" to trigger
   an immediate flush. *)
let () = Printf.printf "I flush %s%! and %s!\n%!" "here" "there"

(*-----*)

(* seeme - demo stdio output buffering *)
#load "unix.cma";;
let () =
  output_string stdout "Now you don't see it...";
  Unix.sleep 2;
  print_endline "now you do"

(*-----*)

(* A channel can be explicitly flushed: *)
let () = flush stderr

(* All channels can be flushed at once (errors are ignored): *)
let () = flush_all ()

(* Closing a channel flushes automatically: *)
let () =
  output_string stdout "I get written.\n";
  close_out stdout

(* Calls to exit result in a flush_all, and exit is always called at
   termination even if an error occurs. *)

```

```

let () =
  output_string stderr "Bye!\n";
  exit 0

```

### 7.13 Reading from Many Filehandles Without Blocking

```

#load "unix.cma";;

let () =
  (* list all file descriptors to poll *)
  let readers = [file_descr1; file_descr2; file_descr3] in
  let ready, _, _ = Unix.select readers [] [] 0.0 in
  (* input waiting on the filehandles in "ready" *)
  ()

(*-----*)

let () =
  let in_channel = Unix.in_channel_of_descr file_descr in
  let found, _, _ = Unix.select [file_descr] [] [] 0.0 (* just check *) in
  match found with
  | [] -> ()
  | _ ->
    let line = input_line in_channel in
    Printf.printf "I read %s\n%!" line

```

### 7.14 Doing Non-Blocking I/O

```

#load "unix.cma";;

(* Pass the O_NONBLOCK flag when calling Unix.openfile. *)
let file_descr =
  try Unix.openfile "/dev/cua0" [Unix.O_RDWR; Unix.O_NONBLOCK] 0o666
  with Unix.Unix_error (code, func, param) ->
    Printf.eprintf "Can't open modem: %s\n" (Unix.error_message code);
    exit 2

(*-----*)

(* If the file descriptor already exists, use Unix.set_nonblock. *)
let () = Unix.set_nonblock file_descr

(*-----*)

(* In non-blocking mode, calls that would block throw exceptions. *)
let () =
  let chars_written =
    try
      Some (Unix.single_write file_descr buffer 0 (String.length buffer))
    with
      | Unix.Unix_error (Unix.EAGAIN, _, _)
      | Unix.Unix_error (Unix.EWOULDBLOCK, _, _) -> None in
  match chars_written with

```

```

    | Some n when n = String.length buffer ->
      (* successfully wrote *)
      ()
    | Some n ->
      (* incomplete write *)
      ()
    | None ->
      (* would block *)
      ()

let () =
  let chars_read =
    try
      Some (Unix.read file_descr buffer 0 buffer_size)
    with
      | Unix.Unix_error (Unix.EAGAIN, _, _)
      | Unix.Unix_error (Unix.EWOULDBLOCK, _, _) -> None in
  match chars_read with
  | Some n ->
    (* successfully read n bytes from file_descr *)
    ()
  | None ->
    (* would block *)
    ()

```

## 7.15 Determining the Number of Bytes to Read

```

#load "unix.cma";;

(* OCaml does not expose the FIONREAD ioctl call. It's better to use
non-blocking reads anyway. There is the following function in
Pervasives which gives you the length of an input channel, but it
works by doing a seek so it only works on regular files: *)

let () =
  let length = in_channel_length in_channel in
  (* ... *)
  ()

```

## 7.16 Storing Filehandles in Variables

```

(* Channels and file descriptors are ordinary, first-class values in
OCaml. No special contortions are necessary to store them in data
structures, pass them as arguments, etc. *)

```

## 7.17 Caching Open Output Filehandles

```

module FileCache = struct
  let isopen = Hashtbl.create 0
  let maxopen = ref 16

  let resize () =
    if Hashtbl.length isopen >= !maxopen

```

```

then
  begin
    let newlen = !maxopen / 3 in
    let items = ref [] in
    Hashtbl.iter
      (fun filename (chan, count) ->
        items := (count, filename, chan) :: !items)
      isopen;
    let items = Array.of_list !items in
    Array.sort compare items;
    let pivot = Array.length items - newlen in
    for i = 0 to Array.length items - 1 do
      let (count, filename, chan) = items.(i) in
      if i < pivot
      then (close_out chan;
            Hashtbl.remove isopen filename)
           else (Hashtbl.replace isopen filename (chan, 0))
      done
    end

    let output ?(mode=[Open_creat; Open_append]) ?(perm=0o640) file-
name data =
      let (chan, count) =
        try Hashtbl.find isopen filename
        with Not_found ->
          resize ();
          (open_out_gen mode perm filename, 0) in
      output_string chan data;
      flush chan;
      Hashtbl.replace isopen filename (chan, count + 1)

    let close filename =
      try
        match Hashtbl.find isopen filename with (chan, _) ->
          close_out chan;
          Hashtbl.remove isopen filename
        with Not_found -> ()
      end

    (*-----*)

    (* splitwulog - split wuftp log by authenticated user *)
    #load "str.cma";;
    let outdir = "/var/log/ftp/by-user"
    let regexp = Str.regexp " "
    let () =
      try
        while true do
          let line = input_line stdin in
          let chunks = Array.of_list (Str.split regexp line) in
          let user = chunks.(Array.length chunks - 5) in
          let path = Filename.concat outdir user in
          FileCache.output path (line ^ "\n")
        end
      with _ -> ()
  end

```

```

done
with End_of_file -> ()

```

## 7.18 Printing to Many Filehandles Simultaneously

```

(* Save your channels in a list and iterate through them normally. *)
let () =
  List.iter
    (fun channel ->
      output_string channel stuff_to_print)
    channels

(* For convenience, you can define a helper function and use currying. *)
let write_data channel = output_string channel data
let () = List.iter (write stuff_to_print) channels

(*-----*)

(* Open a pipe to "tee". Requires a Unix environment. *)
#load "unix.cma";;
let () =
  let channel =
    Unix.open_process_out "tee file1 file2 file3 >/dev/null" in
  output_string channel "whatever\n";
  ignore (Unix.close_process_out channel)

(*-----*)

(* Redirect standard output to a tee. *)
let () =
  let reader, writer = Unix.pipe () in
  match Unix.fork () with
  | 0 ->
    Unix.close writer;
    Unix.dup2 reader Unix.stdin;
    Unix.close reader;
    Unix.execvp "tee" [| "tee"; "file1"; "file2"; "file3" |]
  | pid ->
    Unix.close reader;
    Unix.dup2 writer Unix.stdout;
    Unix.close writer
let () =
  print_endline "whatever";
  close_out stdout;
  ignore (Unix.wait ())

```

## 7.19 Opening and Closing File Descriptors by Number

```

(* An abstraction barrier exists between file descriptor numbers and
   file_descr values, but Ocamlnet provides functions in the Netsys
   module to map between the two. *)
#load "unix.cma";;
#directory "+netsys";;

```

```

#load "netsys.cma";;

(* Open the descriptor itself. *)
let file_descr = Netsys.file_descr_of_int fdnum
let in_channel = Unix.in_channel_of_descr file_descr

(* Open a copy of the descriptor. *)
let file_descr = Unix.dup (Netsys.file_descr_of_int fdnum)
let in_channel = Unix.in_channel_of_descr file_descr

(* After processing... *)
let () = close_in in_channel

```

## 7.20 Copying Filehandles

```

#load "unix.cma";;

let () =
  (* Take copies of the file descriptors. *)
  let oldout = Unix.dup Unix.stdout in
  let olderr = Unix.dup Unix.stderr in

  (* Redirect stdout and stderr. *)
  let output =
    Unix.openfile
      "/tmp/program.out"
      [Unix.O_WRONLY; Unix.O_CREAT; Unix.O_TRUNC]
      0o666 in
  Unix.dup2 output Unix.stdout;
  Unix.close output;
  let copy = Unix.dup Unix.stdout in
  Unix.dup2 copy Unix.stderr;
  Unix.close copy;

  (* Run the program. *)
  ignore (Unix.system joe_random_process);

  (* Close the redirected file handles. *)
  Unix.close Unix.stdout;
  Unix.close Unix.stderr;

  (* Restore stdout and stderr. *)
  Unix.dup2 oldout Unix.stdout;
  Unix.dup2 olderr Unix.stderr;

  (* Avoid leaks by closing the independent copies. *)
  Unix.close oldout;
  Unix.close olderr

```

## 7.21 Program: netlock

```
drivelock.ml:
```

```

#!/usr/bin/ocaml
(* drivelock - demo LockDir module *)
#use "netlock.ml";;
let die msg = prerr_endline msg; exit 1
let () =
  Sys.set_signal Sys.sigint
    (Sys.Signal_handle (fun _ -> die "outta here"));
  LockDir.debug := true;
  let path =
    try Sys.argv.(1)
    with Invalid_argument _ ->
      die ("usage: " ^ Sys.argv.(0) ^ " <path>") in
    (try LockDir.nflock ~naptime:2 path
     with LockDir.Error _ ->
       die ("couldn't lock " ^ path ^ " in 2 seconds"));
  Unix.sleep 100;
  LockDir.nunflock path

(*-----*)

netlock.ml:

#load "unix.cma";;

(* module to provide very basic filename-level *)
(* locks. No fancy systems calls. In theory, *)
(* directory info is sync'd over NFS. Not *)
(* stress tested. *)

module LockDir :
sig
  exception Error of string

  val debug : bool ref
  val check : int ref
  val nflock : ?naptime:int -> string -> unit
  val nunflock : string -> unit
end = struct

  exception Error of string

  let debug = ref false
  let check = ref 1

  module StringSet = Set.Make(String)
  let locked_files = ref StringSet.empty

  (* helper function *)
  let name2lock pathname =
    let dir = Filename.dirname pathname in
    let file = Filename.basename pathname in

```

```

let dir = if dir = "." then Sys.getcwd () else dir in
let lockname = Filename.concat dir (file ^ ".LOCKDIR") in
lockname

let nflock ?(naptime=0) pathname =
  let lockname = name2lock pathname in
  let whosegot = Filename.concat lockname "owner" in
  let start = Unix.time () in
  let missed = ref 0 in

  (* if locking what I've already locked, raise exception *)
  if StringSet.mem pathname !locked_files
  then raise (Error (pathname ^ " already locked"));

  Unix.access (Filename.dirname pathname) [Unix.W_OK];

begin
  try
    while true do
      try
        Unix.mkdir lockname 0o777;
        raise Exit
      with Unix.Unix_error (e, _, _) ->
        incr missed;
        if !missed > 10
        then raise (Error
                    (Printf.sprintf "can't get %s: %s"
                     lockname (Unix.error_message e)));
        if !debug
        then
          begin
            let owner = open_in whosegot in
            let lockee = input_line owner in
            close_in owner;
            Printf.eprintf "%s[%d]: lock on %s held by %s\n%!"
              Sys.argv.(0) (Unix.getpid ()) pathname lockee
            end;
            Unix.sleep !check;
            if naptime > 0 && Unix.time () > start +. float naptime
            then raise Exit
          done
        with Exit -> ()
      end;
    end;

  let owner =
    try
      open_out_gen [Open_wronly; Open_creat; Open_excl] 0o666 whosegot
    with Sys_error e ->
      raise (Error ("can't create " ^ e)) in
    Printf.fprintf owner "%s[%d] on %s\n"
      Sys.argv.(0) (Unix.getpid ()) (Unix.gethostname ());
    close_out owner;
    locked_files := StringSet.add pathname !locked_files

```

```

(* free the locked file *)
let nunflock pathname =
  let lockname = name2lock pathname in
  let whosegot = Filename.concat lockname "owner" in
  Unix.unlink whosegot;
  if !debug then Printf.eprintf "releasing lock on %s\n%!" lockname;
  locked_files := StringSet.remove pathname !locked_files;
  Unix.rmdir lockname

(* anything forgotten? *)
let () =
  at_exit
    (fun () ->
      StringSet.iter
        (fun pathname ->
          let lockname = name2lock pathname in
          let whosegot = Filename.concat lockname "owner" in
          Printf.eprintf "releasing forgotten %s\n%!" lockname;
          Unix.unlink whosegot;
          Unix.rmdir lockname)
        !locked_files)
end

```

## 7.22 Program: lockarea

```

(* The "fcntl" system call is not available in the OCaml standard library.
   You would have to drop down to C in order to lock regions of a file as
   described in the original Perl recipe. *)

```

## 8 File Contents

```
let () =
  try
    while true do
      let line = input_line datafile in
      let size = String.length line in
      Printf.printf "%d\n" size (* output size of line *)
    done
  with End_of_file -> ()

(*-----*)

let line_stream_of_channel channel =
  Stream.from
    (fun _ -> try Some (input_line channel) with End_of_file -> None)

let output_size line =
  Printf.printf "%d\n" (String.length line) (* output size of line *)

let () =
  Stream.iter output_size (line_stream_of_channel datafile)

(*-----*)

let lines =
  let xs = ref [] in
  Stream.iter
    (fun x -> xs := x :: !xs)
    (line_stream_of_channel datafile);
  List.rev !xs

(*-----*)

let slurp_channel channel =
  let buffer_size = 4096 in
  let buffer = Buffer.create buffer_size in
  let string = String.create buffer_size in
  let chars_read = ref 1 in
  while !chars_read <> 0 do
    chars_read := input channel string 0 buffer_size;
    Buffer.add_substring buffer string 0 !chars_read
  done;
  Buffer.contents buffer

let slurp_file filename =
  let channel = open_in_bin filename in
  let result =
    try slurp_channel channel
    with e -> close_in channel; raise e in
  close_in channel;
  result
```

```

let whole_file = slurp_file filename

(*-----*)

let () =
  (* Onetwothree *)
  List.iter (output_string handle) ["One"; "two"; "three"];

  (* Sent to default output handle *)
  print_string "Baa baa black sheep\n"

(*-----*)

let buffer = String.make 4096 '\000'
let rv = input handle buffer 0 4096
(* rv is the number of bytes read, *)
(* buffer holds the data read *)

(*-----*)

#load "unix.cma";;
let () =
  Unix.ftruncate descr length;
  Unix.truncate (Printf.sprintf "/tmp/%d.pid" (Unix.getpid ())) length

(*-----*)

let () =
  let pos = pos_in datafile in
  Printf.printf "I'm %d bytes from the start of datafile.\n" pos

(*-----*)

let () =
  seek_in in_channel pos;
  seek_out out_channel pos

#load "unix.cma";;
let () =
  Unix.lseek descr 0 Unix.SEEK_END; (* seek to the end *)
  Unix.lseek descr pos Unix.SEEK_SET; (* seek to pos *)
  Unix.lseek descr (-20) Unix.SEEK_CUR; (* seek back 20 bytes *)

(*-----*)

#load "unix.cma";;
let () =
  let written =
    Unix.write datafile mystring 0 (String.length mystring) in
  let read =
    Unix.read datafile mystring 5 256 in
  if read <> 256 then Printf.printf "only read %d bytes, not 256\n" read

```

```
(*-----*)

#load "unix.cma";;
let () =
  (* don't change position *)
  let pos = Unix.lseek handle 0 Unix.SEEK_CUR in
  (* ... *)
  ()
```

## 8.1 Reading Lines with Continuation Characters

```
let () =
  let buffer = Buffer.create 16 in
  let rec loop () =
    let line = input_line chan in
    if line <> "" && line.[String.length line - 1] = '\\\
'
    then (Buffer.add_string
          buffer (String.sub line 0 (String.length line - 1)));
         loop ()
    else Buffer.add_string buffer line;
         let line = Buffer.contents buffer in
         Buffer.clear buffer;
         (* process full record in line here *)
         loop () in
  try loop () with End_of_file -> ()
```

## 8.2 Counting Lines (or Paragraphs or Records) in a File

```
#load "unix.cma";;

let () =
  let proc = Unix.open_process_in ("wc -l < " ^ file) in
  let count = int_of_string (input_line proc) in
  ignore (Unix.close_process_in proc);
  (* count now holds the number of lines read *)
  ()

(*-----*)

let () =
  let count = ref 0 in
  let chan = open_in file in
  (try
    while true do
      ignore (input_line chan);
      incr count
    done
  with End_of_file -> close_in chan);
  (* !count now holds the number of lines read *)
  ()

(*-----*)
```

```

#load "str.cma";;

let () =
  let delim = Str.regexp "[ \n\r\t]*$" in
  let count = ref 0 in
  let in_para = ref false in
  let chan = open_in file in
  (try
    while true do
      if Str.string_match delim (input_line chan) 0
      then in_para := false
      else begin
        if not !in_para then incr count;
        in_para := true
      end
    end
  done
  with End_of_file -> close_in chan);
(* !count now holds the number of paragraphs read *)
()

```

### 8.3 Processing Every Word in a File

```

let word_stream_of_channel channel =
  (* Thanks to Mac Mason for figuring this out. *)
  let buffer = (Scanf.Scanning.from_channel channel) in
  Stream.from
    (fun count ->
      try
        match Scanf.bscanf buffer " %s " (fun x -> x) with
        | "" -> None
        | s -> Some s
      with End_of_file ->
        None)

  (*-----*)

let () =
  Stream.iter
    (fun chunk ->
      (* do something with chunk *)
      ())
    (word_stream_of_channel stdin)

  (*-----*)

(* Make a word frequency count *)
let seen = Hashtbl.create 0
let () =
  Stream.iter
    (fun word ->
      Hashtbl.replace seen word
        (try Hashtbl.find seen word + 1
         with Not_found -> 1))

```

```

(word_stream_of_channel stdin)

(* output hash in a descending numeric sort of its values *)
let () =
  let words = ref [] in
  Hashtbl.iter (fun word _ -> words := word :: !words) seen;
  List.iter
    (fun word ->
      Printf.printf "%5d %s\n" (Hashtbl.find seen word) word)
    (List.sort
      (fun a b -> compare (Hashtbl.find seen b) (Hashtbl.find seen a))
      !words)

(*-----*)

(* Line frequency count *)

let line_stream_of_channel channel =
  Stream.from
    (fun _ -> try Some (input_line channel) with End_of_file -> None)

let seen = Hashtbl.create 0
let () =
  Stream.iter
    (fun line ->
      Hashtbl.replace seen line
        (try Hashtbl.find seen line + 1
          with Not_found -> 1))
    (line_stream_of_channel stdin)

let () =
  let lines = ref [] in
  Hashtbl.iter (fun line _ -> lines := line :: !lines) seen;
  List.iter
    (fun line ->
      Printf.printf "%5d %s\n" (Hashtbl.find seen line) line)
    (List.sort
      (fun a b -> compare (Hashtbl.find seen b) (Hashtbl.find seen a))
      !lines)

```

## 8.4 Reading a File Backwards by Line or Paragraph

```

let lines = ref []
let () =
  try
    while true do
      lines := input_line chan :: !lines
    done
  with End_of_file -> ()
let () =
  List.iter
    (fun line ->
      (* do something with line *)

```

```
    ())
!lines
```

## 8.5 Trailing a Growing File

```
#load "unix.cma";;

let sometime = 1

let () =
  let chan = open_in file in
  while Sys.file_exists file do
    (try
      let line = input_line chan in
      (* ... *)
      ()
    with End_of_file ->
      Unix.sleep sometime)
  done;
  close_in chan
```

## 8.6 Picking a Random Line from a File

```
let () =
  Random.self_init ();
  let count = ref 1 in
  let line = ref "" in
  try
    while true do
      let next = input_line stdin in
      if Random.int !count < 1 then line := next;
      incr count
    done
  with End_of_file ->
    (* !line is the random line *)
    ()
```

## 8.7 Randomizing All Lines

```
(* assumes the fisher_yates_shuffle function from Chapter 4 *)
let shuffle list =
  let array = Array.of_list list in
  fisher_yates_shuffle array;
  Array.to_list array

let () =
  Random.self_init ();
  let lines = ref [] in
  (try
    while true do
      lines := (input_line input) :: !lines
    done
  with End_of_file -> ());
```

```

let reordered = shuffle !lines in
List.iter
  (fun line ->
    output_string output line;
    output_char output '\n')
  reordered

```

## 8.8 Reading a Particular Line in a File

```

(* Read lines until the desired line number is found. *)
let () =
  let line = ref "" in
  for i = 1 to desired_line_number do line := input_line handle done;
  print_endline !line

(* Read lines into an array. *)
let () =
  let lines = ref [] in
  (try while true do lines := input_line handle :: !lines done
  with End_of_file -> ());
  let lines = Array.of_list (List.rev !lines) in
  let line = lines.(desired_line_number) in
  print_endline line

(* Build an index file containing line offsets. *)
let build_index data_file index_file =
  set_binary_mode_out index_file true;
  let offset = ref 0 in
  try
    while true do
      ignore (input_line data_file);
      output_binary_int index_file !offset;
      offset := pos_in data_file
    done
  with End_of_file ->
    flush index_file

(* Read a line using the index file. *)
let line_with_index data_file index_file line_number =
  set_binary_mode_in index_file true;
  let size = 4 in
  let i_offset = size * (line_number - 1) in
  seek_in index_file i_offset;
  let d_offset = input_binary_int index_file in
  seek_in data_file d_offset;
  input_line data_file

(*-----*)

#!/usr/bin/ocaml
(* print_line-v1 - linear style *)

let () =

```

```

if Array.length Sys.argv <> 3
then (prerr_endline "usage: print_line FILENAME LINE_NUMBER"; exit 255);

let filename = Sys.argv.(1) in
let line_number = int_of_string Sys.argv.(2) in
let infile =
  try open_in filename
  with Sys_error e -> (prerr_endline e; exit 255) in
let line = ref "" in
begin
  try
    for i = 1 to line_number do line := input_line infile done
  with End_of_file ->
    Printf.eprintf "Didn't find line %d in %s\n" line_number filename;
    exit 255
end;
print_endline !line

(*-----*)

#!/usr/bin/ocaml
(* print_line-v2 - index style *)
#load "unix.cma";;
(* build_index and line_with_index from above *)
let () =
  if Array.length Sys.argv <> 3
  then (prerr_endline "usage: print_line FILENAME LINE_NUMBER"; exit 255);

  let filename = Sys.argv.(1) in
  let line_number = int_of_string Sys.argv.(2) in
  let orig =
    try open_in filename
    with Sys_error e -> (prerr_endline e; exit 255) in

  (* open the index and build it if necessary *)
  (* there's a race condition here: two copies of this *)
  (* program can notice there's no index for the file and *)
  (* try to build one. This would be easily solved with *)
  (* locking *)
  let indexname = filename ^ ".index" in
  let idx = Unix.openfile indexname [Unix.O_CREAT; Unix.O_RDWR] 0o666 in
  build_index orig (Unix.out_channel_of_descr idx);

  let line =
    try
      line_with_index orig (Unix.in_channel_of_descr idx) line_number
    with End_of_file ->
      Printf.eprintf "Didn't find line %d in %s\n" line_number filename;
      exit 255 in
  print_endline line

```

## 8.9 Processing Variable-Length Text Fields

```
(* given "record" with field separated by "pattern",
   extract "fields". *)
#load "str.cma";;
let regexp = Str.regexp pattern
let fields = Str.split_delim regexp record

(* same as above using PCRE library, available at:
   http://www.ocaml.info/home/ocaml_sources.html#pcre-ocaml *)
#directory "+pcre";;
#load "pcre.cma";;
let fields = Pcre.split ~pat:pattern record

(*-----*)

# Str.full_split (Str.regexp "[+-]") "3+5-2";;
- : Str.split_result list =
[Str.Text "3"; Str.Delim "+"; Str.Text "5"; Str.Delim "-"; Str.Text "2"]

# Pcre.split ~pat:"([+-])" "3+5-2";;
- : string list = ["3"; "+"; "5"; "-"; "2"]

(*-----*)

let fields = Str.split_delim (Str.regexp ":") record
let fields = Str.split_delim (Str.regexp "[ \n\r\t]+") record
let fields = Str.split_delim (Str.regexp " ") record

let fields = Pcre.split ~pat:":" record
let fields = Pcre.split ~pat:"\\s+" record
let fields = Pcre.split ~pat:" " record
```

## 8.10 Removing the Last Line of a File

```
#load "unix.cma";;

let () =
  let descr = Unix.openfile file [Unix.O_RDWR] 0o666 in
  let in_channel = Unix.in_channel_of_descr descr in
  let position = ref 0 in
  let last_position = ref 0 in
  begin
    try
      while true do
        ignore (input_line in_channel);
        last_position := !position;
        position := pos_in in_channel;
      done
    with End_of_file -> ()
  end;
  Unix.ftruncate descr !last_position;
  Unix.close descr
```

## 8.11 Processing Binary Files

```
set_binary_mode_in in_channel true
set_binary_mode_out out_channel true

(*-----*)

let () =
  let gifname = "picture.gif" in
  let gif = open_in gifname in
  set_binary_mode_in gif true;
  (* now DOS won't mangle binary input from "gif" *)
  set_binary_mode_out stdout true;
  (* now DOS won't mangle binary output to "stdout" *)
  let buff = String.make 8192 '\000' in
  let len = ref (-1) in
  while !len <> 0 do
    len := input gif buff 0 8192;
    output stdout buff 0 !len
  done
```

## 8.12 Using Random-Access I/O

```
let () =
  let address = recsize * recno in
  seek_in fh address;
  really_input fh buffer 0 recsize

(*-----*)

let () =
  let address = recsize * (recno - 1) in
  (* ... *)
  ()
```

## 8.13 Updating a Random-Access File

```
let () =
  let address = recsize * recno in
  seek_in in_channel address;
  let buffer = String.create recsize in
  really_input in_channel buffer 0 recsize;
  close_in in_channel;
  (* update fields, then *)
  seek_out out_channel address;
  output_string out_channel buffer;
  close_out out_channel

(*-----*)

#!/usr/bin/ocaml
(* weekearly -- set someone's login date back a week *)
#load "unix.cma";;
```

```

let sizeof = 4 + 12 + 16
let user =
  if Array.length Sys.argv > 1
  then Sys.argv.(1)
  else (try Sys.getenv "USER"
        with Not_found -> Sys.getenv "LOGNAME")

let address = (Unix.getpwnam user).Unix.pw_uid * sizeof

let () =
  let lastlog = open_in "/var/log/lastlog" in
  seek_in lastlog address;
  let line = String.make 12 ' ' in
  let host = String.make 16 ' ' in
  let time = input_binary_int lastlog in
  really_input lastlog line 0 12;
  really_input lastlog host 0 16;
  let buffer = String.create sizeof in
  really_input lastlog buffer 0 sizeof;
  close_in lastlog;

  let time = time - 24 * 7 * 60 * 60 in (* back-date a week *)

  let lastlog = open_out_gen [Open_wronly] 0o666 "/var/log/lastlog" in
  seek_out lastlog address;
  output_binary_int lastlog time;
  close_out lastlog

```

## 8.14 Reading a String from a Binary File

```

let () =
  let in_channel = open_in_bin file in
  seek_in in_channel addr;
  let buffer = Buffer.create 0 in
  let ch = ref (input_char in_channel) in
  while !ch <> '\000' do
    Buffer.add_char buffer !ch;
    ch := input_char in_channel;
  done;
  close_in in_channel;
  let string = Buffer.contents buffer in
  print_endline string

(*-----*)

(* bgets - get a string from an address in a binary file *)
open Printf

let file, addr =
  match Array.to_list Sys.argv with
  | _ :: file :: addr when List.length addr > 0 -> file, addr
  | _ -> eprintf "usage: %s file addr ...\n" Sys.argv.(0); exit 0

```

```

let () =
  let in_channel = open_in_bin file in
  List.iter
    (fun addr ->
      let addr = int_of_string addr in
      seek_in in_channel addr;
      let buffer = Buffer.create 0 in
      let ch = ref (input_char in_channel) in
      while !ch <> '\000' do
        Buffer.add_char buffer !ch;
        ch := input_char in_channel;
      done;
      printf "%#x %#o %d \"%s\"\n"
        addr addr addr (Buffer.contents buffer))
    addrs;
  close_in in_channel

(*-----*)

(* strings - pull strings out of a binary file *)
#load "str.cma";;

let find_strings =
  let pat = "[\040-\176\r\n\t ]" in
  let regexp = Str.regexp (pat ^ pat ^ pat ^ pat ^ "+") in
  fun f input ->
    List.iter
      (function Str.Delim string -> f string | _ -> ())
      (Str.full_split regexp input)

let file =
  try Sys.argv.(1)
  with Invalid_argument _ ->
    Printf.eprintf "usage: %s file\n" Sys.argv.(0);
    exit 0

let () =
  let in_channel = open_in_bin file in
  try
    while true do
      let buffer = Buffer.create 0 in
      let ch = ref (input_char in_channel) in
      while !ch <> '\000' do
        Buffer.add_char buffer !ch;
        ch := input_char in_channel;
      done;
      find_strings print_endline (Buffer.contents buffer)
    done
  with End_of_file ->
    close_in in_channel

```

## 8.15 Reading Fixed-Length Records

```
(* Using the Bitstring library by Richard W.M. Jones.
   http://code.google.com/p/bitstring/ *)
let () =
  try
    while true do
      let bitstring = Bitstring.bitstring_of_chan_max file recordsize in
      let fields = unpack bitstring in
      (* ... *)
      ()
    done
  with Match_failure _ -> ()

(*-----*)

(* Layout based on /usr/include/bits/utmp.h for a Linux system. *)
let recordsize = 384
let unpack bits =
  bitmatch bits with
  | { ut_type : 16 : littleendian;
      _ : 16; (* padding *)
      ut_pid : 32 : littleendian;
      ut_line : 256 : string;
      ut_id : 32 : littleendian;
      ut_user : 256 : string;
      ut_host : 2048 : string;
      ut_exit : 32 : littleendian;
      ut_session : 32 : littleendian;
      ut_tv_sec : 32 : littleendian;
      ut_tv_usec : 32 : littleendian;
      ut_addr_v6 : 128 : string } ->
    (ut_type, ut_pid, ut_line, ut_id, ut_user, ut_host,
     ut_exit, ut_session, ut_tv_sec, ut_tv_usec, ut_addr_v6)
```

## 8.16 Reading Configuration Files

```
#load "str.cma";;

let user_preferences = Hashtbl.create 0

let () =
  let comments = Str.regexp "#.*" in
  let leading_white = Str.regexp "^[ \t]+" in
  let trailing_white = Str.regexp "[ \t]+$" in
  let equals_delim = Str.regexp "[ \t]*=[ \t]*" in
  Stream.iter
    (fun s ->
      let s = Str.replace_first comments "" s in
      let s = Str.replace_first leading_white "" s in
      let s = Str.replace_first trailing_white "" s in
      (* anything left? *)
      if String.length s > 0 then
```

```

        match Str.bounded_split_delim equals_delim s 2 with
        | [var; value] -> Hashtbl.replace user_preferences var value
        | _ -> failwith s
    (* defined in this chapter's introduction *)
    (line_stream_of_channel config)

(*-----*)

(* load variables from ocaml source - toplevel scripts only *)
#use ".progrc";;

```

## 8.17 Testing a File for Trustworthiness

```

#load "unix.cma";;

let () =
  try
    let {Unix.st_dev = dev;
        st_ino = ino;
        st_kind = kind;
        st_perm = perm;
        st_nlink = nlink;
        st_uid = uid;
        st_gid = gid;
        st_rdev = rdev;
        st_size = size;
        st_atime = atime;
        st_mtime = mtime;
        st_ctime = ctime} = Unix.stat filename in
      (* ... *)
      ()
  with Unix.Unix_error (e, _, _) ->
    Printf.eprintf "no %s: %s\n" filename (Unix.error_message e);
    exit 0

(*-----*)

let () =
  let info =
    try Unix.stat filename
    with Unix.Unix_error (e, _, _) ->
      Printf.eprintf "no %s: %s\n" filename (Unix.error_message e);
      exit 0 in
    if info.Unix.st_uid = 0
    then Printf.printf "Superuser owns %s\n" filename;
    if info.Unix.st_atime > info.Unix.st_mtime
    then Printf.printf "%s has been read since it was written.\n" filename

(*-----*)

let is_safe path =
  let info = Unix.stat path in
  (* owner neither superuser nor me *)

```

```

(* the real uid can be retrieved with Unix.getuid () *)
if (info.Unix.st_uid <> 0) && (info.Unix.st_uid <> Unix.getuid ())
then false
else
  (* check whether the group or other can write file. *)
  (* use 0o066 to detect either reading or writing *)
  if info.Unix.st_perm land 0o022 = 0
  then true (* no one else can write this *)
  else if info.Unix.st_kind <> Unix.S_DIR
  then false (* non-directories aren't safe *)
  else if info.Unix.st_perm land 0o1000 <> 0
  then true (* but directories with the sticky bit (0o1000) are *)
  else false

(*-----*)

let is_very_safe path =
  let rec loop path parent =
    if not (is_safe path)
    then false
    else if path <> parent
    then loop parent (Filename.dirname parent)
    else true in
  loop path (Filename.dirname path)

```

## 8.18 Program: tailwtmp

```

(*pp camlp4o -I /path/to/bitstring bitstring.cma pa_bitstring.cmo *)

(* tailwtmp - watch for logins and logouts; *)
(* uses linux utmp structure, from utmp(5) *)

let days = [| "Sun"; "Mon"; "Tue"; "Wed"; "Thu"; "Fri"; "Sat" |]
let months = [| "Jan"; "Feb"; "Mar"; "Apr"; "May"; "Jun";
                "Jul"; "Aug"; "Sep"; "Oct"; "Nov"; "Dec" |]

let string_of_tm tm =
  Printf.sprintf "%s %s %2d %02d:%02d:%02d %04d"
    days.(tm.Unix.tm_wday)
    months.(tm.Unix.tm_mon)
    tm.Unix.tm_mday
    tm.Unix.tm_hour
    tm.Unix.tm_min
    tm.Unix.tm_sec
    (tm.Unix.tm_year + 1900)

let trim_asciiz s =
  try String.sub s 0 (String.index s '\000')
  with Not_found -> s

let () =
  let sizeof = 384 in
  let wtmp = open_in "/var/log/wtmp" in

```

```

seek_in wtmp (in_channel_length wtmp);
while true do
  let buffer = Bitstring.bitstring_of_chan_max wtmp sizeof in
  (bitmatch buffer with
   | { ut_type : 16 : littleendian;
       _ : 16; (* padding *)
       ut_pid : 32 : littleendian;
       ut_line : 256 : string;
       ut_id : 32 : littleendian;
       ut_user : 256 : string;
       ut_host : 2048 : string;
       ut_exit : 32 : littleendian;
       ut_session : 32 : littleendian;
       ut_tv_sec : 32 : littleendian;
       ut_tv_usec : 32 : littleendian;
       ut_addr_v6 : 128 : string } ->
    Printf.printf "%1d %-8s %-12s %10ld %-24s %-16s %5ld %-32s\n%!"
      ut_type (trim_ascii ut_user) (trim_ascii ut_line) ut_id
      (string_of_tm (Unix.localtime (Int32.to_float ut_tv_sec)))
      (trim_ascii ut_host) ut_pid (Digest.to_hex ut_addr_v6)
   | { _ } -> ());
  if pos_in wtmp = in_channel_length wtmp
  then Unix.sleep 1
done

```

## 8.19 Program: tctee

```

#!/usr/bin/ocaml
(* tctee - clone that groks process tees *)
#load "unix.cma";;

let ignore_ints = ref false
let append      = ref false
let unbuffer    = ref false
let nostdout    = ref false
let names       = ref []

let () =
  Arg.parse
  [
    "-a", Arg.Set append,      "Append to output files";
    "-i", Arg.Set ignore_ints, "Ignore interrupts";
    "-u", Arg.Set unbuffer,    "Unbuffered output";
    "-n", Arg.Set nostdout,    "No standard output";
  ]
  (fun name -> names := name :: !names)
  (Printf.sprintf "Usage: %s [-a] [-i] [-u] [-n] [filenames] ..."
   Sys.argv.(0));
  names := List.rev !names

let fhs = Hashtbl.create 0
let status = ref 0

```

```

let () =
  if not !nostdout then
    (* always go to stdout *)
    Hashtbl.replace fhs stdout "standard output";

  if !ignore_ints
  then
    List.iter
      (fun signal -> Sys.set_signal signal Sys.Signal_ignore)
      [Sys.sigint; Sys.sigterm; Sys.sighup; Sys.sigquit];

  List.iter
    (fun name ->
      if name.[0] = '|'
      then
        Hashtbl.replace fhs
          (Unix.open_process_out
            (String.sub name 1 (String.length name - 1)))
            name
        else
          begin
            let mode =
              if !append
              then [Open_wronly; Open_creat; Open_append]
              else [Open_wronly; Open_creat; Open_trunc] in
            try Hashtbl.replace fhs (open_out_gen mode 0o666 name) name
            with Sys_error e ->
              Printf.eprintf "%s: couldn't open %s: %s\n%!"
                Sys.argv.(0) name e;
              incr status
          end)
      !names;

  begin
    try
      while true do
        let line = input_line stdin in
        Hashtbl.iter
          (fun fh name ->
            try
              output_string fh line;
              output_string fh "\n";
              if !unbuffer then flush fh
            with Sys_error e ->
              Printf.eprintf "%s: couldn't write to %s: %s\n%!"
                Sys.argv.(0) name e;
              incr status)
            fhs
          done
        with End_of_file -> ()
    end;

  Hashtbl.iter

```

```

(fun fh name ->
  let close =
    if name.[0] = '|'
    then fun p -> ignore (Unix.close_process_out p)
    else close_out in
  try close fh
  with Sys_error e ->
    Printf.eprintf "%s: couldn't close %s: %s\n%!"
      Sys.argv.(0) name e;
    incr status)
fhs;

exit !status

```

## 8.20 Program: laston

```

(* laston - find out when a given user last logged on *)

#load "str.cma";;
#load "unix.cma";;

open Printf
open Unix

let lastlog = open_in "/var/log/lastlog"
let sizeof = 4 + 12 + 16
let line = String.make 12 ' '
let host = String.make 16 ' '

let days = [| "Sun"; "Mon"; "Tue"; "Wed"; "Thu"; "Fri"; "Sat" |]
let months = [| "Jan"; "Feb"; "Mar"; "Apr"; "May"; "Jun";
  "Jul"; "Aug"; "Sep"; "Oct"; "Nov"; "Dec" |]

let format_time time =
  let tm = localtime time in
  sprintf "%s %s %2d %02d:%02d:%02d %04d"
    days.(tm.tm_wday)
    months.(tm.tm_mon)
    tm.tm_mday
    tm.tm_hour
    tm.tm_min
    tm.tm_sec
    (tm.tm_year + 1900)

let trim_asciiz s =
  try String.sub s 0 (String.index s '\000')
  with Not_found -> s

let () =
  Array.iter
    (fun user ->
      try
        let u =

```

```

    try getpwuid (int_of_string user)
    with Failure _ -> getpwnam user in
seek_in lastlog (u.pw_uid * sizeof);
let time = input_binary_int lastlog in
really_input lastlog line 0 12;
really_input lastlog host 0 16;
let line = trim_asciiiz line in
let host = trim_asciiiz host in
printf "%-8s UID %5d %s%s%s\n"
    u.pw_name
    u.pw_uid
    (if time <> 0
     then format_time (float_of_int time)
     else "never logged in")
    (if line <> "" then " on " ^ line else "")
    (if host <> "" then " from " ^ host else "")
with Not_found ->
    printf "no such uid %s\n" user)
(Array.sub Sys.argv 1 (Array.length Sys.argv - 1))

```

## 9 Directories

open Unix

```
(* handle_unix_error generates a nice error message and exits *)
let entry = handle_unix_error stat "/usr/bin/vi"
let entry = handle_unix_error stat "/usr/bin/"
let entry = handle_unix_error fstat filedescr

(* without handle_unix_error an exception is raised for errors *)
let inode = stat "/usr/bin/vi"
let ctime = inode.st_ctime
let size = inode.st_size

(* don't know any equivalent in ocaml *)
(* maybe one could use file(1) (to know if it is an ASCII text file) *)
let dirhandle = handle_unix_error opendir "/usr/bin" in
begin
  try
    while true do
      let file = readdir dirhandle in
        Printf.printf "Inside /usr/bin is something called %s\n" file
      done
    with
      | End_of_file -> ()
end;
closedir dirhandle;;
```

### 9.1 Getting and Setting Timestamps

```
let (readtime, writetime) =
  let inode = stat filename in
    (inode.st_atime, inode.st_mtime);;

utimes filename newreadtime newwritetime;;

(*-----*)

let second_per_day = 60. *. 60. *. 24. in
let (atime, mtime) =
  let inode = stat filename in
    (inode.st_atime, inode.st_mtime) in
let newreadtime = atime -. 7. *. second_per_day
and newwritetime = mtime -. 7. *. second_per_day in
try
  utimes filename newreadtime newwritetime
with
  | Unix_error (er,_,_) ->
    Printf.eprintf
      "couldn't backdate %s by a week w/ utime: %s\n"
      filename (error_message er);;

(*-----*)
```

```

let mtime = (stat file).st_mtime in
utimes file (time ()) mtime ;;

(*-----*)

(* compile with ocamlc unix.cma uvi.ml -o uvi *)
open Unix

let main () =
  if (Array.length Sys.argv <> 2)
  then
    Printf.eprintf "Usage: uvi filename\n";
    let filename = Sys.argv.(1) in
    let atime, mtime =
      let st = stat filename in
      (st.st_atime, st.st_mtime) in
    let editor =
      begin
        try
          Sys.getenv "editor"
        with
          | Not_found -> "vi"
        end in
      Sys.command (Printf.sprintf "%s %s" editor filename);
      utimes filename atime mtime in
    main ();;

(*-----*)

```

## 9.2 Deleting a File

```

unlink filename;;                                (* use unix library *)
Sys.remove filename;;                            (* in the standard library *)

let error_flag = ref(None) in
let local_unlink filename =
  try
    unlink filename
  with
    | Unix_error (er,_,_) ->
      error_flag := (Some er) in
List.iter local_unlink filenames;
match !error_flag with
| Some er ->
  Printf.eprintf "Couldn't unlink all of";
  List.iter (Printf.eprintf " %s") filenames;
  Printf.eprintf ": %s\n" (error_message er)
| None ();;

(*-----*)

let error_flag = ref(0) in

```

```

let local_unlink count filename =
  try
    unlink filename;
    count + 1
  with
    | Unix_error (er,_,_) ->
      count in
let count = (List.fold_left local_unlink filenames 0)
and len = List.length filenames in
if count <> len
then
  Printf.eprintf "Could only delete %i of %i file\n" count len;;

(*-----*)

```

### 9.3 Copying or Moving a File

```

(*-----*)

(* Note : this doesn't use the unix library, only the standard one *)

let copy oldfile newfile =
  let infile = open_in oldfile
  and outfile = open_out newfile
  and blksize = 16384 in
  let buf = String.create blksize in
  let rec real_copy () =
    let byte_read = input infile buf 0 blksize in
    if byte_read <> 0 then
      begin
        (* Handle partial write : nothing to do *)
        output outfile buf 0 byte_read;
        real_copy ()
      end in
  real_copy ();
  close_in infile;
  close_out outfile;;

(*-----*)
Sys.command ("cp " ^ oldfile ^ " " ^ newfile) (* Unix *)
Sys.command (String.concat " " ["copy";oldfile;newfile]) (* Dos *)

(*-----*)

Unix.copy "datafile.dat" "datafile.bak";;

Sys.rename "datafile.dat" "datafile.bak";;

(*-----*)

```

### 9.4 Recognizing Two Names for the Same File

```

#load "unix.cma";;

```

```

(* Count the number of times a (dev, ino) pair is seen. *)
let seen = Hashtbl.create 0
let do_my_thing filename =
  let {Unix.st_dev=dev; st_ino=ino} = Unix.stat filename in
  Hashtbl.replace seen (dev, ino)
    (try Hashtbl.find seen (dev, ino) + 1
     with Not_found -> 1);
  if Hashtbl.find seen (dev, ino) = 1
  then
    begin
      (* do something with filename because we haven't
       seen it before. *)
    end

(*-----*)

(* Maintain a list of files for each (dev, ino) pair. *)
let seen = Hashtbl.create 0
let () =
  List.iter
    (fun filename ->
      let {Unix.st_dev=dev; st_ino=ino} = Unix.stat filename in
      Hashtbl.replace seen (dev, ino)
        (try filename :: Hashtbl.find seen (dev, ino)
         with Not_found -> [filename]))
    files
let () =
  Hashtbl.iter
    (fun (dev, ino) filenames ->
      Printf.printf "(%d, %d) => [%s]\n"
        dev ino (String.concat " " filenames))
    seen

```

## 9.5 Processing All Files in a Directory

```

(* Using Sys.readdir. *)
let () =
  Array.iter
    (fun file ->
      let path = Filename.concat dirname file in
      (* do something with path *)
      ())
    (Sys.readdir dirname)

(*-----*)

(* Using Unix.opendir, readdir, and closedir. Note that the "." and ".."
directories are included in the result unlike with Sys.readdir. *)
#load "unix.cma";;

let () =
  let dir =

```

```

try Unix.opendir dirname
with Unix.Unix_error (e, _, _) ->
  Printf.eprintf "can't opendir %s: %s\n"
    dirname (Unix.error_message e);
  exit 255 in
try
  while true do
    let file = Unix.readdir dir in
    let path = Filename.concat dirname file in
    (* do something with path *)
    ()
  done
with End_of_file ->
  Unix.closedir dir

(*-----*)

(* Get a list of full paths to plain files. *)
let plainfiles dir =
  List.filter
    (fun path ->
      match Unix.lstat path with
      | {Unix.st_kind=Unix.S_REG} -> true
      | _ -> false)
    (List.map
      (Filename.concat dir)
      (Array.to_list (Sys.readdir dir)))

```

## 9.6 Globbing, or Getting a List of Filenames Matching a Pattern

```

(* See recipe 6.9 for a more powerful globber. *)
#load "str.cma";;

(* OCaml does not come with a globbing function. As a workaround, the
  following function builds a regular expression from a glob pattern.
  Only the '*' and '?' wildcards are recognized. *)
let regexp_of_glob pat =
  Str.regexp
    (Printf.sprintf "%s$"
      (String.concat ""
        (List.map
          (function
            | Str.Text s -> Str.quote s
            | Str.Delim "*" -> ".*"
            | Str.Delim "?" -> "."
            | Str.Delim _ -> assert false)
          (Str.full_split (Str.regexp "[*?]") pat))))))

(* Now we can build a very basic globber. Only the filename part will
  be used in the glob pattern, so directory wildcards will break in
  this simple example. *)
let glob pat =
  let basedir = Filename.dirname pat in

```

```

let files = Sys.readdir basedir in
let regexp = regexp_of_glob (Filename.basename pat) in
List.map
  (Filename.concat basedir)
  (List.filter
    (fun file -> Str.string_match regexp file 0)
    (Array.to_list files))

(* Find all data files in the pleac directory. *)
let files = glob "pleac/*.data"

(*-----*)

(* Find and sort directories with numeric names. *)
let dirs =
  List.map snd (* extract pathnames *)
    (List.sort compare (* sort names numerically *)
      (List.filter (* path is a dir *)
        (fun (_, s) -> Sys.is_directory s)
        (List.map (* form (name, path) *)
          (fun s -> (int_of_string s, Filename.concat path s))
          (List.filter (* just numerics *)
            (fun s ->
              try ignore (int_of_string s); true
            with _ -> false)
            (Array.to_list
              (Sys.readdir path)))))) (* all files *)

```

## 9.7 Processing All Files in a Directory Recursively

```

let rec find_files f error root =
  Array.iter
    (fun filename ->
      let path = Filename.concat root filename in
      let is_dir =
        try Some (Sys.is_directory path)
        with e -> error root e; None in
      match is_dir with
      | Some true -> if f path then find_files f error path
      | Some false -> ignore (f path)
      | None -> ())
    (try Sys.readdir root with e -> error root e; [| |])

let process_file fn =
  (* Print the name of each directory and file found. *)
  Printf.printf "%s: %s\n"
    (if Sys.is_directory fn then "directory" else "file") fn;

  (* Prune directories named ".svn". *)
  not (Sys.is_directory fn && Filename.basename fn = ".svn")

let handle_error fn exc =
  Printf.eprintf "Error reading %s: %s\n" fn (Printexc.to_string exc)

```

```

let () =
  List.iter (find_files process_file handle_error) dirlist

(*-----*)

(* Add a trailing slash to the names of directories. *)
let () =
  List.iter
    (find_files
     (fun fn ->
       print_endline
         (if Sys.is_directory fn then (fn ^ "/") else fn);
       true)
     (fun _ _ -> ()))
    (match List.tl (Array.to_list Sys.argv) with
     | [] -> ["."]
     | dirs -> dirs)

(*-----*)

(* Sum the file sizes of a directory tree. *)
#load "unix.cma";;
let sum = ref 0
let () =
  List.iter
    (find_files
     (fun fn ->
       sum := !sum + (match Unix.stat fn
                       with {Unix.st_size=size} -> size);
       true)
     (fun _ _ -> ()))
    (match List.tl (Array.to_list Sys.argv) with
     | [] -> ["."]
     | dirs -> dirs);
  Printf.printf "%s contains %d bytes\n"
    (String.concat " " (List.tl (Array.to_list Sys.argv))) !sum

(*-----*)

(* Find the largest file in a directory tree. *)
#load "unix.cma";;
let saved_size = ref 0
let saved_name = ref ""
let () =
  List.iter
    (find_files
     (fun fn ->
       (match Unix.stat fn with
        | {Unix.st_size=size} ->
          if size > !saved_size
          then (saved_size := size; saved_name := fn));
       true)
     (fun _ _ -> ()))
    (match List.tl (Array.to_list Sys.argv) with
     | [] -> ["."]
     | dirs -> dirs);
  Printf.printf "%s contains %d bytes\n"
    (String.concat " " (List.tl (Array.to_list Sys.argv))) !sum

```

```

        (fun _ _ -> ()))
    (match List.tl (Array.to_list Sys.argv) with
    | [] -> ["."]
    | dirs -> dirs);
Printf.printf "Biggest file %s in %s is %d bytes long.\n"
!saved_name
(String.concat " " (List.tl (Array.to_list Sys.argv)))
!saved_size

(*-----*)

(* Find the youngest file or directory. *)
#load "unix.cma";;
let saved_age = ref 0.
let saved_name = ref ""
let () =
  List.iter
  (find_files
   (fun fn ->
    (match Unix.stat fn with
    | {Unix.st_mtime=age} ->
      if age > !saved_age
      then (saved_age := age; saved_name := fn));
    true)
   (fun _ _ -> ()))
  (match List.tl (Array.to_list Sys.argv) with
  | [] -> ["."]
  | dirs -> dirs);
  match Unix.localtime !saved_age with
  | {Unix.tm_year=year; tm_mon=month; tm_mday=day} ->
    Printf.printf "%04d-%02d-%02d %s\n"
      (year + 1900) (month + 1) day
      !saved_name

(*-----*)

(* fdirs - find all directories *)
let () =
  List.iter
  (find_files
   (fun fn ->
    if Sys.is_directory fn then print_endline fn;
    true)
   (fun _ _ -> ()))
  (match List.tl (Array.to_list Sys.argv) with
  | [] -> ["."]
  | dirs -> dirs)

```

## 9.8 Removing a Directory and Its Contents

```

(* rmtree - remove whole directory trees like rm -r *)
#load "unix.cma";;

```

```

let rec finddepth f roots =
  Array.iter
    (fun root ->
      (match Unix.lstat root with
        | {Unix.st_kind=Unix.S_DIR} ->
          finddepth f
            (Array.map (Filename.concat root) (Sys.readdir root))
        | _ -> ());
      f root)
    roots

let zap path =
  match Unix.lstat path with
  | {Unix.st_kind=Unix.S_DIR} ->
    Printf.printf "rmdir %s\n%!" path;
    Unix.rmdir path
  | _ ->
    Printf.printf "unlink %s\n%!" path;
    Unix.unlink path

let () =
  if Array.length Sys.argv < 2
  then (Printf.eprintf "usage: %s dir ..\n" Sys.argv.(0); exit 1);
  finddepth zap (Array.sub Sys.argv 1 (Array.length Sys.argv - 1))

```

## 9.9 Renaming Files

```

#load "unix.cma";;
let () = List.iter
  (fun file ->
    let newname = file in
      (* change newname *)
      Unix.rename file newname)
  names

(* rename - Larry's filename fixer *)
#load "unix.cma";;
#directory "+pcre";;
#load "pcre.cma";;
let () =
  match Array.to_list Sys.argv with
  | prog :: pat :: templ :: files ->
    let replace = Pcre.replace ~pat ~templ in
      List.iter
        (fun file ->
          let file' = replace file in
            Unix.rename file file')
          files
  | _ -> prerr_endline "Usage: rename pattern replacment [files]"

(*
% rename '\.orig$' '' *.orig
% rename '$' '.bad' *.f

```

```

% rename '([\^/]+)~$' '.#$1' /tmp/*~
% find /tmp -name '*~' -exec rename '([\^/]+)~$' '.#$1' {} \;
*)

```

## 9.10 Splitting a Filename into Its Component Parts

```

let splitext name =
  try
    let root = Filename.chop_extension name in
    let i = String.length root in
    let ext = String.sub name i (String.length name - i) in
    root, ext
  with Invalid_argument _ ->
    name, ""

let dir = Filename.dirname path
let file = Filename.basename path
let name, ext = splitext file

```

## 9.11 Program: symirror

```

#!/usr/bin/ocaml
(* symirror - build spectral forest of symlinks *)
#load "unix.cma";;

open Printf

let die msg = prerr_endline msg; exit 1

let () =
  if Array.length Sys.argv < 3
  then die (sprintf "usage: %s realdir mirrordir" Sys.argv.(0))

let srcdir, dstdir = Sys.argv.(1), Sys.argv.(2)
let cwd = Unix.getcwd ()

let fix_relative path =
  if Filename.is_relative path
  then Filename.concat cwd path
  else path

let is_dir dir =
  try Some (Sys.is_directory dir)
  with Sys_error _ -> None

let () =
  match (is_dir srcdir, is_dir dstdir) with
  | (None, _) | (Some false, _) ->
    die (sprintf "%s: %s is not a directory" Sys.argv.(0) srcdir)
  | (_, Some false) ->
    die (sprintf "%s: %s is not a directory" Sys.argv.(0) dstdir)
  | (_, None) ->
    Unix.mkdir dstdir 0o7777 (* be forgiving *)

```

```

    | (Some _, Some _) ->
      ()
    (* cool *)

(* fix relative paths *)
let srcdir, dstdir = fix_relative srcdir, fix_relative dstdir

let rec find f roots =
  Array.iter
    (fun root ->
      f root;
      match Unix.lstat root with
      | {Unix.st_kind=Unix.S_DIR} ->
          find f (Array.map
            (Filename.concat root)
            (Sys.readdir root))
      | _ -> ())
    roots

let wanted name =
  if name <> Filename.current_dir_name
  then
    let {Unix.st_dev=dev; st_ino=ino; st_kind=kind; st_perm=perm} =
      Unix.lstat name in
    (* preserve directory permissions *)
    let perm = perm land 0o7777 in
    (* correct name *)
    let name =
      if String.length name > 2 && String.sub name 0 2 = "./"
      then String.sub name 2 (String.length name - 2)
      else name in
    if kind = Unix.S_DIR
    then
      (* make a real directory *)
      Unix.mkdir (Filename.concat dstdir name) perm
    else
      (* shadow everything else *)
      Unix.symlink
        (Filename.concat srcdir name)
        (Filename.concat dstdir name)

let () =
  Unix.chdir srcdir;
  find wanted [|"."|]

```

## 9.12 Program: lst

```

#!/usr/bin/ocaml
(* lst - list sorted directory contents (depth first) *)
#load "unix.cma";;

open Unix
open Printf

```

```

let opt_m = ref false
let opt_u = ref false
let opt_c = ref false
let opt_s = ref false
let opt_r = ref false
let opt_i = ref false
let opt_l = ref false
let names = ref []

let () =
  Arg.parse
  [
    "-m", Arg.Set opt_m, "Use mtime (modify time) [DEFAULT]";
    "-u", Arg.Set opt_u, "Use atime (access time)";
    "-c", Arg.Set opt_c, "Use ctime (inode change time)";
    "-s", Arg.Set opt_s, "Use size for sorting";
    "-r", Arg.Set opt_r, "Reverse sort";
    "-i", Arg.Set opt_i, "Read pathnames from stdin";
    "-l", Arg.Set opt_l, "Long listing";
  ]
  (fun name -> names := name :: !names)
  (sprintf
    "Usage: %s [-m] [-u] [-c] [-s] [-r] [-i] [-l] [dirs ...]
or %s -i [-m] [-u] [-c] [-s] [-r] [-l] <filelist"
    Sys.argv.(0) Sys.argv.(0));
  names :=
    match !names with
    | [] when not !opt_i -> ["."]
    | names -> names

let die msg = prerr_endline msg; exit 1

let () =
  let int_of_bool = function true -> 1 | false -> 0 in
  if (int_of_bool !opt_c
    + int_of_bool !opt_u
    + int_of_bool !opt_s
    + int_of_bool !opt_m) > 1
  then die "can only sort on one time or size"

let idx = fun {st_mtime=t} -> t
let idx = if !opt_u then fun {st_atime=t} -> t else idx
let idx = if !opt_c then fun {st_ctime=t} -> t else idx
let idx = if !opt_s then fun {st_size=s} -> float s else idx
let time_idx = if !opt_s then fun {st_mtime=t} -> t else idx

let rec find f roots =
  Array.iter
  (fun root ->
    f root;
    match lstat root with
    | {st_kind=S_DIR} ->
      find f (Array.map

```

```

                                (Filename.concat root)
                                (Sys.readdir root))
    | _ -> ()
    roots

let time = Hashtbl.create 0
let stat = Hashtbl.create 0

(* get stat info on the file, saving the desired *)
(* sort criterion (mtime, atime, ctime, or size) *)
(* in the time hash indexed by filename.          *)
(* if they want a long list, we have to save the *)
(* entire stat structure in stat.                 *)
let wanted name =
  try
    let sb = Unix.stat name in
      Hashtbl.replace time name (idx sb);
      if !opt_l then Hashtbl.replace stat name sb
    with Unix_error _ -> ()

(* cache user number to name conversions *)
let user =
  let user = Hashtbl.create 0 in
  fun uid ->
    Hashtbl.replace user uid
      (try (getpwuid uid).pw_name
         with Not_found -> ("#" ^ string_of_int uid));
    Hashtbl.find user uid

(* cache group number to name conversions *)
let group =
  let group = Hashtbl.create 0 in
  fun gid ->
    Hashtbl.replace group gid
      (try (getgrgid gid).gr_name
         with Not_found -> ("#" ^ string_of_int gid));
    Hashtbl.find group gid

let days = [| "Sun"; "Mon"; "Tue"; "Wed"; "Thu"; "Fri"; "Sat" |]
let months = [| "Jan"; "Feb"; "Mar"; "Apr"; "May"; "Jun";
                "Jul"; "Aug"; "Sep"; "Oct"; "Nov"; "Dec" |]

let format_time time =
  let tm = localtime time in
  sprintf "%s %s %2d %02d:%02d:%02d %04d"
    days.(tm.tm_wday)
    months.(tm.tm_mon)
    tm.tm_mday
    tm.tm_hour
    tm.tm_min
    tm.tm_sec
    (tm.tm_year + 1900)

```

```

let () =
  if !opt_i
  then
    begin
      begin
        try
          while true do
            names := (input_line Pervasives.stdin) :: !names
          done
          with End_of_file -> ()
        end;
        List.iter wanted (List.rev !names)
      end
    else find wanted (Array.of_list (List.rev !names))

(* sort the files by their cached times, youngest first *)
let skeys =
  List.sort
    (fun a b -> compare (Hashtbl.find time b) (Hashtbl.find time a))
    (Hashtbl.fold (fun k v a -> k :: a) time [])

(* but flip the order if -r was supplied on command line *)
let skeys = if !opt_r then List.rev skeys else skeys

let () =
  List.iter
    (fun skey ->
      if !opt_l
      then
        let sb = Hashtbl.find stat skey in
        printf "%6d %04o %6d %8s %8s %8d %s %s\n"
          sb.st_ino
          (sb.st_perm land 0o7777)
          sb.st_nlink
          (user sb.st_uid)
          (group sb.st_gid)
          sb.st_size
          (format_time (time_idx sb))
          skey
        else
          print_endline skey)
    skeys

```

## 10 Subroutines

```
(* A function is bound to a variable (as with everything) with the let key-
word
*)

let hello () =
  incr greeted; (* global reference *)
  printf "hi there!\n";

(* Other forms for declaring a function are as follows *)

let hello =
  fun () ->
    incr greeted; (* global reference *)
    printf "hi there!\n";

let hello =
  function () ->
    incr greeted; (* global reference *)
    printf "hi there!\n";

(* The typical way of calling this function is *)

hello ();;
```

### 10.1 Accessing Subroutine Arguments

```
(* All values passed to a function must be named in the para-
meter list to the
* function *)

let hypotenuse side1 side2 =
  sqrt ((side1 ** 2.) +. (side2 ** 2.));

(* Note, however, that if the parameters are defined/sent as a tu-
ple then they
* can be accessed in one of two equivalent ways *)

let hypotenuse (side1,side2) =
  sqrt ((side1 ** 2.) +. (side2 ** 2.));

let hypotenuse sides =
  let side1,side2 = sides in
  sqrt ((side1 ** 2.) +. (side2 ** 2.));

(* In both of these cases, however, we must pass the arguments as a tu-
ple *)

print_float hypotenuse (3.,4.);

(* since most data structures are immutable, one gener-
ally does not need to copy
```

```

* the parameters into local variables *)

let nums = [1.4; 3.5; 6.7];;
let int_all l =
  List.map int_of_float l;;

(*
# let ints = int_all nums;;
val ints : int list = [1; 3; 6]

# nums;;
- : float list = [1.4; 3.5; 6.7]
*)

(* However, one needs to be careful when mutable data is passed in and
* operations that alter that data are used *)

let nums = [|1.4; 3.5; 6.7|];;
let int_all2 a =
  Array.iteri (fun i x -> a.(i) <- 10. *. x) a;
  a;;
let int_all3 a =
  Array.map int_of_float a;;

(*
# let a2 = int_all2 nums;;
val a2 : int array = [|1; 3; 6|]

# nums;;
- : float array = [|1.4; 3.5; 6.7|]

# let a3 = times10 nums;;
val a3 : float array = [|14.; 35.; 67.|]

# nums;;
- : float array = [|14.; 35.; 67.|]
*)

(* To write functions that change their caller's variables, those vari-
ables must
* be mutable structures, such as references *)
let nums = ref [1.4; 3.5; 6.7];;
let trunc_em l =
  l := List.map floor !l;
  !l;;

(*
# let n2 = trunc_em nums;;
val n2 : float list = [1.; 3.; 6.]

# !nums;;
- : float list = [1.; 3.; 6.]

```

\*)

## 10.2 Making Variables Private to a Function

```
(* to declare a variable local to a function, simply use let in-
side the function
* body *)
```

```
let somefunc () =
  let variable = ... in
  let another,anarray,ahash = ... in
  ... ;;

let check_x x =
  let y = "whatever" in
  run_check ();
  if condition then printf "got %s" x;;

let save_array arguments =
  global_list := arguments @ !global_list;;
```

## 10.3 Creating Persistent Private Variables

```
let mysub =
  let variable = ... in
  fun args -> ... ;;

(* To write a counter *)
let next_counter =
  let counter = ref 0 in
  fun () ->
    incr counter;
    !counter;;

let next_counter,prev_counter =
  let counter = ref 42 in
  (fun () -> incr counter; !counter),
  (fun () -> decr counter; !counter);;
```

## 10.4 Determining Current Function Name

(\* The names of functions are not available at runtime. However, using the camlp4 preprocessor, we can expose various pieces of static information.

The "macro" parser provides the current file and location as `__FILE__` and `__LOCATION__`, respectively. With a bit more customization, we can expose the current function name as well.

To do this, we'll make a copy of `camlp4/Camlp4Filters/Camlp4Profiler.ml` from the OCaml sources and rename it to `"Camlp4FuncNamer.ml"`. Then, we'll change the definition of `"decorate_this_expr"` to the following: \*)

```

(*-----*)

value decorate_this_expr e id =
  let _loc = Ast.loc_of_expr e in
  <:expr< let __FUNC__ = '$str:id$ in $$ >>;

(*-----*)

(* This has the effect of exposing the current function name as the
   string, __FUNC__, which we can use just like __FILE__. To build this
   syntax extension, use a command like the following: *)

ocamlc -c -pp camlp4rf -I /usr/lib/ocaml/3.10.2/camlp4 Camlp4FuncNamer.ml

(*-----*)

(* Now, we'll write a simple test program called "main.ml": *)

(* Comment out this line to silence log messages. *)
DEFINE DEBUG

(* Default function name if Camlp4FuncNamer doesn't provide one. *)
let __FUNC__ = "<toplevel>"

(* Log macro with Printf formatting. *)
DEFINE LOG =
  IFDEF DEBUG THEN
    Printf.kprintf
      (Printf.eprintf "%s[%s]: %s\n%!" __FUNC__ __FILE__)
  ELSE
    Printf.kprintf (fun _ -> ())
  END

(* An example named function. *)
let test_function () =
  let str = "Hello, world!" in
  let num = 42 in
  LOG "str=\"%s\", num=%d" str num;
  print_endline "test complete"

(* Some code to run at the toplevel. *)
let () =
  LOG "not in a function";
  test_function ()

(*-----*)

(* We can compile this program as follows: *)

ocamlc -pp "camlp4of Camlp4FuncNamer.cmo" \
  -I /usr/lib/ocaml/3.10.2/camlp4 \
  -o main main.ml

```

```
(* Running it, we get this output: *)

<toplevel>[main.ml]: not in a function
test_function[main.ml]: str="Hello, world!", num=42
test complete
```

## 10.5 Passing Arrays and Hashes by Reference

```
(* Because all OCaml variables represent pointers to their data, all func-
tion
* arguments are implicitly passed by reference *)

array_diff array1 array2;;

let a = [| 1; 2 |];;
let b = [| 5; 8 |];;
let add_vec_pair x y =
  Array.init (Array.length x) (fun i -> x.(i) + y.(i));;

(*
# let c = add_vec_pair a b;;
val c : int array = [|6; 10|]
*)
```

## 10.6 Detecting Return Context

```
(* OCaml's type safety doesn't allow this kind of shenanigans un-
less you bring
* union types into play --
but you still need to ensure that the return type of
* all three contexts is the same *)

type 'a lORs =
  List of 'a list
  | Scalar of 'a
  | Void of unit ;;

let mysub arg =
  match arg with
  | List l -> (* list context, do something with l *)
  | Scalar s -> (* scalar context, do something with s *)
  | Void _ -> (* void context, do something with nothing *);;

(* or equivalently *)
let mysub = function
  | List l -> (* list context, do something with l *)
  | Scalar s -> s (* scalar context, do something with s *)
  | Void _ -> (* void context, do something with nothing *);;

mysub (Void ());;          (* void context *)
mysub (Scalar arg);;      (* scalar context *)
mysub (List arg);;        (* list context *)
```

## 10.7 Passing by Named Parameter

```
(* To name the arguments of a function, use labels *)
let thefunc ~increment ~finish ~start =
  ... ;;

(* It can be called like *)
thefunc ~increment:"20s" ~start:"+5m" ~finish:"+30m";;

(* Note that you can use different names for the labels and vari-
ables, and if
* the application is total, the labels can be omitted *)
let divide ~numerator:x ~denominator:y =
  x / y;;

(*
# divide ~denominator:2 ~numerator:100;;
- : int = 50

# divide 20 4;;
- : int = 5
*)

(* If you want to provide default values, you need to use optional argu-
ments,
* but this requires at least one unlabelled argument *)

let fraction ?(y = 2) x =
  x / y;;

(*
fraction 30 ~y:3;;
- : int = 10

fraction 30;;
- : int = 15
*)
```

## 10.8 Skipping Selected Return Values

```
(* Use _, which matches any pattern and throws away the value it matches *)

let a,_,c = func ();;
let _,_,d = func ();;
```

## 10.9 Returning More Than One Array or Hash

```
(* Just stick all of the values in a tuple and return it *)
let somefunc () =
  let arr = ... in
  let hash = ... in
  ...
  (arr,hash);;
```

```
let a,h = somefunc ();;
```

## 10.10 Returning Failure

```
(* Use an appropriate exception *)
```

```
let failing_routine () =  
  ...  
  raise Failure "Bad things happened...";;  
  
try failing_routine () with  
  Failure s -> printf "failing_routine failed because: %s\n" s;;
```

## 10.11 Prototyping Functions

```
(* This is pretty much unnecessary due to OCaml's type inference --  
you will  
* know at compile time if you try to pass invalid arguments to a function *)
```

## 10.12 Handling Exceptions

```
(* To handle exceptions, which are thrown with the raise keyword, wrap the  
* possibly excep-  
tional call in a try ... with block. You only need to do this  
* where appropriate *)
```

```
let slurp_to_list filename =  
  (* Note, if filename does not exist in the current direc-  
  tory, it will raise a  
  * Sys_error exception *)  
  let ic = open_in filename and  
  l = ref [] in  
  let rec loop () =  
    let line = input_line ic in  
    l := line::!l;  
    loop () in  
  try loop () with End_of_file -> close_in ic; List.rev !l;;  
  
let lfind name l =  
  (* Note, if no elements in the list satisfy the predicate, List.find will  
  * raise the Not_found exception *)  
  List.find (fun x -> Str.string_match (Str.regexp ("$" ^ name)) x 0) l;;  
  
let findSmurfette =  
  try  
    print_endline (lfind "Smurfette" (slurp_to_list "smurfs"))  
  with  
    Sys_error s -> prerr_endline ("Dammit! - " ^ s)  
  | Not_found -> prerr_endline "Hmmm... Smurfette is not in smurfs";;
```

## 10.13 Saving Global Values

```
(* To do this in OCaml --
which doesn't like global state in the first place --
* you need to manually store the old value and replace it before exiting the
* block *)

let age = ref 18;;
if condition then
  (
    let org_age = !age in
    age := 23;
    func ();
    age := org_age
  );;

(* for local handles, just create a new channel inside your block *)
let get_motd () =
  let motd = open_in "/etc/motd" in
  let retval =
    ... in
  close_in motd;
  retval;;
```

## 10.14 Redefining a Function

```
(* If you want to redefine a function... go ahead. Functions are first class
* members in OCaml *)

let f x y =
  x + y;;

f 5 7;;
(* - : int = 12 *)

let f x y =
  x - y;;

f 5 7;;

(* - : int = -2 *)

(* to do it temporarily, either save to old value and then restore it, or just
* redefine it in the current block. The old value will be restored when you
* exit the scope of that block *)

let g = f
and f x y =
  x * y;;
```

```

f 5 7;;

(* - : int = 35 *)

let f = g;;

f 5 7;;

(* - : int = -2 *)

let g () =
  let f x y =
    x / y in
  f 5 7;;

g ();;

(* - : int = 0 *)

f 5 7;;

(* - : int = -2 *)

```

## 10.15 Trapping Undefined Function Calls with AUTOLOAD

```

(* Since OCaml is statically typed, any attempt to call an undefined
function will result in a compiler error. There is no way to capture
and handle this event at runtime. *)

```

## 10.16 Nesting Subroutines

```

(* Just define the inner function within the outer one *)
let outer x =
  let x = x + 35 in
  let inner () =
    x * 19 in
  x + inner ();;

```

## 10.17 Program: Sorting Your Mail

```

let slurp_to_string filename =
  let ic = open_in filename and
  buf = Buffer.create 4096 in
  let rec loop () =
    let line = input_line ic in
    Buffer.add_string buf line;
    Buffer.add_string buf "\n";
    loop () in
  try loop () with End_of_file -> close_in ic; Buffer.contents buf;;

(* Note: The following function does something slightly differ-
ent than the Perl

```

```

* version, as it returns a subject,message #,reference to the message tuple
* sorted by subject -
> message number instead of just a list of messages sorted
* by subject -> message number -- it's trivial to get just what the Perl
* version does from this... *)

let sortedMail fn =
  let msglist =
    (* I had to add this filtering step due to some wierd struc-
    ture in my mbox
    * file. go figure... *)
    List.filter (fun s -> String.sub s 0 5 = "From:")
      (List.map (fun x -> "From" ^ x)
        (Str.split (Str.regexp "^From") (slurp_to_string fn)))
    and counter = ref (-1) in
  (* let subjList = *)
  List.sort compare
    (List.map
      (fun s ->
        ignore (Str.search_forward
          (* Not positive this regex is equivalent to the Perl ver-
          sion, but it
          * seems to work --
          you can use the third party PCRE module if you
          * want to be positive *)
          (Str.regexp "^Sub-
          ject:[ \t]*(?:[Rr][Ee]:[ \t]*\)*\(.*\)") s 0);
          incr counter;
          (try (String.lowercase (Str.matched_group 2 s)) with Not_found -
          > ""),
          !counter,
          ref s)
        msglist));

List.iter (fun (_,_,rm) -> print_endline !rm) (sortedMail "mbox");;

(* To sort by using a hashtable *)

let keys h =
  let k = Hashtbl.fold (fun k v b -> k::b) h [] in
  (* filter out duplicates *)
  List.fold_left (fun b x -> if List.mem x b then b else x::b) [] k;;

let sortedMailByHash fn =
  let msglist =
    (* I had to add this filtering step due to some wierd struc-
    ture in my mbox
    * file. go figure... *)
    List.filter (fun s -> String.sub s 0 5 = "From:")
      (List.map (fun x -> "From" ^ x)
        (Str.split (Str.regexp "^From") (slurp_to_string fn)))
    and counter = ref (-1) in
  let h = Hashtbl.create (List.length msglist) in

```

```

(* let subjList = *)
(* List.sort compare *)
(List.iter
  (fun s ->
    ignore (Str.search_forward
      (* Not positive this regex is equivalent to the Perl ver-
      sion, but it
      * seems to work --
      you can use the third party PCRE module if you
      * want to be positive *)
      (Str.regexp "^Sub-
      ject:[ \t]*\(:?[Rr][Ee]:[ \t]*\)*\(.*\)") s 0);
    incr counter;
    let sub =
      try
        (String.lowercase (Str.matched_group 2 s))
      with Not_found -> "" in
      Hashtbl.add h sub s)
    msglist;
  List.flatten
    (List.map (fun x -> List.rev (Hashtbl.find_all h x))
      (List.sort (keys h))));;

List.iter (fun m -> print_endline m) (sortedMailByHash "mbox"));;

```

## 11 References and Records

```
(* Create a reference to an integer *)
let aref = ref 0

let () =
  (* Assign to aref's contents *)
  aref := 3;

  (* Print the value that the reference "aref" refers to *)
  Printf.printf "%d\n" !aref;

  (* Since references are just records with a single field, "contents",
     the following operations have the same effect as above *)
  aref.contents <- 3;
  Printf.printf "%d\n" aref.contents;

  (* Fast increment and decrement operations are available for int refs *)
  incr aref; Printf.printf "after incr: %d\n" !aref;
  decr aref; Printf.printf "after decr: %d\n" !aref

(* Create a type for "person" records *)
type person = { name : string;
                address : string;
                birthday : int;
                }

let () =
  (* Create a "person" record *)
  let nat = { name      = "Leonhard Euler";
             address   = "1729 Ramunjan Lane\nMathword, PI 31416";
             birthday  = 0x5bb5580;
             } in

  (* Display the person's name and address *)
  Printf.printf "\nname: %s\naddress: %s\n" nat.name nat.address;

  (* Same as above, using pattern-matching *)
  let {name=n; address=a} = nat in
  Printf.printf "\nname: %s\naddress: %s\n" n a
```

### 11.1 Taking References to Arrays

```
(* The following two sections use lists instead of arrays since
   list refs can be enlarged and copied easily. Also, arrays are
   mutable in OCaml, whereas lists are immutable. *)

(* Create a reference to a list *)
let lref      = ref list
let anon_list = ref [9; 7; 5; 3; 1]
let anon_copy = ref !anon_list

let () =
```

```

(* Add an item to the list *)
anon_list := 11 :: !anon_list;

(* Get the number of items from the list ref *)
let num_items = List.length !anon_list in

(* Print original data *)
print_endline (String.concat ", "
               (List.map (fun i -> string_of_int i) !anon_list));

(* Sort it *)
anon_list := List.sort compare !anon_list;

(* Print sorted data *)
print_endline (String.concat ", "
               (List.map (fun i -> string_of_int i) !anon_list));

```

## 11.2 Making Hashes of Arrays

```

(* Create a hash that maps strings to string lists *)
let (hash : (string, string list) Hashtbl.t) = Hashtbl.create 0

(* Define a function to add a string to the string list associated
   with a key in the hash creating the string list if necessary *)
let add hash key value =
  Hashtbl.replace hash key
    (try value :: Hashtbl.find hash key
     with Not_found -> [value])

let () =
  (* Populate the hash with some data *)
  add hash "fruit" "apple";
  add hash "fruit" "banana";
  add hash "wine" "merlot";
  add hash "cheese" "cheddar";
  add hash "cheese" "brie";
  add hash "cheese" "havarti";

  (* Iterate and print out the hash's contents *)
  Hashtbl.iter
    (fun key values ->
     Printf.printf "%s: %s\n" key
       (String.concat ", " values))
    hash

(* Hashtbl is somewhat unusual in that it allows multiple values for
   a given key. By using Hashtbl.add instead of Hashtbl.replace, and
   using strings as values instead of string lists, we can save some
   memory *)
let (hash : (string, string) Hashtbl.t) = Hashtbl.create 0
let () =
  Hashtbl.add hash "foo" "bar";
  Hashtbl.add hash "foo" "baz";

```

```

Hashtbl.add hash "goo" "arc";
Hashtbl.iter (Printf.printf "%s => %s\n") hash

```

### 11.3 Taking References to Hashes

```

(* Hashtbls are mutable, so creating a reference to a hash is usually
   not necessary; it creates an *additional* level of indirection. *)
let href = ref hash
let anon_hash = ref (Hashtbl.create 0)
let () =
  (* Have some fun with locally-defined operators *)
  let ( => ) = Hashtbl.replace !anon_hash in
  ( "key1" => "value1"; "key2" => "value2" )
let anon_hash_copy = ref (Hashtbl.copy !href)

```

### 11.4 Taking References to Functions

```

(* Create a reference to a function *)
let fref = ref func
let fref = ref (fun () -> (* ... *) ())

(* Call the referent function *)
let () = !fref ()

(* Create a reference to an association list with function values. *)
let commands = ref []
let () =
  let ( => ) name func = commands := (name, func) :: !commands in
  (
    "happy" => joy;
    "sad"   => sullen;
    "done"  => (fun () -> print_endline "See ya!"; exit 0);
    "mad"   => angry;
  )

let () =
  while true do
    print_string "How are you? ";
    let string = read_line () in
    try
      let command = List.assoc string !commands in
      command ()
    with Not_found ->
      Printf.printf "No such command: %s\n" string
  done

(* Use closures to generate functions that count. *)

let counter_maker () =
  let start = ref 0 in
  fun () ->
    let result = !start in
    incr start; result
    (* this is a closure *)
    (* lexical from enclosing scope *)

```

```

let counter1 = counter_maker ()
let counter2 = counter_maker ()

let () =
  for i = 0 to 4 do
    Printf.printf "%d\n" (counter1 ())
  done;
  Printf.printf "%d %d\n" (counter1 ()) (counter2 ())
  (*
    0
    1
    2
    3
    4
    5 0
  *)

(* Use closures to generate functions that keep track of time.
   Note that this example does not need references, since
   since functions are just ordinary values in OCaml. *)

#load "unix.cma";;

let timestamp () =
  let start_time = Unix.time () in
  fun () -> int_of_float (Unix.time () -. start_time)

let () =
  let early = timestamp () in
  Unix.sleep 20;
  let later = timestamp () in
  Unix.sleep 10;
  Printf.printf "It's been %d seconds since early.\n" (early ());
  Printf.printf "It's been %d seconds since later.\n" (later ());
  (*
    It's been 30 seconds since early.
    It's been 10 seconds since later.
  *)

```

## 11.5 Taking References to Scalars

```

(* Environments are immutable in OCaml; there is no way to get a
   reference to a value. If you need a mutable cell, use "ref" as
   described in the introduction. If you need to refer to values
   by name strings, use a Hashtbl.t or similar data structure. *)

```

## 11.6 Creating Arrays of Scalar References

```

(* Create a couple of integer references *)
let a = ref 0
let b = ref 0

```

```

(* Create an array of the references *)
let array_of_refs = [| a; b |]

let () =
  (* Set the value of an element *)
  array_of_refs.(1) := 12;          (* b := 12 *)

  (* Note that this is *not* the same as array mutation! If we were to do:
     array_of_refs.(1) <- ref 12
     (or drop the refs altogether) then we would no longer be aliasing "b".
  *)

  (* Get the value of an element *)
  Printf.printf "%d %d\n" !(array_of_refs.(1)) !b

let () =
  let (a, b, c, d) = (ref 1, ref 2, ref 3, ref 4) in (* initialize *)
  let array = [| a; b; c; d |] in                    (* refs to each value *)

  array.(2) := !(array.(2)) + 9;                    (* !c is now 12 *)

  let tmp = array.(Array.length array - 1) in
  tmp := !tmp * 5;                                  (* !d is now 20 *)

```

## 11.7 Using Closures Instead of Objects

```

(* Since record field names must be unique to their enclosing module,
   define a module to encapsulate the fields of the record type that
   will contain the "methods". *)
module Counter = struct
  type t = { next : unit -> int;
            prev : unit -> int;
            last : unit -> int;
            get  : unit -> int;
            set  : int  -> unit;
            bump : int  -> unit;
            reset : unit -> int }

  let make count =
    let start = count in
    let count = ref start in
    let prev () = decr count; !count in
    { next = (fun () -> incr count; !count);
      prev = prev; last = prev;
      get  = (fun () -> !count);
      set  = (fun count' -> count := count');
      bump = (fun count' -> count := !count + count');
      reset = (fun () -> count := start; !count)
    }
end

(* Create and use a couple of counters. *)

```

```

let () =
  let c1 = Counter.make 20 in
  let c2 = Counter.make 77 in

  Printf.printf "next c1: %d\n" (c1.Counter.next ()); (* 21 *)
  Printf.printf "next c2: %d\n" (c2.Counter.next ()); (* 78 *)
  Printf.printf "next c1: %d\n" (c1.Counter.next ()); (* 22 *)
  Printf.printf "last c1: %d\n" (c1.Counter.prev ()); (* 21 *)
  Printf.printf "old c2: %d\n" (c2.Counter.reset ()); (* 77 *)

(* Same as above, but using a "local open" to temporarily expose
   the record fields for convenience. *)
let () =
  let c1 = Counter.make 20 in
  let c2 = Counter.make 77 in
  let module Local = struct
    open Counter
    let () =
      Printf.printf "next c1: %d\n" (c1.next ()); (* 21 *)
      Printf.printf "next c2: %d\n" (c2.next ()); (* 78 *)
      Printf.printf "next c1: %d\n" (c1.next ()); (* 22 *)
      Printf.printf "last c1: %d\n" (c1.prev ()); (* 21 *)
      Printf.printf "old c2: %d\n" (c2.reset ()); (* 77 *)
    end in ()
end in ()

```

## 11.8 Creating References to Methods

```

(* There is no need to use references just to have a function that
   calls a method. Either write a lambda: *)

let mref = fun x y z -> obj#meth x y z

(* Or, just refer to the method directly: *)

let mref = obj#meth

(* Later... *)

let () = mref "args" "go" "here"

```

## 11.9 Constructing Records

```

#load "str.cma";;

type record = { name : string;
                empno : int;
                mutable title : string;
                mutable age : int;
                mutable salary : float;
                mutable pals : string list;
              }

let record = { name = "Jason";

```

```

        empno = 132;
        title = "deputy peon";
        age = 23;
        salary = 37000.00;
        pals = [ "Norbert"; "Rhys"; "Phineas" ]
    }

let () =
    Printf.printf "I am %s, and my pals are %s.\n"
        record.name
        (String.concat " ", " record.pals)

let byname = Hashtbl.create 0

let () =
    (* store record *)
    Hashtbl.replace byname record.name record;

    (* later on, look up by name *)
    begin
        try
            let rp = Hashtbl.find byname "Aron" in
                Printf.printf "Aron is employee %d\n" rp.empno
            with Not_found ->
                (* raised if missing *)
                ()
        end;

    (* give jason a new pal *)
    let jason = Hashtbl.find byname "Jason" in
        jason.pals <- "Theodore" :: jason.pals;
        Printf.printf "Jason now has %d pals\n" (List.length jason.pals);

    Hashtbl.iter
        (fun name record ->
            Printf.printf "%s is employee number %d\n" name record.empno)
        byname

let employees = Hashtbl.create 0

let () =
    (* store record *)
    Hashtbl.replace employees record.empno record;

    (* lookup by id *)
    begin
        try
            let rp = Hashtbl.find employees 132 in
                Printf.printf "employee number 132 is %s\n" rp.name
            with Not_found ->
                ()
        end;
    end;

```

```

let jason = Hashtbl.find byname "Jason" in
jason.salary <- jason.salary *. 1.035

(* Return true if the string s contains the given substring. *)
let contains s substring =
  try ignore (Str.search_forward (Str.regexp_string substring) s 0); true
  with Not_found -> false

let () =
  (* A filter function for hash tables, written as a fold. *)
  let grep f hash =
    Hashtbl.fold
      (fun key value result ->
        if f value then value :: result else result)
      hash [] in

  (* Select records matching criteria. *)
  let peons =
    grep (fun employee -> contains employee.title "peon") employees in
  let tsevens =
    grep (fun employee -> employee.age = 27) employees in

  (* Go through all records. *)
  let records = Hashtbl.fold (fun _ v a -> v :: a) employees [] in
  List.iter
    (fun rp ->
      Printf.printf "%s is age %d.\n" rp.name rp.age)
    (List.sort (fun r1 r2 -> compare r1.age r2.age) records)

  (* Create an array of lists of records by age. *)
  let byage = Array.create 150 []
  let () =
    Hashtbl.iter
      (fun _ employee ->
        byage.(employee.age) <- employee :: byage.(employee.age))
    employees

  (* Print all employees by age. *)
  let () =
    Array.iteri
      (fun age emps ->
        match emps with
        | [] -> ()
        | _ ->
          Printf.printf "Age %d: " age;
          List.iter (fun emp -> Printf.printf "%s " emp.name) emps;
          print_newline ())
      byage

  (* Similar approach using List.map and String.concat. *)
  let () =
    Array.iteri
      (fun age emps ->

```

```

    match emps with
    | [] -> ()
    | _ ->
        Printf.printf "Age %d: %s\n" age
        (String.concat ", " (List.map (fun r -> r.name) emps)))
byage

```

## 11.10 Reading and Writing Hash Records to Text Files

```

#load "str.cma";;

(* Define a list reference to contain our data. *)
let (list_of_records : (string, string) Hashtbl.t list ref) = ref []

(* Read records from standard input. *)
let () =
  let regexp = Str.regexp "\\([^:]+\\):[ \\t]*\\((.*)\\" in
  let record = ref (Hashtbl.create 0) in
  begin
    try
      while true do
        let line = read_line () in
        if Str.string_match regexp line 0
        then
          let field = Str.matched_group 1 line in
          let value = Str.matched_group 2 line in
          Hashtbl.replace !record field value
        else
          (list_of_records := !record :: !list_of_records;
           record := Hashtbl.create 0)
        done
      with End_of_file ->
        if Hashtbl.length !record > 0
        then list_of_records := !record :: !list_of_records
    end

  (* Write records to standard output. *)
  let () =
    List.iter
      (fun record ->
        Hashtbl.iter
          (fun field value -> Printf.printf "%s: %s\n" field value)
          record;
        print_newline ())
      !list_of_records

```

## 11.11 Printing Data Structures

```

(* If you are in the OCaml toplevel, simply enter an expression to
view its type and value. *)
# let reference = ref ( [ "foo", "bar" ],
                        3,
                        fun () -> print_endline "hello, world" );;

```

```

val reference : ((string * string) list * int * (unit -> unit)) ref =
  {contents = ([("foo", "bar"]), 3, <fun>)}

(* From within your own programs, use the Std.print and Std.dump
   functions from the Extlib library, available at
   http://ocaml-lib.sourceforge.net/ *)
# Std.print reference;;
(([("foo", "bar")], 3, <closure>))
- : unit = ()
# Std.dump reference;;
- : string = "([(\\"foo\\", \\"bar\\"), 3, <closure>))"

```

## 11.12 Copying Data Structures

```

(* Immutable data structures such as int, char, float, tuple, list, Set,
   and Map can be copied by assignment. *)
let v2 = v1
let r2 = ref !r1

(* Objects can be shallow-copied using Oo.copy. *)
let o2 = Oo.copy o1

(* Several built-in types include copy functions. *)
let a2 = Array.copy a1
let h2 = Hashtbl.copy h1
let s2 = String.copy s1

(* Any data structure can be deep-copied by running it through Marshal,
   though this is not very efficient. *)
let (copy : 'a -> 'a) =
  fun value ->
    Marshal.from_string
      (Marshal.to_string value [Marshal.Closures])
    0

```

## 11.13 Storing Data Structures to Disk

```

let () =
  (* Store a data structure to disk. *)
  let out_channel = open_out_bin "filename" in
  Marshal.to_channel out_channel data [];
  close_out out_channel;

  (* Load a data structure from disk. *)
  let in_channel = open_in_bin "filename" in
  let data = Marshal.from_channel in_channel in
  (* ... *)
  ();;

#load "unix.cma";;
let () =
  (* Store a data structure to disk, with exclusive locking. *)
  let out_channel = open_out_bin "filename" in

```

```

Unix.lockf (Unix.descr_of_out_channel out_channel) Unix.F_LOCK 0;
Marshal.to_channel out_channel data [];
close_out out_channel;

(* Load a data structure from disk, with shared locking. *)
let in_channel = open_in_bin "filename" in
Unix.lockf (Unix.descr_of_in_channel in_channel) Unix.F_RLOCK 0;
let data = Marshal.from_channel in_channel in
(* ... *)
()

```

## 11.14 Transparently Persistent Data Structures

(\* See recipes 14.8 and 14.9 for examples of (mostly) transparent persistence using DBM and Marshal in a type-safe manner. \*)

## 11.15 Program: Binary Trees

(\* bintree - binary tree demo program \*)

```

type 'a tree = { value : 'a;
                 left  : 'a tree option;
                 right : 'a tree option }

let rec string_of_tree tree =
  Printf.sprintf "{ value = %d; left = %s; right = %s }"
    tree.value
    (match tree.left with
     | None -> "None"
     | Some tree -> Printf.sprintf "Some (%s)" (string_of_tree tree))
    (match tree.right with
     | None -> "None"
     | Some tree -> Printf.sprintf "Some (%s)" (string_of_tree tree))

(* insert given value into proper point of
   provided tree.  If no tree provided,
   fill one in for our caller. *)
let rec insert tree value =
  match tree with
  | None -> { value = value; left = None; right = None }
  | Some tree ->
    if tree.value > value
    then { value = tree.value;
           left  = Some (insert tree.left value);
           right = tree.right }
    else if tree.value < value
    then { value = tree.value;
           left  = tree.left;
           right = Some (insert tree.right value) }
    else tree

(* recurse on left child,
   then show current value,

```

```

    then recurse on right child. *)
let rec in_order tree =
  match tree with
  | None -> ()
  | Some tree ->
    in_order tree.left;
    print_int tree.value;
    print_string " ";
    in_order tree.right

(* show current value,
   then recurse on left child,
   then recurse on right child. *)
let rec pre_order tree =
  match tree with
  | None -> ()
  | Some tree ->
    print_int tree.value;
    print_string " ";
    pre_order tree.left;
    pre_order tree.right

(* recurse on left child,
   then recurse on right child,
   then show current value. *)
let rec post_order tree =
  match tree with
  | None -> ()
  | Some tree ->
    post_order tree.left;
    post_order tree.right;
    print_int tree.value;
    print_string " "

(* find out whether provided value is in the tree.
   if so, return the node at which the value was found.
   cut down search time by only looking in the correct
   branch, based on current value. *)
let rec search tree value =
  match tree with
  | Some tree ->
    if tree.value = value
    then Some tree
    else search (if value < tree.value then tree.left else tree.right) value
  | None -> None

(* reference to the root of the tree *)
let root = ref None

(* first generate 20 random inserts *)
let () =
  Random.self_init ();
  for n = 0 to 19 do

```

```

    root := Some (insert !root (Random.int 1000))
done

(* now dump out the tree all three ways *)
let () =
  print_string "Pre order: "; pre_order !root; print_newline ();
  print_string "In order: "; in_order !root; print_newline ();
  print_string "Post order: "; post_order !root; print_newline ()

(* prompt until EOF *)
let () =
  try
    while true do
      let line = read_line () in
      let num = int_of_string line in
      let found = search !root num in
      match found with
      | Some tree ->
          Printf.printf "Found %d at %s, %d\n"
            num
            (string_of_tree tree)
            tree.value
      | None ->
          Printf.printf "No %d in the tree\n" num
    done
  with End_of_file ->
    ()

```

## 12 Packages, Libraries, and Modules

(\* When an OCaml source file is compiled, it becomes a module. The name of the module is the capitalized form of the filename. For example, if the source file is "my\_module.ml", the module name is "My\_module".

Modules can also be created explicitly within a source file. If "my\_module.ml" contains "module Foo = struct ... end", a module named "My\_module.Foo" will be created.

Here is an example of the definition and use of two modules within a single source file: \*)

```
module Alpha = struct
  let name = "first"
end

module Omega = struct
  let name = "last"
end

let () =
  Printf.printf "Alpha is %s, Omega is %s.\n"
    Alpha.name Omega.name

(* Alpha is first, Omega is last. *)

(*-----*)

(* The "#use" and "#load" commands are known as toplevel directives.
   They can only be used while interacting with the interpreter or from
   scripts that are run using the "ocaml" program. *)

(* "#use" loads a source file into the current scope. *)
#use "FileHandle.ml";

(* "#load" loads a module from a compiled bytecode file. This has the
   same effect as including this file during bytecode compilation. *)
#load "FileHandle.cmo";

(* "#load" can be used with libraries as well as modules. Bytecode
   libraries use an extension of ".cma". *)
#load "library.cma";

(* The "open" statement can be used in any source file. It allows any
   values defined within a module to be used without being prefixed by
   the module name. *)
open FileHandle

(* Modules form a hierarchy; submodules can be opened in a similar
   fashion by prefixing them with the parent module's name. *)
open Cards.Poker
```

```
(* It is often convenient to use Gerd Stolpmann's "findlib" system,
   which makes it considerably easier to load libraries into the
   interpreter. *)
```

```
# #use "topfind";;
- : unit = ()
Findlib has been successfully loaded. Additional directives:
  #require "package";;      to load a package
  #list;;                  to list the available packages
  #camlp4o;;               to load camlp4 (standard syntax)
  #camlp4r;;               to load camlp4 (revised syntax)
  #predicates "p,q,...";;  to set these predicates
  Topfind.reset();;        to force that packages will be reloaded
  #threads;;               to enable threads
```

```
- : unit = ()
# #require "extlib";;
/usr/lib/ocaml/3.10.2/extlib: added to search path
/usr/lib/ocaml/3.10.2/extlib/extLib.cma: loaded
```

```
(* The above use of "#require" has the same effect as typing the
   following: *)
```

```
#directory "+extlib";;
#load "extLib.cma";;
```

```
(* More information on the "findlib" system is available here:
   http://projects.camlcity.org/projects/findlib.html
```

The "#directory" directive above is built into OCaml and allows you to add additional directories to the path that is searched when loading modules. You can use a prefix of '+' to indicate that the directory is under the standard library path, which is usually something like "/usr/lib/ocaml/3.10.2/".

Modules can be easily aliased using assignment. This will also cause the interpreter to output the module's signature, which can be used as a quick reference. \*)

```
# module S = ExtString.String;;
module S :
  sig
    val init : int -> (int -> char) -> string
    val find : string -> string -> int
    val split : string -> string -> string * string
    val nsplit : string -> string -> string list
    val join : string -> string list -> string
    ...
  end
# S.join;;
- : string -> string list -> string = <fun>
```

```
(* Many useful libraries can be found at The Caml Hump:
```

```
http://caml.inria.fr/cgi-bin/hump.cgi *)
```

## 12.1 Defining a Module's Interface

```
(* Interfaces, also known as module types or signatures, are usually
   saved in files with the same name as the corresponding module but
   with a ".mli" extension. For instance, if the module is defined in
   "YourModule.ml", the interface will be in "YourModule.mli". *)
```

```
(* YourModule.mli *)
val version : string
```

```
(* YourModule.ml *)
let version = "1.00"
```

```
(* As with modules, interfaces can also be defined explicitly inside
   of a source file. *)
```

```
module type YourModuleSignature =
sig
  val version : string
end
```

```
module YourModule : YourModuleSignature =
struct
  let version = "1.00"
end
```

```
(* Signatures can also be anonymous. *)
```

```
module YourModule :
sig
  val version : string
end =
struct
  let version = "1.00"
end
```

## 12.2 Trapping Errors in require or use

```
(* Due to static typing, missing modules are detected at compilation
   time, so this is not normally an error you can catch (or need to).
   When using ocaml interactively or as an interpreter, the "#load"
   directive can fail, resulting in a message like the following:
```

```
Cannot find file <filename>.
```

```
being printed to standard output. This is also not an error you can
catch, and its occurrence will not stop the script from executing.
It is possible to dynamically load modules and detect the failure of
this operation with Dynlink. An example is given in the next recipe. *)
```

### 12.3 Delaying use Until Run Time

```
(* Registry.ml *)
let (registry : (string, unit -> unit) Hashtbl.t) = Hashtbl.create 32

(* SomeModule.ml *)
let say_hello () = print_endline "Hello, world!"
let () = Hashtbl.replace Registry.registry "say_hello" say_hello

(* Main program *)
let filename = "SomeModule.cmo"
let funcname = "say_hello"
let () =
  Dynlink.init ();
  (try Dynlink.loadfile filename
   with Dynlink.Error e -> failwith (Dynlink.error_message e));
  (Hashtbl.find Registry.registry funcname) ()

(* Note that the Dynlink module currently supports dynamic loading of
  bytecode modules only. There is a project to add support for dynamic
  loading of native code which has been merged with OCaml's CVS HEAD.
  Details are available at http://alain.frisch.fr/natdynlink.html *)
```

### 12.4 Making Variables Private to a Module

```
#load "str.cma";;
module Flipper :
sig
  val flip_boundary : string -> string
  val flip_words : string -> string
end =
struct
  let separatrix = ref " " (* hidden by signature *)
  let flip_boundary sep =
    let prev_sep = !separatrix in
    separatrix := sep;
    prev_sep
  let flip_words line =
    let words = Str.split (Str.regexp_string !separatrix) line in
    String.concat !separatrix (List.rev words)
end
```

### 12.5 Determining the Caller's Package

```
(* This is very difficult to do in OCaml due to the lack of reflection
  capabilities. Determining the current module name is reasonably easy,
  however, by using the __FILE__ constant exposed by camlp4's macro
  extensions. *)

(*pp camlp4of *)
let __MODULE__ = String.capitalize (Filename.chop_extension __FILE__)
let () = Printf.printf "I am in module %s\n" __MODULE__
```

## 12.6 Automating Module Clean-Up

```
(* Use the built-in function, "at_exit", to schedule clean-up handlers
   to run when the main program exits. *)

#load "unix.cma";;

let logfile = "/tmp/mylog"
let lf = open_out logfile

let logmsg msg =
  Printf.fprintf lf "%s %d: %s\n%!"
    Sys.argv.(0) (Unix.getpid ()) msg

(* Setup code. *)
let () =
  logmsg "startup"

(* Clean-up code. *)
let () =
  at_exit
    (fun () ->
      logmsg "shutdown";
      close_out lf)
```

## 12.7 Keeping Your Own Module Directory

```
(* To add a directory to the module include path, pass the "-I" option
   to any of the compiler tools. For example, if you have a module in
   ~/ocamllib called Utils with a filename of utils.cmo, you can build
   against this module with the following: *)

$ ocamlc -I ~/ocamllib utils.cmo test.ml -o test

(* Within the toplevel interpreter, and from ocaml scripts, you can use
   the "#directory" directive to add directories to the include path: *)

#directory "/home/myuser/ocamllib";;
#load "utils.cmo";;

(* In both cases, prefixing the include directory with a '+' indicates
   that the directory should be found relative to the standard include
   path. *)

#directory "+pcre";;
#load "pcre.cma";;

(* If you have findlib installed, you can print out the include path by
   typing "ocamlfind printconf path" at the command line. *)

$ ocamlfind printconf path
/usr/local/lib/ocaml/3.10.2
/usr/lib/ocaml/3.10.2
```

```
/usr/lib/ocaml/3.10.2/METAS
```

```
(* Instead of keeping a directory of ".cmo" (or ".cmx") files, you may
prefer to build a library (.cma for bytecode, ".cmxa" for native).
This will pack all of your modules into a single file that is easy to
use during compilation: *)
```

```
$ ocamlc -a slicer.cmo dicer.cmo -o tools.cma
$ ocamlc tools.cma myprog.ml -o myprog
```

## 12.8 Preparing a Module for Distribution

```
(* The easiest way to prepare a library for distribution is to build with
OCamlMakefile and include a META file for use with findlib.
```

```
OCamlMakefile is available here:
http://www.ocaml.info/home/ocaml\_sources.html#OCamlMakefile
```

```
findlib is available here:
http://projects.camlcity.org/projects/findlib.html *)
```

```
(* Put the following in a file called "Makefile" and edit to taste: *)
```

```
OCAMLMAKEFILE = OCamlMakefile

RESULT = mylibrary
SOURCES = mylibrary.mli mylibrary.ml
PACKS = pcre

all: native-code-library byte-code-library
install: libinstall
uninstall: libuninstall

include $(OCAMLMAKEFILE)
```

```
(* Put the following in a file called "META" and edit to taste: *)
```

```
name = "mylibrary"
version = "1.0.0"
description = "My library"
requires = "pcre"
archive(byte) = "mylibrary.cma"
archive(native) = "mylibrary.cmxa"
```

```
(* Now you can build bytecode and native libraries with "make" and
install them into the standard library location with "make install".
If you make a change, you will have to "make uninstall" before you
can "make install" again. Once a library is installed, it's simple
to use: *)
```

```
$ ledit ocaml
Objective Caml version 3.10.2
```

```

# #use "topfind";;
- : unit = ()
Findlib has been successfully loaded. Additional directives:
  #require "package";;      to load a package
  #list;;                  to list the available packages
  #camlp4o;;               to load camlp4 (standard syntax)
  #camlp4r;;               to load camlp4 (revised syntax)
  #predicates "p,q,...";;  to set these predicates
  Topfind.reset();;        to force that packages will be reloaded
  #thread;;                to enable threads

- : unit = ()
# #require "mylibrary";;
/usr/lib/ocaml/3.10.2/pcr: added to search path
/usr/lib/ocaml/3.10.2/pcr/pcr.cma: loaded
/usr/local/lib/ocaml/3.10.2/mylibrary: added to search path
/usr/local/lib/ocaml/3.10.2/mylibrary/mylibrary.cma: loaded

(* To compile against your new library, use the "ocamlfind" tool as a
   front-end to "ocamlc" and "ocamlopt": *)

$ ocamlfind ocamlc -package mylibrary myprogram.ml -o myprogram
$ ocamlfind ocamlopt -package mylibrary myprogram.ml -o myprogram

```

## 12.9 Speeding Module Loading with SelfLoader

```

(* OCaml supports native compilation. If module load time is an issue,
   it's hard to find a better solution than "ocamlopt". If compilation
   is slow as well, try "ocamlopt.opt", which is the natively-compiled
   native compiler. *)

```

## 12.10 Speeding Up Module Loading with Autoloader

```

(* This recipe is not relevant or applicable to OCaml. *)

```

## 12.11 Overriding Built-In Functions

```

#load "unix.cma";;

(* The Unix module returns the time as a float. Using a local module
   definition and an "include", we can override this function to return
   an int32 instead. (This is a bit silly, but it illustrates the basic
   technique. *)
module Unix = struct
  include Unix
  let time () = Int32.of_float (time ())
end

(* Use the locally modified Unix.time function. *)
let () =
  let start = Unix.time () in
  while true do
    Printf.printf "%ld\n" (Int32.sub (Unix.time ()) start)

```

```

done

(* Operators can also be locally modified. Here, we'll temporarily
   define '-' as int32 subtraction. *)
let () =
  let ( - ) = Int32.sub in
  let start = Unix.time () in
  while true do
    Printf.printf "%ld\n" (Unix.time () - start)
  done

```

## 12.12 Reporting Errors and Warnings Like Built-Ins

```

(* There are two built-in functions that raise standard exceptions.
   Many standard library functions use these. "invalid_arg" raises
   an Invalid_argument exception, which takes a string parameter: *)
let even_only n =
  if n land 1 <> 0 (* one way to test *)
  then invalid_arg (string_of_int n);
  (* ... *)
  ()

(* "failwith" raises a Failure exception, which also takes a string
   parameter (though it is typically used to identify the name of
   the function as opposed to the argument). *)
let even_only n =
  if n mod 2 <> 0 (* here's another *)
  then failwith "even_only";
  (* ... *)
  ()

(* In most cases, it is preferable to define your own exceptions. *)
exception Not_even of int
let even_only n =
  if n land 1 <> 0 then raise (Not_even n);
  (* ... *)
  ()

(* OCaml does not provide a facility for emitting warnings. You can
   write to stderr, which may be an acceptable substitute. *)
let even_only n =
  let n =
    if n land 1 <> 0 (* test whether odd number *)
    then (Printf.eprintf "%d is not even, continuing\n%!" n; n + 1)
    else n in
  (* ... *)
  ()

```

## 12.13 Referring to Packages Indirectly

```

(* Generally, it is best to use tables of functions, possibly with
   Dynlink, to delay the choice of module and function until runtime.
   It is however possible--though inelegant--to (ab)use the toplevel

```

```

    for this purpose. *)

open Printf

(* Toplevel evaluator. Not type-safe. *)
let () = Toploop.initialize_toplevel_env ()
let eval text = let lexbuf = (Lexing.from_string text) in
  let phrase = !Toploop.parse_toplevel_phrase lexbuf in
  ignore (Toploop.execute_phrase false Format.std_formatter phrase)
let get name = Obj.obj (Toploop.getvalue name)
let set name value = Toploop.setvalue name (Obj.repr value)

(* Some module and value names, presumably not known until runtime. *)
let modname = "Sys"
let varname = "ocaml_version"
let aryname = "argv"
let funcname = "getenv"

(* Use the toplevel to evaluate module lookups dynamically. *)
let () =
  eval (sprintf "let (value : string) = %s.%s;;" modname varname);
  print_endline (get "value");
  eval (sprintf "let (values : string array) = %s.%s;;" modname aryname);
  Array.iter print_endline (get "values");
  eval (sprintf "let (func : string -
> string) = %s.%s;;" modname funcname);
  print_endline ((get "func") "HOME");

```

## 12.14 Using h2ph to Translate C #include Files

(\* There are several tools for translating C header files to OCaml bindings, many of which can be found at The Caml Hump:

<http://caml.inria.fr/cgi-bin/hump.en.cgi?sort=0&browse=42>

Of the available tools, "ocamlffi" (also known as simply "FFI") seems to work best at accomplishing the task of parsing header files, but it has not been maintained in many years and cannot handle the deep use of preprocessor macros in today's Unix headers. As a result, it is often necessary to create a header file by hand, and so long as this is required, better results can be achieved with Xavier Leroy's CamlIDL tool. CamlIDL can be found here:

[http://caml.inria.fr/pub/old\\_caml\\_site/camlidl/](http://caml.inria.fr/pub/old_caml_site/camlidl/)

The following recipes will use CamlIDL. First, we'll wrap the Unix "gettimeofday" system call by writing the following to a file named "time.idl": \*)

```

quote(C, "#include <sys/time.h>");

struct timeval {
  [int32] int tv_sec;

```

```

    [int32] int tv_usec;
};

struct timezone {
    int tz_minuteswest;
    int tz_dsttime;
};

int gettimeofday([out] struct timeval *tv, [in] struct timezone *tz);

(* We can now build three files, "time.ml", "time.mli", and
   "time_stubs.c", corresponding to the OCaml implementation, OCaml
   interface, and OCaml-to-C stubs, by running the following command: *)

$ camlidl -no-include time.idl

(* CamlIDL automatically translates the two structs defined in the IDL
   into OCaml record types and builds an external function reference
   for "gettimeofday", resulting in the following generated OCaml
   implementation in "time.ml": *)

(* File generated from time.idl *)

type timeval = {
    tv_sec: int32;
    tv_usec: int32;
}
and timezone = {
    tz_minuteswest: int;
    tz_dsttime: int;
}

external gettimeofday : timezone option -> int * timeval
    = "camlidl_time_gettimeofday"

(* Now, we can use "ocamlc -c" as a front-end to the C compiler to build
   the stubs, producing time_stubs.o. *)

$ ocamlc -c time_stubs.c

(* The OCaml source can be built and packed into a library along with
   the compiled stubs using "ocamlc -a": *)

$ ocamlc -a -custom -o time.cma time.mli time.ml time_stubs.o \
    -cclib -lcamlidl

(* Finally, we can write a simple test program to use our newly-exposed
   "gettimeofday" function. *)

(* test.ml *)
let time () =
    let res, {Time.tv_sec=seconds; tv_usec=microseconds} =
        Time.gettimeofday None in

```

```

    Int32.to_float seconds +. (Int32.to_float microseconds /. 1_000_000.)
let () = Printf.printf "%f\n" (time ())

(* Compiling this test program is straightforward. *)

$ ocamlc -o test time.cma test.ml

(* Running it produces the current time with millisecond precision. *)

$ ./test
1217173408.931277

(*-----*)

(* The next two recipes will wrap the Unix "ioctl" function, allowing
   us to make a few low-level I/O system calls. To make things easier,
   we'll use the following Makefile (make sure you use tabs, not spaces,
   if you cut and paste this code): *)

all: jam winsz

jam: ioctl.cma jam.ml
    ocamlc -o jam ioctl.cma jam.ml

winsz: ioctl.cma winsz.ml
    ocamlc -o winsz ioctl.cma winsz.ml

ioctl.cma: ioctl.mli ioctl.ml ioctl_stubs.o
    ocamlc -a -custom -o ioctl.cma ioctl.mli ioctl.ml ioctl_stubs.o \
        -cclib -lcamlidl

ioctl_stubs.o: ioctl_stubs.c
    ocamlc -c ioctl_stubs.c

ioctl.mli ioctl.ml ioctl_stubs.c: ioctl.idl
    camlidl -no-include ioctl.idl

clean:
    rm -f *.cma *.cmi *.cmo *.c *.o ioctl.ml ioctl.mli jam winsz

(*-----*)

(* ioctl.idl: *)

quote(C, "#include <sys/ioctl.h>");

enum ioctl {
    TIOCSTI,
    TIOCGWINSZ,
};

int ioctl([in] int fd,
          [in] enum ioctl request,

```

```

[in, out, string] char *argp);

(*-----*)

(* jam - stuff characters down STDIN's throat *)

(* Simulate input on a given terminal. *)
let jam ?(tty=0) s =
  String.iter
    (fun c -> ignore (Ioctl.ioctl tty Ioctl.TIOCSTI (String.make 1 c))) s

(* Stuff command-line arguments into STDIN. *)
let () = jam (String.concat " " (List.tl (Array.to_list (Sys.argv))))

(*-----*)

(* winsz - find x and y for chars and pixels *)

(* Decode a little-endian short integer from a string and offset. *)
let decode_short s i =
  Char.code s.[i] lor Char.code s.[i + 1] lsl 8

(* Read and display the window size. *)
let () =
  let winsize = String.make 8 '\000' in
  ignore (Ioctl.ioctl 0 Ioctl.TIOCGWINSZ winsize);
  let row = decode_short winsize 0 in
  let col = decode_short winsize 2 in
  let xpixel = decode_short winsize 4 in
  let ypixel = decode_short winsize 6 in
  Printf.printf "(row,col) = (%d,%d)" row col;
  if xpixel <> 0 || ypixel <> 0
  then Printf.printf " (xpixel,ypixel) = (%d,%d)" xpixel ypixel;
  print_newline ()

```

## 12.15 Using h2xs to Make a Module with C Code

```

(* Building libraries with C code is much easier with the aid of
OCamlMakefile. The following Makefile is all it takes to build
the "time" library from the previous recipe: *)

OCAMLMAKEFILE = OCamlMakefile

RESULT = time
SOURCES = time.idl
NOIDLHEADER = yes

all: byte-code-library native-code-library

include $(OCAMLMAKEFILE)

(* Now, a simple "make" will perform the code generation with camlidl
and produce static and dynamic libraries for bytecode and native

```

compilation. Furthermore, "make top" will build a custom toplevel interpreter called "time.top" with the Time module built in: \*)

```
$ ./time.top
    Objective Caml version 3.10.2

# Time.gettimeofday None;;
- : int * Time.timeval =
(0, {Time.tv_sec = 12174835501; Time.tv_usec = 6452041})

(* With the addition of a "META" file combined with the "libinstall"
   and "libuninstall" targets, this library can be installed to the
   standard location for use in other projects. See recipe 12.8,
   "Preparing a Module for Distribution", for an example. *)
```

## 12.16 Documenting Your Module with Pod

```
(** Documentation for OCaml programs can be generated with the ocaml doc
   tool, included in the standard distribution. Special comments like
   this one begin with two asterisks which triggers ocaml doc to
   include them in the documentation. The first special comment in a
   module becomes the main description for that module. *)

(** Comments can be placed before variables... *)
val version : string

(** ...functions... *)
val cons : 'a -> 'a list -> 'a list

(** ...types... *)
type choice = Yes | No | Maybe of string

(* ... and other constructs like classes, class types, modules, and
   module types. Simple comments like this one are ignored. *)

(** {2 Level-two headings look like this} *)

(** Text in [square brackets] will be formatted using a monospace font,
   ideal for identifiers and other bits of code. Text written in curly
   braces with a bang in front {!Like.this} will be hyperlinked to the
   corresponding definition. *)

(* To generate HTML documentation, use a command like the following: *)

$ ocaml doc -html -d destdir Module1.mli Module1.ml ...

(* To generate Latex documentation, use a command like the following: *)

$ ocaml doc -latex -d destdir Module1.mli Module1.ml ...

(* If you use OCamlMakefile, you can type "make doc" to build HTML and
   PDF documentation for your entire project. You may want to customize
   the OCAMLDOC and DOC_FILES variables to suit your needs. *)
```

## 12.17 Building and Installing a CPAN Module

```
(* Installing a module from The Caml Hump differs from project to
project, since it is not as standardized as CPAN. However, in most
cases, "make" and "make install" do what you expect. Here's how to
install easy-format, which can be found on the Hump at the following
URL: http://caml.inria.fr/cgi-bin/hump.en.cgi?contrib=651 *)
```

```
$ tar xzf easy-format.tar.gz
$ cd easy-format
$ make
ocamlc -c easy_format.mli
ocamlc -c -dtypes easy_format.ml
touch bytecode
ocamlc -c easy_format.mli
ocamlopt -c -dtypes easy_format.ml
touch nativecode

$ sudo make install
[sudo] password for root: .....
echo "version = \"1.0.0\"" > META; cat META.tpl >> META
INSTALL_FILES="META easy_format.cmi easy_format.mli"; \
    if test -f bytecode; then \
        INSTALL_FILES="$INSTALL_FILES easy_format.cmo "; \
    fi; \
    if test -f nativecode; then \
        IN-
STALL_FILES="$INSTALL_FILES easy_format.cmx easy_format.o"; \
    fi; \
    ocamlfind install easy-format $INSTALL_FILES
Installed /usr/local/lib/ocaml/3.10.2/easy-format/easy_format.o
Installed /usr/local/lib/ocaml/3.10.2/easy-format/easy_format.cmx
Installed /usr/local/lib/ocaml/3.10.2/easy-format/easy_format.cmo
Installed /usr/local/lib/ocaml/3.10.2/easy-format/easy_format.mli
Installed /usr/local/lib/ocaml/3.10.2/easy-format/easy_format.cmi
Installed /usr/local/lib/ocaml/3.10.2/easy-format/META
```

## 12.18 Example: Module Template

```
(* Some.ml *)
module Module =
struct
  (* set the version for version checking *)
  let version = "0.01"

  (* initialize module globals (accessible as Some.Module.var1 *)
  let var1 = ref ""
  let hashit = Hashtbl.create 0

  (* file-private lexicals go here *)
  let priv_var = ref ""
  let secret_hash = Hashtbl.create 0
```

```

(* here's a file-private function *)
let priv_func () =
  (* stuff goes here. *)
  ()

(* make all your functions, whether exported or not *)
let func1 () = (* ... *) ()
let func2 () = (* ... *) ()
let func3 a b = (* ... *) ()
let func4 h = (* ... *) ()

(* module clean-up code here *)
let () =
  at_exit
    (fun () ->
      (* ... *)
      ())
end

(* Some.mli *)
module Module :
sig
  val version : string
  val var1 : string ref
  val hashit : (string, string) Hashtbl.t
  (* priv_var, secret_hash, and priv_func are omitted,
     making them private and inaccessible... *)
  val func1 : unit -> unit
  val func2 : unit -> unit
  val func3 : 'a -> 'b -> unit
  val func4 : (string, string) Hashtbl.t -> unit
end

```

## 12.19 Program: Finding Versions and Descriptions of Installed Modules

```

(* Use "findlib". You can use the "ocamlfind" program to get a list of
   installed libraries from the command line: *)

```

```

$ ocamlfind list
benchmark          (version: 0.6)
bigarray           (version: [distributed with Ocaml])
cairo              (version: n/a)
cairo.lablgtk2    (version: n/a)
calendar          (version: 2.0.2)
camlimages        (version: 2.2.0)
camlimages.graphics (version: n/a)
camlimages.lablgtk2 (version: n/a)
camlp4            (version: [distributed with Ocaml])
camlp4.exceptiontracer (version: [distributed with Ocaml])
camlp4.extend     (version: [distributed with Ocaml])
...

```

```

(* You can also use the "#list" directive from the interpreter: *)

```

```

$ ledit ocaml
    Objective Caml version 3.10.2

# #use "topfind";;
- : unit = ()
Findlib has been successfully loaded. Additional directives:
  #require "package";;      to load a package
  #list;;                   to list the available packages
  #camlp4o;;                to load camlp4 (standard syntax)
  #camlp4r;;                to load camlp4 (revised syntax)
  #predicates "p,q,...";;  to set these predicates
  Topfind.reset();;        to force that packages will be reloaded
  #thread;;                 to enable threads

- : unit = ()
# #list;;
benchmark      (version: 0.6)
bigarray       (version: [distributed with Ocaml])
cairo          (version: n/a)
cairo.lablgtk2 (version: n/a)
...

```

## 13 Classes, Objects, and Ties

```
(* The simplest object possible. This object has no data, no methods,
   and does not belong to any class. *)
let obj = object end

(* The simplest class possible, and an instance of it. *)
class encoder = object end
let obj = new encoder

(* A class living inside of a module. *)
module Data = struct
  class encoder = object end
end
let obj = new Data.encoder

(* An object with data and a method. *)
let obj = object
  val data = [3; 5]
  method at n = List.nth data n
end
let () =
  (* Display the object's identity (an integer) and call a method. *)
  Printf.printf "%d %d\n" (Oo.id obj) (obj#at 1)

(* A module containing a class with data and a method. *)
module Human = struct
  class ['a] cannibal data = object
    val data : 'a list = data
    method at n = List.nth data n
  end
end
let () =
  let obj = new Human.cannibal [3; 5] in
  Printf.printf "%d %d\n" (Oo.id obj) (obj#at 1)

(* Method calls are indicated by the '#' operator. *)
let encoded = obj#encode "data"

(* There is no notion of a class method in OCaml.
   Use a module-level function instead. *)
let encoded = Data.Encoder.encode "data"

(* Using the "class" keyword is much like defining a function. *)
class klass (initial_name : string) = object
  val mutable my_name = initial_name
  method name = my_name
  method set_name name = my_name <- name
end
let () =
  let obj = new klass "Class" in
  print_endline obj#name;
  obj#set_name "Clown";
```

```

print_endline obj#name

(* Initialization can be performed prior to object creation. *)
class random n =
  let rng = Random.State.make_self_init () in
  object
    method next () = Random.State.int rng n
  end
let () =
  let r = new random 10 in
  Printf.printf "Three random numbers: %d, %d, %d.\n"
    (r#next ()) (r#next ()) (r#next ())

(* Initialization can also be performed after object creation.
   Note the "self" parameter, which can be used much like the
   "this" reference in other OO languages. *)
class late_initializer name = object (self)
  val my_name = name
  method prepare_name () = String.capitalize my_name
  initializer Printf.printf "%s is ready\n" (self#prepare_name ())
end
let obj = new late_initializer "object"

(* Methods are curried just like functions. This allows them to
   be used just like functions in many cases. It is customary for
   methods to take at least one argument (even if it is unit)
   unless they represent an object's attribute. *)
module Human = struct
  class cannibal (name : string) = object
    val mutable name = name
    method name = name
    method feed who = print_endline ("Feeding " ^ who)
    method move where = print_endline ("Moving to " ^ where)
    method die () = print_endline "Dying"
  end
end
let () =
  let lector = new Human.cannibal "Hannibal" in
  let feed, move, die = lector#feed, lector#move, lector#die in
  Printf.printf "Cannibal's name is %s\n" lector#name;
  feed "Zak";
  move "New York";
  die ()

```

### 13.1 Constructing an Object

```

#load "unix.cma";;

class klass args = object (self)
  val mutable start = 0.
  val mutable age = 0
  val extra = Hashtbl.create 0

```

```

(* Private method to initialize fields. Sets start to
   the current time, and age to 0. If called with arguments,
   init interprets them as key+value pairs to initialize the
   hashtable "extra" with. *)
method private init () =
  start <- Unix.time ();
  List.iter
    (fun (k, v) -> Hashtbl.replace extra k v)
    args

  initializer
    self#init ()
end

```

## 13.2 Destroying an Object

```

(* The Gc.finalise function can be used to create finalizers,
   which are like destructors but run at garbage collection time,
   for any value, not just objects. You can still use a method if
   you want: *)

```

```

class klass =
object (self)
  initializer
    Gc.finalise (fun self -> self#destroy ()) self
  method destroy () =
    Printf.printf "klass %d is dying\n" (Oo.id self)
end
let () =
  ignore (new klass);
  Gc.full_major ()

```

```

(* The "destroy" method above is public. If you want to keep it
   hidden, you can create a finalizer in a let-binding instead: *)

```

```

class klass =
  let destroy obj =
    Printf.printf "klass %d is dying\n" (Oo.id obj) in
object (self)
  initializer Gc.finalise destroy self
end

```

## 13.3 Managing Instance Data

```

(* Using a get and set method. *)

```

```

class person = object
  val mutable name = ""
  method name = name
  method set_name name' = name <- name'
end

```

```

(* Using a single method that does both get and set. *)
class person = object

```

```

val mutable age = 0

(* Unit argument required due to optional argument. *)
method age ?set () =
  match set with Some age' -> (age <- age'; age) | None -> age
end

(* Sample call of get and set: happy birthday! *)
let () =
  let obj = new person in
  ignore (obj#age ~set:(obj#age () + 1) ())

(* This class converts input when the name is set. *)
#load "str.cma";;
class person =
  let funny_chars = Str.regexp ".*[^\n\r\t A-Za-z0-9'-]" in
  let numbers = Str.regexp ".*[0-9]" in
  let not_blank = Str.regexp ".*[^\n\r\t ]" in
  let multiword = Str.regexp ".*[^\n\r\t ]+[\n\r\t ]+[^\n\r\t ]" in
object
  val mutable name = ""
  method name = name
  method set_name name' =
    if Str.string_match funny_chars name' 0
    then failwith "funny characters in name"
    else if Str.string_match numbers name' 0
    then failwith "numbers in name"
    else if not (Str.string_match not_blank name' 0)
    then failwith "name is blank"
    else if not (Str.string_match multiword name' 0)
    then failwith "prefer multiword name"
    else name <- String.capitalize name'
end

(* A typical class with attributes and methods. *)
class person = object
  (* Instance variables *)
  val mutable name = ""
  val mutable age = 0
  val mutable peers = []

  (* Accessors *)
  method name = name
  method set_name name' = name <- name'
  method age = age
  method set_age age' = age <- age'
  method peers = peers
  method set_peers peers' = peers <- peers'

  (* Behavioral methods *)
  method exclaim () =
    Printf.sprintf "Hi, I'm %s age %d, working with %s"
      name age (String.concat " " peers)

```

```

method happy_birthday () =
  age <- age + 1
end

```

### 13.4 Managing Class Data

```

(* There are no class methods in OCaml. Use a module instead. *)
module Person = struct
  let _body_count = ref 0
  let population () = !_body_count
  let destroy person = decr _body_count
  class person = object (self)
    initializer
      incr _body_count;
      Gc.finalise destroy self
  end
end

(* Later, the user can say this: *)
let () =
  let people = ref [] in
  for i = 1 to 10 do people := new Person.person :: !people done;
  Printf.printf "There are %d people alive.\n" (Person.population ())
  (* There are 10 people alive. *)

(* A class with an attribute that changes all instances when set. *)
module FixedArray = struct
  let _bounds = ref 7 (* default *)
  let max_bounds () = !_bounds
  let set_max_bounds max = _bounds := max
  class fixed_array = object
    method max_bounds = !_bounds
    method set_max_bounds bounds' = _bounds := bounds'
  end
end

let () =
  (* Set for whole class *)
  FixedArray.set_max_bounds 100;
  let alpha = new FixedArray.fixed_array in
  Printf.printf "Bound on alpha is %d\n" alpha#max_bounds;
  (* 100 *)
  let beta = new FixedArray.fixed_array in
  beta#set_max_bounds 50;
  Printf.printf "Bound on alpha is %d\n" alpha#max_bounds;
  (* 50 *)

(* To make the bounds read only, just remove the set method. *)

```

### 13.5 Using Classes as Structs

```

(* Immediate objects can be used like records, and their types are
   inferred automatically. Unlike with records, object fields names
   do not have to be unique to a module, which can be convenient. *)

```

```

let p = object
  method name = "Jason Smythe"
  method age = 13
  method peers = [| "Wilbur"; "Ralph"; "Fred" |]
end
(* val p : < age : int; name : string; peers : string array > = <obj> *)

(* Fetch various values, including the zeroth friend. *)
let () =
  Printf.printf "At age %d, %s's first friend is %s.\n"
    p#age p#name p#peers.(0)
(* At age 13, Jason Smythe's first friend is Wilbur. *)

(*-----*)

(* Immediate objects can be nested. *)
let folks = object
  method head = object
    method name = "John"
    method age = 34
  end
end
(* val folks : < head : < age : int; name : string > > = <obj> *)
let () =
  Printf.printf "%s's age is %d\n"
    folks#head#name folks#head#age
(* John's age is 34 *)

(*-----*)

(* If you want to maintain an invariant, it's better to use a class. *)
exception Unreasonable_age of int
class person init_name init_age = object (self)
  val mutable name = ""
  val mutable age = 0
  method name = name
  method age = age
  method set_name name' = name <- name'
  method set_age age' =
    if age' > 150 then raise (Unreasonable_age age') else age <- age'
  initializer
    self#set_name init_name;
    self#set_age init_age
end

```

## 13.6 Cloning Objects

```

(* Objects can be cloned with Oo.copy. *)
let ob1 = new some_class
(* later on *)
let ob2 = Oo.copy ob1

(* Objects can also be cloned using the functional update syntax. *)

```

```

class person (name : string) (age : int) = object
  val name = name
  val age = age
  method name = name
  method age = age
  method with_name name' = {< name = name' >}
  method with_age age' = {< age = age' >}
  method copy () = {< >}
end

```

### 13.7 Calling Methods Indirectly

```

(* Create a hashtable mapping method names to method calls. *)
let methods = Hashtbl.create 3
let () =
  Hashtbl.replace methods "run" (fun obj -> obj#run ());
  Hashtbl.replace methods "start" (fun obj -> obj#start ());
  Hashtbl.replace methods "stop" (fun obj -> obj#stop ())

(* Call the three methods on the object by name. *)
let () =
  List.iter
    (fun m -> (Hashtbl.find methods m) obj)
    ["start"; "run"; "stop"]

(* You can alias a method as long as it takes at least one argument. *)
let () =
  let meth = obj#run in
    (* ... *)
  meth ()

```

### 13.8 Determining Subclass Membership

```

(* OCaml has no runtime type information and therefore no "instanceof"
operator. One alternative would be to provide methods to query for
an object's class. *)
class widget (name : string) = object
  method name = name
  method is_widget = true
  method is_gadget = false
end
class gadget name = object
  inherit widget name
  method is_gadget = true
end

(* Another solution would be to use the visitor pattern. *)
class widget (name : string) = object (self)
  method name = name
  method accept (v : visitor) = v#visit_widget (self :> widget)
end
and gadget name = object (self)
  inherit widget name

```

```

    method accept (v : visitor) = v#visit_gadget (self := gadget)
end
and visitor ~visit_widget ~visit_gadget = object
  method visit_widget = (visit_widget : widget -> unit)
  method visit_gadget = (visit_gadget : gadget -> unit)
end
let () =
  let visitor = new visitor
    ~visit_gadget: (fun gadget ->
      Printf.printf "Found gadget: %s\n" gadget#name)
    ~visit_widget: (fun widget ->
      Printf.printf "Found widget: %s\n" widget#name) in
  List.iter
    (fun obj -> obj#accept visitor)
    [new widget "a"; new gadget "b"; new widget "c"]

(* Yet another solution would be to rethink your design in terms of
   variants and pattern matching. *)

```

### 13.9 Writing an Inheritable Class

```

class person = object (self)
  val mutable name = ""
  val mutable age = 0
  method name = name
  method age = age
  method set_name name' = name <- name'
  method set_age age' = age <- age'
end

(*-----*)

let () =
  let dude = new person in
  dude#set_name "Jason";
  dude#set_age 23;
  Printf.printf "%s is age %d\n" dude#name dude#age

(*-----*)

class employee = object (self)
  inherit person
end

(*-----*)

let () =
  let empl = new employee in
  empl#set_name "Jason";
  empl#set_age 23;
  Printf.printf "%s is age %d\n" empl#name empl#age

```

### 13.10 Accessing Overridden Methods

```
class person (name : string) (age : int) = object
  val mutable name = name
  val mutable age = age
  method name = name
  method age = age
  method set_name name' = name <- name'
  method set_age age' = age <- age'
end

class liar name age = object
  (* Call superclass constructor and alias superclass as "super". *)
  inherit person name age as super
  (* Call overridden "age" method. *)
  method age = super#age - 10
end
```

### 13.11 Generating Attribute Methods Using AUTOLOAD

```
(* Use Jacques Garrigue's pa_oo syntax extension, available at:
   http://www.math.nagoya-u.ac.jp/~garrigue/code/ocaml.html *)

(*pp camlp4o pa_oo.cmo *)
class person () = object (self)
  val mutable name = "" with accessor
  val mutable age = 0 with accessor
  val mutable parent = None with reader
  method spawn () = {< parent = Some self >}
end

let () =
  let dad = new person () in
  dad#name <- "Jason";
  dad#age <- 23;
  let kid = dad#spawn () in
  kid#name <- "Rachel";
  kid#age <- 2;
  Printf.printf "Kid's parent is %s\n"
    (match kid#parent with
     | Some parent -> parent#name
     | None -> "unknown")
```

### 13.12 Solving the Data Inheritance Problem

```
(* Use prefixes in instance variable names so we can tell them apart. *)
class person () = object
  val mutable person_age = 0
  method age = person_age
  method set_age age' = person_age <- age'
end

(* Now we can access both instance variables as needed. *)
```

```

class employee () = object
  inherit person ()
  val mutable employee_age = 0
  method age = employee_age
  method set_age age' = employee_age <- age'
  method person_age = person_age
  method set_person_age age' = person_age <- age'
end

```

### 13.13 Coping with Circular Data Structures

(\* OCaml features a generational garbage collector that can handle circular references, so you do not need to do anything special to safely dispose of circular data structures. The "DESTROY" method has been omitted from this translation since it is unnecessary.

Option types are used heavily due to the imperative style of the original recipe, which makes this code somewhat verbose. \*)

```

(* A polymorphic, circular data structure. *)
class ['a] ring () = object (self)
  val mutable dummy = (None : 'a ring_node option)
  val mutable count = 0

  (* Initialize dummy now that a reference to self is available. *)
  initializer
    (let node = new ring_node () in
     node#set_prev (Some node);
     node#set_next (Some node);
     dummy <- Some node)

  (* Return the number of values in the ring. *)
  method count = count

  (* Insert a value into the ring structure. *)
  method insert value =
    let node = new ring_node () in
    node#set_value (Some value);
    (match dummy with
     | Some ring_dummy ->
       node#set_next ring_dummy#next;
       (match ring_dummy#next with
        | Some ring_dummy_next ->
          ring_dummy_next#set_prev (Some node)
        | None -> assert false);
       ring_dummy#set_next (Some node);
       node#set_prev (Some ring_dummy);
       count <- count + 1
     | None -> assert false)

  (* Find a value in the ring. *)
  method search value =
    match dummy with

```

```

    | Some ring_dummy ->
      (match ring_dummy#next with
      | Some ring_dummy_next ->
        let node = ref ring_dummy_next in
        while !node != ring_dummy && !node#value <> (Some value)
        do node :=
          match !node#next with
          | Some n -> n
          | None -> assert false
        done;
        !node
      | None -> assert false)
    | None -> assert false

(* Delete a node from the ring structure. *)
method delete_node node =
  (match node#prev with
  | Some node_prev -> node_prev#set_next node#next
  | None -> assert false);
  (match node#next with
  | Some node_next -> node_next#set_prev node#prev
  | None -> assert false);
  count <- count - 1

(* Delete a node from the ring structure by value. *)
method delete_value value =
  let node = self#search value in
  match dummy with
  | Some ring_dummy when node != ring_dummy ->
    self#delete_node node
  | _ -> ()
end

(* A node in the ring structure which contains a polymorphic value. *)
and ['a] ring_node () = object
  val mutable prev = (None : 'a ring_node option)
  val mutable next = (None : 'a ring_node option)
  val mutable value = (None : 'a option)
  method prev = prev
  method next = next
  method value = value
  method set_prev prev' = prev <- prev'
  method set_next next' = next <- next'
  method set_value value' = value <- value'
end

```

### 13.14 Overloading Operators

```

(* Create a class with "compare_to" and "to_string" methods. *)
class class name idnum = object (self)
  val name = (name : string)
  val idnum = (idnum : int)

```

```

method name = name
method idnum = idnum

method compare_to (other : klass) =
  compare
    (String.uppercase self#name)
    (String.uppercase other#name)

method to_string =
  Printf.sprintf "%s (%05d)"
    (String.capitalize self#name)
    self#idnum
end

(* Define a comparison operator that invokes a "compare_to" method. *)
let ( <=> ) o1 o2 = (o1 #compare_to o2 : int)

(* Demonstrate these two methods. *)
let () =
  let a = new klass "test1" 5 in
  let b = new klass "TEST2" 10 in
  Printf.printf "%d\n" (a <=> b);
  Printf.printf "%s\n%s\n" a#to_string b#to_string

(* Define a module to contain our time type. *)
module TimeNumber = struct

  (* TimeNumber.t contains the time values. *)
  class t hours minutes seconds = object (self)
    val mutable hours = (hours : int)
    val mutable minutes = (minutes : int)
    val mutable seconds = (seconds : int)

    method hours = hours
    method minutes = minutes
    method seconds = seconds

    method set_hours hours' = hours <- hours'
    method set_minutes minutes' = minutes <- minutes'
    method set_seconds seconds' = seconds <- seconds'

    (* TimeNumber.t#add adds two times together. *)
    method add (other : t) =
      let answer = new t
        (self#hours + other#hours)
        (self#minutes + other#minutes)
        (self#seconds + other#seconds) in
      if answer#seconds >= 60
      then (answer#set_seconds (answer#seconds mod 60);
            answer#set_minutes (answer#minutes + 1));
      if answer#minutes >= 60
      then (answer#set_minutes (answer#minutes mod 60);
            answer#set_hours (answer#hours + 1));

```

```

    answer
end

(* TimeNumber.Operators is a submodule that is designed to be
   imported using "open". It redefines the built-in arithmetic
   operators to work on TimeNumber.t values. *)
module Operators = struct
  let ( + ) (t1 : t) (t2 : t) = t1 #add t2
  (* let ( - ) (t1 : t) (t2 : t) = t1 #sub t2 *)
  (* let ( * ) (t1 : t) (t2 : t) = t1 #mult t2 *)
  (* let ( / ) (t1 : t) (t2 : t) = t1 #div t2 *)
end

end

(* Globally import the custom operators. This will make them work on
   TimeNumber.t values *only* - to do regular integer addition, you
   will now have to use Pervasives.( + ) and so on. *)
open TimeNumber.Operators
let () =
  let t1 = new TimeNumber.t 2 59 59 in
  let t2 = new TimeNumber.t 1 5 6 in
  let t3 = t1 + t2 in
  Printf.printf "%02d:%02d:%02d\n" t3#hours t3#minutes t3#seconds

(* Locally import the custom operators using a "let module". The
   operators will only be redefined within the "Local" module. *)
let () =
  let t1 = new TimeNumber.t 2 59 59 in
  let t2 = new TimeNumber.t 1 5 6 in
  let t3 =
    let module Local = struct
      open TimeNumber.Operators
      let result = t1 + t2
    end in Local.result in
  Printf.printf "%02d:%02d:%02d\n" t3#hours t3#minutes t3#seconds

(* The openin syntax extension can simplify the above technique.
   openin is available at http://alain.frisch.fr/soft.html#openin *)
let () =
  let t1 = new TimeNumber.t 2 59 59 in
  let t2 = new TimeNumber.t 1 5 6 in
  let t3 = open TimeNumber.Operators in t1 + t2 in
  Printf.printf "%02d:%02d:%02d\n" t3#hours t3#minutes t3#seconds

(*-----*)

(* show_strnum - demo operator overloading *)

class strnum value = object
  method value = value
  method spaceship (other : strnum) = compare value (other#value)
  method concat (other : strnum) = new strnum (value ^ other#value)

```

```

method repeat n = new strnum (String.concat ""
                                (Array.to_list (Array.make n value)))
end

let ( + ) a b = a #concat b
let ( * ) a b = a #repeat b
let ( <=> ) a b = a #spaceship b
let ( < ) a b = a <=> b < 0
let ( <= ) a b = a <=> b <= 0
let ( = ) a b = a <=> b = 0
let ( >= ) a b = a <=> b >= 0
let ( > ) a b = a <=> b > 0

(*-----*)

let x = new strnum "Red"
let y = new strnum "Black"
let z = x + y
let r = z * 3

let () =
  Printf.printf "values are %s, %s, %s, and %s\n"
    x#value y#value z#value r#value;
  Printf.printf "%s is %s %s\n"
    x#value (if x < y then "LT" else "GE") y#value

(*
  values are Red, Black, RedBlack, and RedBlackRedBlackRedBlack
  Red is GE Black
*)

(*-----*)

(* demo_fixnum - show operator overloading *)

module FixNum = struct
  let default_places = ref 0
  class t ?places (value : float) = object
    val mutable places =
      match places with
      | Some n -> n
      | None -> !default_places
    val value = value
    method places = places
    method set_places n = places <- n
    method value = value
    method to_string = Printf.sprintf "FixNum.t: %.*f" places value
  end
end

let ( + ) a b =
  new FixNum.t ~places:(max a#places b#places) (a#value +. b#value)
let ( - ) a b =

```

```

    new FixNum.t ~places:(max a#places b#places) (a#value -. b#value)
let ( * ) a b =
    new FixNum.t ~places:(max a#places b#places) (a#value *. b#value)
let ( / ) a b =
    new FixNum.t ~places:(max a#places b#places) (a#value /. b#value)

(*-----*)

(* let () = FixNum.default_places := 5 *)

let x = new FixNum.t 40.
let y = new FixNum.t 12.

let () =
  Printf.printf "sum of %s and %s is %s\n"
    x#to_string y#to_string (x + y)#to_string;
  Printf.printf "product of %s and %s is %s\n"
    x#to_string y#to_string (x * y)#to_string

let z = x / y

let () =
  Printf.printf "%s has %d places\n" z#to_string z#places;
  if z#places = 0 then z#set_places 2;
  Printf.printf "div of %s by %s is %s\n"
    x#to_string y#to_string z#to_string;
  Printf.printf "square of that is %s\n" (z * z)#to_string

(*
sum of FixNum.t: 40 and FixNum.t: 12 is FixNum.t: 52
product of FixNum.t: 40 and FixNum.t: 12 is FixNum.t: 480
FixNum.t: 3 has 0 places
div of FixNum.t: 40 by FixNum.t: 12 is FixNum.t: 3.33
square of that is FixNum.t: 11.11
*)

```

### 13.15 Creating Magic Variables with tie

```

(* OCaml does not have anything like Perl's "tie" feature; you can't
make an identifier evaluate to anything other than itself. Since
"tie" is just syntax sugar anyway, all of the examples can be done
with regular classes and objects. *)

```

```

class ['a] value_ring values = object
  val mutable values = (values : 'a list)
  method get =
    match values with
    | h :: t -> values <- t @ [h]; h
    | [] -> raise Not_found
  method add value =
    values <- value :: values
end

```

```

(*-----*)

let () =
  let colors = new value_ring ["red"; "blue"] in
  Printf.printf "%s %s %s %s %s %s\n"
    colors#get colors#get colors#get
    colors#get colors#get colors#get;
  (* blue red blue red blue red *)

  colors#add "green";
  Printf.printf "%s %s %s %s %s %s\n"
    colors#get colors#get colors#get
    colors#get colors#get colors#get
  (* blue red green blue red green *)

(*-----*)

(* Magic hash that autoappends. *)

class ['a, 'b] append_hash size = object
  val hash = (Hashtbl.create size : ('a, 'b) Hashtbl.t)
  method get k = Hashtbl.find hash k
  method set k v =
    Hashtbl.replace hash k
      (try v :: Hashtbl.find hash k with Not_found -> [v])
  method each f = Hashtbl.iter f hash
end

(*-----*)

let () =
  let tab = new append_hash 3 in
  tab#set "beer" "guinness";
  tab#set "food" "potatoes";
  tab#set "food" "peas";
  tab#each
    (fun k vs ->
      Printf.printf "%s => [%s]\n"
        k (String.concat " " (List.rev vs)))

(*-----*)

(*
  beer => [guinness]
  food => [potatoes peas]
*)

(*-----*)

(* For a more lightweight syntax, you can override the .{}
  operator--which normally works with Bigarrays--to work on
  any object with "get" and "set" methods. *)

```

```

module Bigarray = struct
  module Array1 = struct
    let get obj = obj#get
    let set obj = obj#set
  end
end

let () =
  let tab = new append_hash 3 in
  tab>{"beer"} <- "guinness";
  tab>{"food"} <- "potatoes";
  tab>{"food"} <- "peas";
  tab#each
    (fun k vs ->
      Printf.printf "%s => [%s]\n"
        k (String.concat " " (List.rev vs)));
  print_endline (List.hd tab>{"beer"})

(*-----*)

(* Hash that magically folds case. *)

class ['a] folded_hash size = object
  val hash = (Hashtbl.create size : (string, 'a) Hashtbl.t)
  method get k = Hashtbl.find hash (String.lowercase k)
  method set k v = Hashtbl.replace hash (String.lowercase k) v
  method each f = Hashtbl.iter f hash
end

(*-----*)

let () =
  let tab = new folded_hash 2 in
  tab>{"VILLAIN"} <- "big ";
  tab>{"herOine"} <- "red riding hood";
  tab>{"villain"} <- tab>{"villain"} ^ "bad wolf";
  tab#each (Printf.printf "%s is %s\n")

(*
  heroine is red riding hood
  villain is big bad wolf
*)

(*-----*)

(* Hash that permits key *or* value lookups. *)
class ['a] rev_hash size = object
  val hash = (Hashtbl.create size : ('a, 'a) Hashtbl.t)
  method get k = Hashtbl.find hash k
  method set k v =
    Hashtbl.replace hash k v;
    Hashtbl.replace hash v k
  method each f = Hashtbl.iter f hash
end

```

```

end

(*-----*)

let () =
  let tab = new rev_hash 8 in
  tab.{`Str "Red"} <- `Str "Rojo";
  tab.{`Str "Blue"} <- `Str "Azul";
  tab.{`Str "Green"} <- `Str "Verde";
  tab.{`Str "EVIL"} <- `StrList [ "No way!"; "Way!!" ];
  let to_string = function
    | `Str s -> s
    | `StrList ss -> "[" ^ String.concat " " ss ^ "]" in
  tab#each
    (fun k v ->
      Printf.printf "%s => %s\n" (to_string k) (to_string v))

(*
  Verde => Green
  Azul => Blue
  Green => Verde
  Blue => Azul
  Red => Rojo
  [No way! Way!!] => EVIL
  EVIL => [No way! Way!!]
  Rojo => Red
*)

(*-----*)

(* Simple counter. *)

class counter start = object
  val mutable value = (start : int)
  method next = value <- value + 1; value
end

let () =
  let c = new counter 0 in
  while true do
    Printf.printf "Got %d\n" c#next
  done

(*-----*)

(* Tee-like class that outputs to multiple channels at once. *)

class tee channels = object
  method print s = List.iter (fun ch -> output_string ch s) channels
end

let () =
  let tee = new tee [stdout; stderr] in

```

```
tee#print "This line goes to both places.\n";
flush_all ()

let () =
  let tee = new tee
    (stdout ::
      (Array.to_list
        (Array.init 10
          (fun _ ->
            snd (Filename.open_temp_file "teetest." "")))))) in
  tee#print "This lines goes many places.\n";
  flush_all ()
```

## 14 Database Access

```
(* OCaml's standard library includes bindings to the NDBM database.
   Bindings to other database systems can be found on the web. *)
```

```
MySQL:
http://raevnos.pennmush.org/code/ocaml-mysql/index.html
```

```
PostgreSQL:
http://www.ocaml.info/home/ocaml_sources.html#postgresql-ocaml
```

```
SQLite:
http://www.ocaml.info/home/ocaml_sources.html#ocaml-sqlite3
```

### 14.1 Making and Using a DBM File

```
#load "dbm.cma";;

(* open database *)
let db = Dbm.opendbm filename [Dbm.Dbm_rdwr; Dbm.Dbm_create] 0o666

(* retrieve from database *)
let v = Dbm.find db key

(* put value into database *)
let () = Dbm.replace db key value

(* check whether in database *)
let () =
  try
    ignore (Dbm.find db key);
    (* ... *)
    ()
  with Not_found ->
    (* ... *)
    ()

(* delete from database *)
let () = Dbm.remove db key

(* close the database *)
let () = Dbm.close db

(*-----*)

(* userstats - generates statistics on who is logged in. *)
(* call with an argument to display totals *)

#load "dbm.cma";;
#load "str.cma";;
#load "unix.cma";;

let db_file = "/tmp/userstats.db" (* where data is kept between runs *)
```

```

let db = Dbm.opendbm db_file [Dbm.Dbm_rdwr; Dbm.Dbm_create] 0o666

let () =
  if Array.length Sys.argv > 1
  then
    begin
      let sort a = Array.sort compare a; a in
      let keys db = Array.of_list
        (let accu = ref [] in
         Dbm.iter (fun key _ -> accu := key :: !accu) db;
          !accu) in
      let users = Array.sub Sys.argv 1 (Array.length Sys.argv - 1) in
      let users = if users = ["ALL"] then sort (keys db) else users in
      Array.iter
        (fun user ->
         Printf.printf "%s\t%s\n"
           user (try Dbm.find db user with Not_found -> ""))
        users
    end
  else
    begin
      let who = Unix.open_process_in "who" in
      let regexp = Str.regexp "[ \t]+" in
      try
        while true do
          (* extract username (first thing on the line) and update *)
          let line = input_line who in
          let user = List.hd (Str.split_delim regexp line) in
          let count =
            try int_of_string (Dbm.find db user)
              with Not_found -> 0 in
            Dbm.replace db user (string_of_int (count + 1))
          done
        with End_of_file ->
          ignore (Unix.close_process_in who)
      end
    end

let () = Dbm.close db

```

## 14.2 Emptying a DBM File

```

let () =
  let db = Dbm.opendbm filename [Dbm.Dbm_rdwr; Dbm.Dbm_create] 0o666 in
  let keys = ref [] in
  Dbm.iter (fun key _ -> keys := key :: !keys) db;
  List.iter (Dbm.remove db) !keys;
  Dbm.close db

(*-----*)

let () =
  Sys.remove filename;
  ignore (Dbm.opendbm filename [Dbm.Dbm_rdwr; Dbm.Dbm_create] 0o666)

```

### 14.3 Converting Between DBM Files

```
(* OCaml does not come with support for any DBM-style databases other
   than NDBM, and no third-party libraries appear to be available. *)
```

### 14.4 Merging DBM Files

```
let () = Dbm.iter (Dbm.replace output) input

(*-----*)

let () =
  Dbm.iter
    (fun key value ->
      try
        let existing = Dbm.find output key value in
          (* decide which value to use and replace if necessary *)
          ()
        with Not_found ->
          Dbm.replace output key value)
    input
```

### 14.5 Locking DBM Files

```
(* dblockdemo - demo locking dbm databases *)
(* Thanks to Janne Hellsten for posting sample code on caml-list! *)

#load "dbm.cma";;
#load "unix.cma";;

let db_file = "/tmp/foo.db"
let lock_file = "/tmp/foo.lock"

let key = try Sys.argv.(1) with Invalid_argument _ -> "default"
let value = try Sys.argv.(2) with Invalid_argument _ -> "magic"
let value = value ^ " " ^ (string_of_int (Unix.getpid ()))

let finally handler f x =
  let result = try f x with e -> handler (); raise e in handler (); result

let create_lock name =
  if not (Sys.file_exists name) then
    let out_channel = open_out name in close_out out_channel

let with_lock name command f =
  create_lock name;
  let fd = Unix.openfile name [Unix.O_RDWR] 0o660 in
  finally
    (fun () -> Unix.close fd)
    (fun () -> Unix.lockf fd command 0; f ()) ()

let create_db name =
  if not (Sys.file_exists (name ^ ".dir")) then
```

```

    let db = Dbm.opendbm name [Dbm.Dbm_rdwr; Dbm.Dbm_create] 0o660 in
    Dbm.close db

let () =
  create_db db_file;

  let do_read () =
    let db = Dbm.opendbm db_file [Dbm.Dbm_rdonly] 0o660 in
    Printf.printf "%d: Read lock granted\n" (Unix.getpid ());
    flush stdout;
    let oldval = try Dbm.find db key with Not_found -> "" in
    Printf.printf "%d: Old value was %s\n" (Unix.getpid ()) oldval;
    flush stdout;
    Dbm.close db in

  let do_write () =
    let db = Dbm.opendbm db_file [Dbm.Dbm_rdwr] 0o660 in
    Printf.printf "%d: Write lock granted\n" (Unix.getpid ());
    flush stdout;
    Dbm.replace db key value;
    Unix.sleep 10;
    Dbm.close db in

  begin
    try
      with_lock lock_file Unix.F_TRLOCK do_read;
    with Unix.Unix_error (error, "lockf", _) ->
      Printf.printf "%d: CONTENTION; can't read during write update! \
                    Waiting for read lock (%s) ... \n"
        (Unix.getpid ()) (Unix.error_message error);
      flush stdout;
      with_lock lock_file Unix.F_RLOCK do_read
    end;

  begin
    try
      with_lock lock_file Unix.F_TLOCK do_write;
    with Unix.Unix_error (error, "lockf", _) ->
      Printf.printf "%d: CONTENTION; must have exclusive lock! \
                    Waiting for write lock (%s) ... \n"
        (Unix.getpid ()) (Unix.error_message error);
      flush stdout;
      with_lock lock_file Unix.F_LOCK do_write
    end;

  Printf.printf "%d: Updated db to %s=%s\n" (Unix.getpid ()) key value

```

## 14.6 Sorting Large DBM Files

(\* OCaml's Dbm module does not provide any mechanism for a custom comparison function. If you need the keys in a particular order you can load them into memory and use List.sort, Array.sort, or a Set. This may not be practical for very large data sets. \*)

## 14.7 Treating a Text File as a Database Array

```
let with_lines_in_file name f =
  if not (Sys.file_exists name)
  then (let out_channel = open_out name in close_out out_channel);

  let in_channel = open_in name in
  let in_lines = ref [] in
  begin
    try
      while true do
        in_lines := input_line in_channel :: !in_lines
      done
    with End_of_file ->
      close_in in_channel
  end;

  let out_lines = f (List.rev !in_lines) in
  let out_channel = open_out name in
  List.iter
    (fun line ->
      output_string out_channel line;
      output_string out_channel "\n")
    out_lines;
  flush out_channel;
  close_out out_channel

let () =
  (* first create a text file to play with *)
  with_lines_in_file "/tmp/textfile"
    (fun lines ->
      ["zero"; "one"; "two"; "three"; "four"]);

  with_lines_in_file "/tmp/textfile"
    (fun lines ->
      (* print the records in order. *)
      print_endline "ORIGINAL\n";
      Array.iteri (Printf.printf "%d: %s\n") (Array.of_list lines);

      (* operate on the end of the list *)
      let lines = List.rev lines in
      let a = List.hd lines in
      let lines = List.rev ("last" :: lines) in
      Printf.printf "\nThe last record was [%s]\n" a;

      (* and the beginning of the list *)
      let a = List.hd lines in
      let lines = "first" :: (List.tl lines) in
      Printf.printf "\nThe first record was [%s]\n" a;

      (* remove the record "four" *)
      let lines =
        List.filter (function "four" -> false | _ -> true) lines in
```

```

(* replace the record "two" with "Newbie" *)
let lines =
  List.map (function "two" -> "Newbie" | x -> x) lines in

(* add a new record after "first" *)
let lines =
  List.fold_right
    (fun x a ->
      if x = "first"
      then x :: "New One" :: a
      else x :: a)
    lines [] in

(* now print the records in reverse order *)
print_endline "\nREVERSE\n";
List.iter print_string
  (List.rev
    (Array.to_list
      (Array.mapi
        (fun i line -> Printf.sprintf "%d: %s\n" i line)
        (Array.of_list lines))));

(* return the new list, which will be written back to the file *)
lines)

(*-----*)
ORIGINAL

0: zero
1: one
2: two
3: three
4: four

The last record was [four]

The first record was [zero]

REVERSE

5: last
4: three
3: Newbie
2: one
1: New One
0: first
-----*)

```

## 14.8 Storing Complex Data in a DBM File

```

(* OCaml includes a Marshal module which does binary serialization and
   deserialization of arbitrary data structures. However, it is not

```

type-safe, so coding errors can result in segmentation faults.

One way to eliminate this risk is to use functors. The following example builds a functor called "MakeSerializedDbm" which extends the Dbm module to provide type-safe serialization of values using a user-defined method such as (but not limited to) Marshal. \*)

```
#load "dbm.cma";;

(* This module type defines a serialization method. It contains a type
   and functions to convert values of that type to and from strings. *)
module type SerializedDbmMethod =
sig
  type value
  val serialize : value -> string
  val deserialize : string -> value
end

(* This module type defines an enhanced Dbm interface that includes a
   type for values to be used instead of strings. *)
module type SerializedDbm =
sig
  type t
  type value
  val opendir : string -> Dbm.open_flag list -> int -> t
  val close : t -> unit
  val find : t -> string -> value
  val add : t -> string -> value -> unit
  val replace : t -> string -> value -> unit
  val remove : t -> string -> unit
  val firstkey : t -> string
  val nextkey : t -> string
  val iter : (string -> value -> 'a) -> t -> unit
end

(* Here is the functor itself. It takes a SerializedDbmMethod as an
   argument and returns a SerializedDbm module instance as a result.
   It is defined mainly in terms of Dbm, with a few overridden
   definitions where the value type is needed. *)
module MakeSerializedDbm (Method : SerializedDbmMethod)
  : SerializedDbm with type value = Method.value =
struct
  include Dbm
  type value = Method.value
  let find db key = Method.deserialize (find db key)
  let add db key value = add db key (Method.serialize value)
  let replace db key value = replace db key (Method.serialize value)
  let iter f db = iter (fun key value -
> f key (Method.deserialize value)) db
end

(* Now, we can easily build typed Dbm interfaces by providing the type
   and conversion functions. In this case, we use Marshal, but we could
```

```

    also use other string-based serialization formats like JSON or XML. *)
module StringListDbm =
  MakeSerializedDbm(struct
    type value = string list
    let serialize x = Marshal.to_string x []
    let deserialize x = Marshal.from_string x 0
  end)

let db = StringList-
Dbm.opendbm "data.db" [Dbm.Dbm_rdwr; Dbm.Dbm_create] 0o666
let () =
  StringListDbm.replace db "Tom Christiansen"
  [ "book author"; "tchrist@perl.com" ];
  StringListDbm.replace db "Tom Boutell"
  [ "shareware author"; "boutell@boutell.com" ];

(* names to compare *)
let name1 = "Tom Christiansen" in
let name2 = "Tom Boutell" in

let tom1 = StringListDbm.find db name1 in
let tom2 = StringListDbm.find db name2 in

let show strings =
  "[" ^ (String.concat "; "
    (List.map (fun s -> "\"" ^ s ^ "\"") strings)) ^ "]" in
Printf.printf "Two Toming: %s %s\n" (show tom1) (show tom2)

```

## 14.9 Persistent Data

```

type data = {mutable variable1: string; mutable variable2: string}

module PersistentStore =
  MakeSerializedDbm(struct
    type value = data
    let serialize x = Marshal.to_string x []
    let deserialize x = Marshal.from_string x 0
  end)

let with_persistent_data f =
  let db =
    Persis-
tentStore.opendbm "data.db" [Dbm.Dbm_rdwr; Dbm.Dbm_create] 0o666 in
  let data =
    try PersistentStore.find db "data"
    with Not_found -> {variable1=""; variable2=""} in
  f data;
  PersistentStore.replace db "data" data
  PersistentStore.close db

let () =
  with_persistent_data
  (fun data ->

```

```

begin
  Printf.printf "variable1 = %s\nvariable2 = %s\n"
    data.variable1 data.variable2;
  data.variable1 <- "foo";
  data.variable2 <- "bar";
end)

```

## 14.10 Executing an SQL Command Using DBI and DBD

(\* This example uses OCamL DBI, a component of the mod\_caml web development library that provides a database abstraction API very similar to that of Perl DBI. It is available for download here:

[http://merjis.com/developers/mod\\_caml](http://merjis.com/developers/mod_caml)

Drivers for particular databases are listed in the introduction. \*)

```

#load "nums.cma";;
#directory "+num-top";;
#load "num_top.cma";;

#directory "+mysql";;
#load "mysql.cma";;

#directory "+dbi";;
#load "dbi.cma";;
#load "dbi_mysql.cmo";;

```

(\* With dbi installed via findlib, the above can be shortened to:

```

#use "topfind";;
#require "dbi.mysql";;
*)

let () =
  let dbh =
    Dbi_mysql.connect
      ~user:"user"
      ~password:"auth"
      "database" in

    let _ = dbh#ex sql [] in

    let sth = dbh#prepare sql in
      sth#execute [];
      sth#iter
        (fun row ->
          print_endline (Dbi.sdebug row);
          (* ... *)
          ());

      sth#finish ();
      dbh#close ()

```

```

(*-----*)

(* dbusers - manage MySQL user table *)

(* This example uses the Mysql module directly rather than going through
OCaml DBI. See the introduction for a link to the Mysql library. *)

#load "unix.cma";;
#directory "+mysql";;
#load "mysql.cma";;

let () =
  let db =
    Mysql.quick_connect
      ~user:"user"
      ~password:"password"
      ~database:"dbname" () in

    ignore (Mysql.exec db "CREATE TABLE users (uid INT, login CHAR(8))");

    let passwd = open_in "/etc/passwd" in
      begin
        try
          while true do
            let line = input_line passwd in
              let user = String.sub line 0 (String.index line ':' ) in
                let {Unix.pw_uid=uid; pw_name=name} = Unix.getpwnam user in
                  let sql =
                    Printf.sprintf "INSERT INTO users VALUES( %s, %s )"
                      (Mysql.ml2int uid)
                      (Mysql.ml2str name) in
                    ignore (Mysql.exec db sql)
              done
            with End_of_file ->
              close_in passwd
          end;

          ignore (Mysql.exec db "DROP TABLE users");

          Mysql.disconnect db

```

### 14.11 Program: ggh - Grep Netscape Global History

```

(* Search the history using the Places SQLite database, new in Firefox 3.
Pattern-matching uses simple substrings, but it could be expanded to use
Str or Pcre by installing a user-defined function. *)

```

```

#directory "+sqlite3";;
#load "sqlite3.cma";;
#load "unix.cma";;

type history = { visit_date : Unix.tm;

```

```

        url      : string;
        title    : string; }

let days = [| "Sun"; "Mon"; "Tue"; "Wed"; "Thu"; "Fri"; "Sat" |]
let months = [| "Jan"; "Feb"; "Mar"; "Apr"; "May"; "Jun";
               "Jul"; "Aug"; "Sep"; "Oct"; "Nov"; "Dec" |]

let string_of_tm tm =
  Printf.sprintf "%s %s %2d %02d:%02d:%02d %04d"
    days.(tm.Unix.tm_wday)
    months.(tm.Unix.tm_mon)
    tm.Unix.tm_mday
    tm.Unix.tm_hour
    tm.Unix.tm_min
    tm.Unix.tm_sec
    (tm.Unix.tm_year + 1900)

let tm_of_micros micros =
  let time = float_of_string micros /. 1000000. in
  Unix.localtime time

let () =
  if Array.length Sys.argv < 2 then
    begin
      Printf.printf "Usage: %s path/to/places.sqlite [pattern]\n"
        Sys.argv.(0);
      exit 0
    end

let file =
  if Array.length Sys.argv > 1 then Sys.argv.(1) else "places.sqlite"

let pattern =
  if Array.length Sys.argv > 2 then Some Sys.argv.(2) else None

let db = Sqlite3.db_open file

let sql =
  Printf.sprintf
    "SELECT  visit_date, url, title
     FROM    moz_places p
     JOIN    moz_historyvisits v
     ON      p.id = v.place_id
     %s
     ORDER BY visit_date DESC"
    (match pattern with
     | None -> ""
     | Some s ->
       (Printf.sprintf "WHERE url LIKE '%s'" OR title LIKE '%s'"
        s s))

let data = ref []

```

```

let res =
  Sqlite3.exec_not_null_no_headers db
  ~cb:(fun row ->
    data := {visit_date = tm_of_micros row.(0);
             url = row.(1);
             title = row.(2)} :: !data) sql

let () =
  match res with
  | Sqlite3.Rc.OK ->
    List.iter
      (fun history ->
        Printf.printf "[%s] %s \"%s\"\n"
          (string_of_tm history.visit_date)
          history.url
          history.title)
      !data
  | r ->
    Printf.eprintf "%s: %s\n"
      (Sqlite3.Rc.to_string r)
      (Sqlite3.errmsg db)

```

## 15 User Interfaces

### 15.1 Parsing Program Arguments

```
let verbose = ref false
let debug = ref false
let output = ref ""

let () =
  Arg.parse
  [
    "-v", Arg.Set verbose, "Verbose mode";
    "-D", Arg.Set debug, "Debug mode";
    "-o", Arg.Set_string output, "Specify output file";
  ]
  (fun s ->
    raise (Arg.Bad (Printf.sprintf "unexpected argument '%s'" s))
    (Printf.sprintf "Usage: %s [-v] [-d] [-o file]" Sys.argv.(0)))

let () =
  if !verbose then print_endline "Verbose mode";
  if !debug then print_endline "Debug mode";
  if !output <> "" then print_endline ("Writing output to " ^ !output);
```

### 15.2 Testing Whether a Program Is Running Interactively

```
#load "unix.cma";;

let i_am_interactive () =
  Unix.isatty Unix.stdin && Unix.isatty Unix.stdout

let () =
  try
    while true do
      if i_am_interactive ()
      then print_string "Prompt: ";
      let line = read_line () in
      if line = "" then raise End_of_file;
      (* do something with the line *)
    done
  with End_of_file -> ()
```

### 15.3 Clearing the Screen

```
#load "unix.cma";;

(* Run the clear command to clear the screen. *)
let () = ignore (Sys.command "clear")

(* Save the output to a string to avoid running a process each time. *)
let clear =
  try
    let proc = Unix.open_process_in "clear" in
```

```

try
  let chars = input_line proc in
  ignore (Unix.close_process_in proc);
  chars
with e -> ignore (Unix.close_process_in proc); ""
with _ -> ""
let () = print_string clear

```

## 15.4 Determining Terminal or Window Size

```

#load "unix.cma";;

(* UNIX only, due to "stty". *)
let get_terminal_size () =
  let in_channel = Unix.open_process_in "stty size" in
  try
    begin
      try
        Scanf.fscanf in_channel "%d %d"
          (fun rows cols ->
            ignore (Unix.close_process_in in_channel);
            (rows, cols))
        with End_of_file ->
          ignore (Unix.close_process_in in_channel);
          (0, 0)
      end
    end
  with e ->
    ignore (Unix.close_process_in in_channel);
    raise e

(* Display a textual bar chart as wide as the console. *)
let () =
  let (height, width) = get_terminal_size () in
  if width < 10
  then (prerr_endline "You must have at least 10 characters";
        exit 255);
  let max_value = List.fold_left max 0.0 values in
  let ratio = (float width -. 10.0) /. max_value in
  List.iter
    (fun value ->
      Printf.printf "%8.1f %s\n"
        value
        (String.make (int_of_float (ratio *. value)) '*'))
    values

```

## 15.5 Changing Text Color

```

(* Requires the ANSITerminal library by Christophe Troestler,
   available at http://math.umh.ac.be/an/software.php#x4-80007 *)

#load "ANSITerminal.cma";;
open ANSITerminal

```

```

let () =
  print_string [red] "Danger Will Robinson!\n";
  print_string [] "This is just normal text.\n";
  print_string [Blink] "<BLINK>Do you hurt yet?</BLINK>\n"

(*-----*)

let () =
  set_autoreset false;
  (* rhyme for the deadly coral snake *)
  print_string [red; on_black] "venom lack\n";
  print_string [red; on_yellow] "kill that fellow\n";
  print_string [green; on_cyan; Blink] "garish!\n";
  print_string [Reset] ""

(*-----*)

let () =
  set_autoreset true;
  List.iter
    (print_string [red; on_white; Bold; Blink])
    ["This way\n";
     "each line\n";
     "has its own\n";
     "attribute set.\n"]

```

## 15.6 Reading from the Keyboard

```

#load "unix.cma";;

let with_cbreak f x =
  let term_init = Unix.tcgetattr Unix.stdin in
  let term_cbreak = { term_init with Unix.c_icanon = false } in
  Unix.tcsetattr Unix.stdin Unix.TCSANOW term_cbreak;
  try
    let result = f x in
    Unix.tcsetattr Unix.stdin Unix.TCSADRAIN term_init;
    result
  with e ->
    Unix.tcsetattr Unix.stdin Unix.TCSADRAIN term_init;
    raise e

let key = with_cbreak input_char stdin

(*-----*)

(* sascii - Show ASCII values for keypresses *)
let sascii () =
  while true do
    let char = Char.code (input_char stdin) in
    Printf.printf " Decimal: %d\tHex: %x\n" char char;
    flush stdout
  done

```

```

let () =
  print_endline
  "Press keys to see their ASCII values. Use Ctrl-C to quit.";
  with_cbreak sascii ()

```

## 15.7 Ringing the Terminal Bell

```

(* OCaml doesn't recognize '\a'; instead use '\007'. *)
let () = print_endline "\007Wake up!"

(* Use the "tput" command to produce a visual bell. *)
let () = ignore (Sys.command "tput flash")

```

## 15.8 Using POSIX termios

```

#!/usr/bin/ocaml
(* demo POSIX termios *)

#load "unix.cma";;

let uncontrol c =
  if c >= '\128' && c <= '\255'
  then Printf.sprintf "M-%c" (Char.chr (Char.code c land 127))
  else if (c >= '\000' && c < '\031') || c = '\127'
  then Printf.sprintf "^%c" (Char.chr (Char.code c lxor 64))
  else String.make 1 c

let term = Unix.tcgetattr Unix.stdin
let erase = term.Unix.c_verase
let kill = term.Unix.c_vkill

let () =
  Printf.printf "Erase is character %d, %s\n"
    (Char.code erase)
    (uncontrol erase);
  Printf.printf "Kill is character %d, %s\n"
    (Char.code kill)
    (uncontrol kill)

let () =
  term.Unix.c_verase <- '#';
  term.Unix.c_vkill <- '@';
  Unix.tcsetattr Unix.stdin Unix.TCSANOW term;
  Printf.printf "erase is #, kill is @; type something: %!";
  let line = input_line stdin in
  Printf.printf "You typed: %s\n" line;
  term.Unix.c_verase <- erase;
  term.Unix.c_vkill <- kill;
  Unix.tcsetattr Unix.stdin Unix.TCSANOW term

(*-----*)

module HotKey :

```

```

sig
  val cbreak : unit -> unit
  val cooked : unit -> unit
  val readkey : unit -> char
end =
struct
  open Unix

  let oterm = {(tcgetattr stdin) with c_vtime = 0}
  let noecho = {oterm with
    c_vtime = 1;
    c_echo = false;
    c_echok = false;
    c_icanon = false}

  let cbreak () = tcsetattr stdin TCSANOW noecho
  let cooked () = tcsetattr stdin TCSANOW oterm

  let readkey () =
    cbreak ();
    let key = input_char (Pervasives.stdin) in
    cooked ();
    key

  let () = cooked ()
end

```

## 15.9 Checking for Waiting Input

```

#load "unix.cma";;

let () =
  Unix.set_nonblock Unix.stdin;
  try
    let char = with_cbreak input_char stdin in
    (* input was waiting and it was char *)
    ()
  with Sys_blocked_io ->
    (* no input was waiting *)
    ()

```

## 15.10 Reading Passwords

```

#load "unix.cma";;

(* Thanks to David Mentre, Remi Vanicat, and David Brown's posts on
   caml-list. Works on Unix only, unfortunately, due to tcsetattr. *)
let read_password () =
  let term_init = Unix.tcgetattr Unix.stdin in
  let term_no_echo = { term_init with Unix.c_echo = false } in
  Unix.tcsetattr Unix.stdin Unix.TCSANOW term_no_echo;
  try
    let password = read_line () in

```

```

    print_newline ();
    Unix.tcsetattr Unix.stdin Unix.TCSAFLUSH term_init;
    password
with e ->
    Unix.tcsetattr Unix.stdin Unix.TCSAFLUSH term_init;
    raise e

let () =
  print_string "Enter your password: ";
  let password = read_password () in
  Printf.printf "You said: %s\n" password

```

## 15.11 Editing Input

(\* ledit is a pure-OCaml readline clone by Daniel de Rauglaudre.  
Source is available here: <http://pauillac.inria.fr/~ddr/ledit/>

It is designed to be used as a command-line wrapper, but it can also be embedded in another program by building it normally and copying `cursor.cmo`, `ledit.cmi`, `ledit.cmo`, and `ledit.mli` into your project.

A guide to compiling and embedding ledit can be found on the OCaml Tutorial Wiki: <http://www.ocaml-tutorial.org/ledit>  
At present, this guide applies to ledit 1.11. This recipe uses ledit 1.15, which is slightly different due to the addition of Unicode support (`Ledit.input_char` now returns a string instead of a char). \*)

```

#load "unix.cma";;
#load "cursor.cmo";;
#load "ledit.cmo";;

let readline prompt =
  Ledit.set_prompt prompt;
  let buffer = Buffer.create 256 in
  let rec loop = function
    | "\n" ->
      Buffer.contents buffer
    | string ->
      Buffer.add_string buffer string;
      loop (Ledit.input_char stdin) in
  loop (Ledit.input_char stdin)

let () =
  let prompt = "Prompt: " in
  let line = readline prompt in
  Printf.printf "You said: %s\n" line

(*-----*)

(* vbsh - very bad shell *)
let () =

```

```

try
  while true do
    let cmd = readline "$ " in
    begin
      match Unix.system cmd with
      | Unix.WEXITED _ -> ()
      | Unix.WSIGNALED signal_num ->
          Printf.printf "Program killed by signal %d\n"
            signal_num
      | Unix.WSTOPPED signal_num ->
          Printf.printf "Program stopped by signal %d\n"
            signal_num
    end;
    flush stdout
  done
with End_of_file -> ()

```

## 15.12 Managing the Screen

```

#!/usr/bin/ocaml
(* rep - screen repeat command *)

#load "unix.cma";;

(* http://www.nongnu.org/ocaml-tmk/ *)
#directory "+curses";;
#load "curses.cma";;

let timeout = 10.0

let (timeout, command) =
  match Array.length Sys.argv with
  | 0 | 1 -> (timeout, [| |])
  | len ->
      if Sys.argv.(1) <> "" && Sys.argv.(1).[0] = '-'
      then (float_of_string
            (String.sub Sys.argv.(1)
              1 (String.length Sys.argv.(1) - 1)),
            Array.sub Sys.argv 2 (len - 2))
      else (timeout, Array.sub Sys.argv 1 (len - 1))

let () =
  if Array.length command = 0
  then (Printf.printf "usage: %s [ -timeout ] cmd args\n" Sys.argv.(0);
        exit 255)

let window = Curses.initscr ()          (* start screen *)
let _ = Curses.noecho ()
let _ = Curses.cbreak ()
let _ = Curses.nodelay window true     (* so getch() is non-blocking *)

let done' s _ = Curses.endwin (); print_endline s; exit 0
let () = Sys.set_signal Sys.sigint (Sys.Signal_handle (done' "Ouch!"))

```

```

let cols, lines = Curses.getmaxyx window

let days = [| "Sun"; "Mon"; "Tue"; "Wed"; "Thu"; "Fri"; "Sat" |]
let months = [| "Jan"; "Feb"; "Mar"; "Apr"; "May"; "Jun";
                "Jul"; "Aug"; "Sep"; "Oct"; "Nov"; "Dec" |]

let format_time time =
  let tm = Unix.localtime time in
  Printf.sprintf "%s %s %2d %02d:%02d:%02d %04d"
    days.(tm.Unix.tm_wday)
    months.(tm.Unix.tm_mon)
    tm.Unix.tm_mday
    tm.Unix.tm_hour
    tm.Unix.tm_min
    tm.Unix.tm_sec
    (tm.Unix.tm_year + 1900)

let time = fst (Unix.mktime {Unix.tm_sec=50; tm_min=45; tm_hour=3;
                           tm_mday=18; tm_mon=0; tm_year=73;
                           tm_wday=0; tm_yday=0; tm_isdst=false})

let () =
  while true do
    let key = ref (-1) in
    while key := Curses.getch (); !key <> -1 do
      if !key = Char.code 'q' then done' "See ya" ()
      done;

    let in_channel =
      Unix.open_process_in (String.concat " " (Array.to_list command)) in
    begin
      try
        for i = 0 to lines - 1 do
          let line = input_line in_channel in
          ignore (Curses.mvaddstr i 0 line);

          Curses standout ();
          ignore (Curses.mvaddstr (lines - 1) (cols - 24)
                                (format_time (Unix.time ()))));
          Curses standend ();

          ignore (Curses.move 0 0);
          ignore (Curses.refresh ());
        done;
        ignore (Unix.close_process_in in_channel)
      with End_of_file ->
        ignore (Unix.close_process_in in_channel)
    end;

    ignore (Unix.select [Unix.stdin] [] [] timeout)
  done

```

```
(*-----*)

let err = Curses.keypad window true      (* enable keypad mode *)
let key = Curses.getch ()
let () =
  if (key = (Char.code 'k') ||           (* vi mode *)
      key = 16 ||                       (* emacs mode *)
      key = Curses.Key.up)             (* arrow mode *)
  then
    begin
      (* do something *)
    end
end
```

### 15.13 Controlling Another Program with Expect

```
(* Use perl4caml to integrate OCaml with Perl:
   http://merjis.com/developers/perl4caml *)
#directory "+perl";
#load "perl4caml.cma";

(* Wrap the needed functionality from CPAN's Expect module: *)
module Expect = struct
  open Perl
  let _ = eval "use Expect"

  exception Error of string

  type match_pattern = Ex of string | Re of string

  class expect () = object (self)
    val sv = call_class_method "Expect" "new" []

    method log_stdout =
      bool_of_sv (call_method sv "log_stdout" [])

    method set_log_stdout bool =
      ignore (call_method sv "log_stdout" [sv_of_bool bool])

    method spawn command parameters =
      let result =
          call_method sv "spawn"
            (sv_of_string command :: List.map sv_of_string parameters) in
        if not (bool_of_sv result)
        then raise (Error (string_of_sv (eval "$!")))

    method expect timeout match_patterns =
      let sv_of_pattern = function
          | Ex s -> [sv_of_string "-ex"; sv_of_string s]
          | Re s -> [sv_of_string "-re"; sv_of_string s] in
        let timeout =
            match timeout with
            | Some i -> sv_of_int i
            | None -> sv_undef () in
```

```

let result =
  call_method sv "expect"
  (timeout ::
    List.flatten (List.map sv_of_pattern match_patterns)) in
if sv_is_undef result
then None
else Some (int_of_sv result - 1)

method send string =
  ignore (call_method sv "send" [sv_of_string string])

method soft_close () =
  ignore (call_method sv "soft_close" [])

method hard_close () =
  ignore (call_method sv "hard_close" [])
end

let spawn command parameters =
  let exp = new expect () in
  exp#spawn command parameters;
  exp
end

(* start the program *)
let command =
  try Expect.spawn "program to run" ["arg 1"; "arg 2"]
  with Expect.Error e ->
    Printf.eprintf "Couldn't start program: %s\n%!" e;
    exit 1

let () =
  (* prevent the program's output from being shown on our stdout *)
  command#set_log_stdout false;

  (* wait 10 seconds for "login:" to appear *)
  if command#expect (Some 10) [Expect.Ex "login"] = None
  then failwith "timed out";

  (* wait 20 seconds for something that matches /[Pp]assword: ?/ *)
  if command#expect (Some 20) [Expect.Re "[Pp]assword: ?"] = None
  then failwith "timed out";

  (* wait forever for "invalid" to appear *)
  if command#expect None [Expect.Ex "invalid"] = None
  then failwith "error occurred; the program probably went away";

  (* send "Hello, world" and a carriage return to the program *)
  command#send "Hello, world\r";

  (* if the program will terminate by itself, finish up with *)
  command#soft_close ();

```

```

(* if the program must be explicitly killed, finish up with *)
command#hard_close ()

(* wait for multiple strings *)
let () =
  match command#expect (Some 30)
  [Expect.Ex "invalid"; Expect.Ex "succes";
   Expect.Ex "error"; Expect.Ex "boom"] with
  | Some which ->
    (* found one of those strings *)
    ()
  | None ->
    ()

```

## 15.14 Creating Menus with Tk

```

(* LablTk is included in the OCaml standard library. *)
#directory "+labltk";;
#load "labltk.cma";;

open Tk

let main = openTk ()

(* Create a horizontal space at the top of the window for the
   menu to live in. *)
let menubar = Frame.create ~relief:'Raised ~borderwidth:2 main
let () = pack ~anchor:'Nw ~fill:'X [menubar]

(* Create a button labeled "File" that brings up a menu *)
let file_menubutton = Menubutton.create ~text:"File" ~underline:1 menubar
let () = pack ~side:'Left [file_menubutton]

(* Create entries in the "File" menu *)
let file_menu = Menu.create file_menubutton
let () = Menubutton.configure ~menu:file_menu file_menubutton
let () = Menu.add_command ~label:"Print" ~command:print file_menu
let () = Menu.add_command ~label:"Save" ~command:save file_menu

(*-----*)

(* Create a menu item using an anonymous callback *)
let () =
  Menu.add_command
    ~label:"Quit Immediately"
    ~command:(fun () -> exit 0)
    file_menu

(*-----*)

(* Add a separator (a horizontal line) to the menu *)
let () = Menu.add_separator file_menu

```

```

(*-----*)

(* Create a checkbox menu item *)
let debug = Textvariable.create ~on:options_menu ()
let () =
  Menu.add_checkboxbutton
    ~label:"Create Debugging File"
    ~variable:debug
    ~onvalue:"1"
    ~offvalue:"0"
    options_menu

(*-----*)

(* Create radiobutton menu items *)
let log_level = Textvariable.create ~on:options_menu ()
let () =
  Menu.add_radiobutton
    ~label:"Level 1"
    ~variable:log_level
    ~value:"1"
    debug_menu
let () =
  Menu.add_radiobutton
    ~label:"Level 2"
    ~variable:log_level
    ~value:"2"
    debug_menu
let () =
  Menu.add_radiobutton
    ~label:"Level 3"
    ~variable:log_level
    ~value:"3"
    debug_menu

(*-----*)

(* Create a nested menu *)
let font_menu = Menu.create format_menubutton
let () = Menu.add_cascade ~label:"Font" ~menu:font_menu format_menu
let font_name = Textvariable.create ~on:font_menu ()
let () =
  Menu.add_radiobutton
    ~label:"Courier"
    ~variable:font_name
    ~value:"courier"
    font_menu
let () =
  Menu.add_radiobutton
    ~label:"Times Roman"
    ~variable:font_name
    ~value:"times"
    font_menu

```

```

(*-----*)

(* To disable tearoffs, use ~tearoff:false when calling Menu.create *)
let font_menu = Menu.create ~tearoff:false format_menubutton

(*-----*)

(* Start the Tk event loop and display the interface *)
let () = Printexc.print mainLoop ()

```

## 15.15 Creating Dialog Boxes with Tk

```

(* Tk::DialogBox is a CPAN module that replaces Tk's standard Dialog
   widget with one that can be customized with additional inputs. To
   get this effect in OCaml would require translating the whole CPAN
   module; instead, for this simple example, we will use the built-in
   Dialog. *)

#directory "+labltk";;
#load "labltk.cma";;

open Tk

let main = openTk ()

let dialog =
  Dialog.create
    ~title:"Register This Program"
    ~buttons:["Register"; "Cancel"]
    ~parent:main
    ~message:"..."

let () =
  match dialog () with
  | 0 -> print_endline "Register"
  | 1 -> print_endline "Cancel"
  | _ -> failwith "this shouldn't happen"

let () = Printexc.print mainLoop ()

(*-----*)

(* Normally, uncaught exceptions are printed to standard error. However,
   by overriding the "camlcb" callback, a custom error handler can be
   installed which creates dialogs instead. *)

#directory "+labltk";;
#load "labltk.cma";;

open Tk

let main = openTk ()

```

```

let show_error =
  let dialog =
    Dialog.create
      ~title:"Error"
      ~buttons:["Acknowledge"]
      ~parent:main in
  fun message -> ignore (dialog ~message ())

(* Override the "camlcb" callback. Note that this is an undocumented
   feature that relies on some internals of Labltk. *)
let () =
  Callback.register "camlcb"
    (fun id args ->
      try (Hashtbl.find Protocol.callback_naming_table id) args
      with e -> show_error (Printexc.to_string e))

let make_error () = failwith "This is an error"

let button1 =
  Button.create ~text:"Make An Error" ~command:make_error main
let () = pack ~side:'Left [button1]

let button2 =
  Button.create ~text:"Quit" ~command:(fun () -> exit 0) main
let () = pack ~side:'Left [button2]

let () = Printexc.print mainLoop ()

```

## 15.16 Responding to Tk Resize Events

```

open Tk

let main = openTk ()

(* Prevent the user from resizing the window. *)
let () =
  bind main
    ~events:['Configure]
    ~action:(fun _ ->
      let width = Winfo.width main in
      let height = Winfo.height main in
      Wm.minsize_set main width height;
      Wm.maxsize_set main width height)

(* Or, use pack to control how widgets are resized. *)
let () = pack ~fill:'Both ~expand:true [widget]
let () = pack ~fill:'X ~expand:true [widget]

(* Make the main area expand horizontally and vertically. *)
let () = pack ~fill:'Both ~expand:true [mainarea]

(* Make the menu bar only expand horizontally. *)

```

```

let () = pack ~fill:'X ~expand:true [menubar]

(* Anchor the menu bar to the top-left corner. *)
let () = pack ~fill:'X ~expand:true ~anchor:'Nw [menubar]

```

## 15.17 Removing the DOS Shell Window with Windows Perl/Tk

(\* Use Harry Chomsky's mkwinapp.ml from the OCaml-Win32 project:

<http://ocaml-win32.sourceforge.net/>

Compile your program using the native compiler and run mkwinapp.exe on the result. \*)

```

C:\MyProg> ocamlc myprog.ml -o myprog.exe
C:\MyProg> ocamlc unix.cma mkwinapp.ml -o mkwinapp.exe
C:\MyProg> mkwinapp myprog.exe

```

(\* Now you can run "myprog" and you won't get a console window. \*)

## 15.18 Program: Small termcap program

```

#!/usr/bin/ocaml

#directory "+curses";;
#load "curses.cma";;
#load "unix.cma";;

let delay = 0.005

(* Bounce lines around the screen until the user interrupts with
   Ctrl-C. *)
let zip () =
  Curses.clear ();
  let maxcol, maxrow = Curses.get_size () in

  let chars = ref ['*'; '-'; '/'; '|'; '\\'; '_'] in
  let circle () = chars := List.tl !chars @ [List.hd !chars] in

  let row, col = ref 0, ref 0 in
  let row_sign, col_sign = ref 1, ref 1 in

  while true do
    ignore (Curses.mvaddch !col !row (Char.code (List.hd !chars)));
    ignore (Curses.refresh ());
    (try ignore (Unix.select [] [] [] delay) with _ -> ());
    row := !row + !row_sign;
    col := !col + !col_sign;
    if !row = maxrow then (row_sign := -1; circle ())
    else if !row = 0 then (row_sign := 1; circle ());
    if !col = maxcol then (col_sign := -1; circle ())
    else if !col = 0 then (col_sign := 1; circle ())
  done

```

```

let () =
  ignore (Curses.initscr ());
  at_exit Curses.endwin;
  zip ()

```

## 15.19 Program: tkshufflepod

```

#!/usr/bin/ocaml
(* tkshufflepod - reorder =head1 sections in a pod file *)
#directory "+labltk";;
#load "labltk.cma";;

open Tk

(* Custom text viewer widget. *)

class viewer parent =
  let toplevel = Toplevel.create parent in
  let frame = Frame.create toplevel in
  let text = Text.create ~width:80 ~height:30 ~state:'Disabled frame in
  let vscroll = Scrollbar.create ~orient:'Vertical frame in

object (self)
  initializer
    self#hide ();
    Text.configure ~yscrollcommand:(Scrollbar.set vscroll) text;
    Scrollbar.configure ~command:(Text.yview text) vscroll;
    pack ~side:'Right ~fill:'Y [vscroll];
    pack ~side:'Left ~fill:'Both ~expand:true [text];
    pack ~side:'Right ~fill:'Both ~expand:true [frame];
    Wm.protocol_set toplevel "WM_DELETE_WINDOW" self#hide

  method show () = Wm.deiconify toplevel; raise_window toplevel
  method hide () = Wm.withdraw toplevel

  method set_title = Wm.title_set toplevel
  method set_body body =
    Text.configure ~state:'Normal text;
    Text.delete ~start:(~Atxy (0, 0), []) ~stop:(~End, []) text;
    Text.insert ~index:(~End, []) ~text:body text;
    Text.configure ~state:'Disabled text
end

(* Give list references a similar interface to Tk
   listbox widgets so we can keep the two in sync. *)

let listref_get listref index =
  match index with
  | ~Num i -> List.nth !listref i
  | _ -> failwith "listref_get"

let listref_delete listref index =

```

```

match index with
| 'Num i ->
    let rec loop current list =
        match list with
        | head :: tail when current = i -> loop (current + 1) tail
        | head :: tail -> head :: loop (current + 1) tail
        | [] -> [] in
    listref := loop 0 !listref
| _ -> failwith "listref_delete"

let listref_insert listref index elt =
    match index with
    | 'Num i ->
        let rec loop current list =
            match list with
            | head :: tail when current = i ->
                elt :: head :: loop (current + 1) tail
            | head :: [] when current = i - 1 -> head :: [elt]
            | head :: tail -> head :: loop (current + 1) tail
            | [] -> [] in
        listref := loop 0 !listref
    | _ -> failwith "listref_insert"

(* Use a line stream to produce a stream of POD chunks. *)

let line_stream_of_channel channel =
    Stream.from
        (fun _ -> try Some (input_line channel) with End_of_file -> None)

let pod_stream_of_channel channel =
    let lines = line_stream_of_channel channel in
    let is_head s = String.length s >= 6 && String.sub s 0 6 = "=head1" in
    let rec next pod_head pod_lines i =
        match Stream.peek lines, pod_head, pod_lines with
        | None, "", _ ->
            (* EOF, no POD found, return EOF *)
            None
        | None, _, _ ->
            (* EOF, POD found, return POD *)
            Some (pod_head, List.rev pod_lines)
        | Some head, "", _ when is_head head ->
            (* Head found *)
            Stream.junk lines;
            next head [] i
        | _, "", _ ->
            (* No head found, keep looking *)
            Stream.junk lines;
            next "" [] i
        | Some head, _, _ when is_head head ->
            (* Next head found, return POD *)
            Some (pod_head, List.rev pod_lines)
        | Some line, _, _ ->
            (* Line found, buffer and continue reading *)

```

```

        Stream.junk lines;
        next pod_head (line :: pod_lines) i in
    Stream.from (next "" [])

(* Read the POD file into memory, and split it into sections. *)

let podfile =
  if Array.length Sys.argv < 2
  then "-"
  else Sys.argv.(1)

let sections = ref []

(* Turn !sections into a list of (text, head) pairs. *)

let () =
  let channel = if podfile = "-" then stdin else open_in podfile in
  Stream.iter
    (fun (head, lines) ->
      sections := (String.concat "\n" lines, head) :: !sections)
    (pod_stream_of_channel channel);
  sections := List.rev !sections;
  close_in channel

(* Fire up Tk and display the list of sections. *)
let main = openTk ()
let listbox = Listbox.create ~width:60 main
let dragging = ref None

(* Singleton viewer instance. *)
let viewer = new viewer main

(* Called when the user clicks on an item in the Listbox. *)
let down event =
  dragging := Some (Listbox.nearest listbox event.ev_MouseY)

(* Called when the user releases the mouse button in the Listbox. *)
let up event =
  dragging := None

(* Called when the user moves the mouse in the Listbox. *)
let move event =
  let dest = Listbox.nearest listbox event.ev_MouseY in
  match !dragging with
  | Some src when src <> dest ->
    let elt = listref_get sections src in
    listref_delete sections src;
    listref_insert sections dest elt;
    let elt = Listbox.get listbox src in
    Listbox.delete listbox ~first:src ~last:src;
    Listbox.insert listbox ~index:dest ~texts:[elt];
    dragging := Some dest
  | _ -> ()

```

```

(* Called to save the list of sections. *)
let save event =
  let channel = if podfile = "-" then stdout else open_out podfile in
  List.iter
    (fun (text, head) ->
      output_string channel head;
      output_string channel "\n";
      output_string channel text;
      output_string channel "\n";
      flush channel)
    !sections;
  if podfile <> "-" then close_out channel

(* Called to display the widget. Uses the viewer widget. *)
let view event =
  dragging := None; (* cancel drag *)
  List.iter
    (fun ('Num i) ->
      let (text, head) = List.nth !sections i in
      viewer#set_title head;
      viewer#set_body (head ^ "\n" ^ text);
      viewer#show ())
    (Listbox.curselection listbox)

let () =
  pack ~expand:true ~fill:'Both [listbox];

  List.iter
    (fun (text, title) -> Listbox.insert listbox 'End [title])
    !sections;

  (* Permit dragging by binding to the Listbox widget. *)
  bind ~events:['ButtonPress] ~fields:['MouseY] ~action:down listbox;
  bind ~events:['ButtonRelease] ~action:up listbox;
  bind ~events:['Motion] ~fields:['MouseY] ~action:move listbox;

  (* Permit viewing by binding double-click. *)
  bind ~events:['Modified ([ 'Double], 'ButtonRelease)] ~action:view list-
  box;

  (* 'q' quits and 's' saves *)
  bind ~events:['KeyPressDetail "s"] ~action:save main;
  bind ~events:['KeyPressDetail "q"] ~action:(fun _ -> exit 0) main;

  Printexc.print mainLoop ()

```

## 16 Process Management and Communication

### 16.1 Gathering Output from a Program

```
(* Process support is mostly in the "unix" library. *)
#load "unix.cma";;

(* Run a command and return its results as a string. *)
let read_process command =
  let buffer_size = 2048 in
  let buffer = Buffer.create buffer_size in
  let string = String.create buffer_size in
  let in_channel = Unix.open_process_in command in
  let chars_read = ref 1 in
  while !chars_read <> 0 do
    chars_read := input in_channel string 0 buffer_size;
    Buffer.add_substring buffer string 0 !chars_read
  done;
  ignore (Unix.close_process_in in_channel);
  Buffer.contents buffer

(* Run a command and return its results as a list of strings,
   one per line. *)
let read_process_lines command =
  let lines = ref [] in
  let in_channel = Unix.open_process_in command in
  begin
    try
      while true do
        lines := input_line in_channel :: !lines
      done;
      with End_of_file ->
        ignore (Unix.close_process_in in_channel)
    end;
    List.rev !lines
  end

(* Example: *)
let output_string = read_process "program args"
let output_lines = read_process_lines "program args"

(*-----*)

(* Create a pipe for the subprocess output. *)
let readme, writeme = Unix.pipe ()

(* Launch the program, redirecting its stdout to the pipe.
   By calling Unix.create_process, we can avoid running the
   command through the shell. *)
let () =
  let pid = Unix.create_process
    program [| program; arg1; arg2 |]
    Unix.stdin writeme Unix.stderr in
  Unix.close writeme;
```

```

let in_channel = Unix.in_channel_of_descr readme in
let lines = ref [] in
begin
  try
    while true do
      lines := input_line in_channel :: !lines
    done
    with End_of_file -> ()
  end;
  Unix.close readme;
  List.iter print_endline (List.rev !lines)

```

## 16.2 Running Another Program

```

(* Run a simple command and retrieve its result code. *)
let status = Sys.command ("vi " ^ myfile)

(*-----*)

(* Use the shell to perform redirection. *)
let _ = Sys.command "cmd1 args | cmd2 | cmd3 >outfile"
let _ = Sys.command "cmd args <infile >outfile 2>errfile"

(*-----*)

(* Run a command, handling its result code or signal. *)
#load "unix.cma";;
let () =
  match Unix.system command with
  | Unix.WEXITED status ->
    Printf.printf "program exited with status %d\n" status
  | Unix.WSIGNALED signal ->
    Printf.printf "program killed by signal %d\n" signal
  | Unix.WSTOPPED signal ->
    Printf.printf "program stopped by signal %d\n" signal

(*-----*)

(* Run a command while blocking interrupt signals. *)
#load "unix.cma";;
let () =
  match Unix.fork () with
  | 0 ->
    (* child ignores INT and does its thing *)
    Sys.set_signal Sys.sigint Sys.Signal_ignore;
    Unix.execv "/bin/sleep" [| "/bin/sleep"; "10" |]
  | pid ->
    (* parent catches INT and berates user *)
    Sys.set_signal Sys.sigint
      (Sys.Signal_handle
        (fun _ -> print_endline "Tsk tsk, no process interruptus"));
    let running = ref true in
    while !running do

```

```

        try (ignore (Unix.waitpid [] pid); running := false)
        with Unix.Unix_error _ -> ()
    done;
    Sys.set_signal Sys.sigint Sys.Signal_default

(*-----*)

(* Run a command with a different name in the process table. *)
#load "unix.cma";;
let shell = "/bin/tcsh"
let () =
  match Unix.fork () with
  | 0 -> Unix.execv shell [| "-csh" |] (* pretend it's a login shell *)
  | pid -> ignore (Unix.waitpid [] pid)

```

### 16.3 Replacing the Current Program with a Different One

```

#load "unix.cma";;
(* Transfer control to the shell to run another program. *)
let () = Unix.execv "/bin/sh" [| "/bin/sh"; "-c"; "archive *.data" |]
(* Transfer control directly to another program in the path. *)
let () = Unix.execvp "archive" [| "archive"; "accounting.data" |]

```

### 16.4 Reading or Writing to Another Program

```

#load "unix.cma";;

(*-----*)

(* Handle each line in the output of a process. *)
let () =
  let readme = Unix.open_process_in "program arguments" in
  let rec loop line =
    (* ... *)
    loop (input_line readme) in
  try loop (input_line readme)
  with End_of_file -> ignore (Unix.close_process_in readme)

(*-----*)

(* Write to the input of a process. *)
let () =
  let writeme = Unix.open_process_out "program arguments" in
  output_string writeme "data\n";
  ignore (Unix.close_process_out writeme)

(*-----*)

(* Wait for a process to complete. *)
let () =
  (* child goes to sleep *)
  let f = Unix.open_process_in "sleep 100000" in
  (* and parent goes to lala land *)

```

```

ignore (Unix.close_process_in f);
ignore (Unix.wait ())

(*-----*)

let () =
  let writeme = Unix.open_process_out "program args" in
  (* program will get hello\n on STDIN *)
  output_string writeme "hello\n";
  (* program will get EOF on STDIN *)
  ignore (Unix.close_process_out writeme)

(*-----*)

(* Redirect standard output to the pager. *)
let () =
  let pager =
    try Sys.getenv "PAGER" (* XXX: might not exist *)
    with Not_found -> "/usr/bin/less" in
  let reader, writer = Unix.pipe () in
  match Unix.fork () with
  | 0 ->
    Unix.close writer;
    Unix.dup2 reader Unix.stdin;
    Unix.close reader;
    Unix.execvp pager [| pager |]
  | pid ->
    Unix.close reader;
    Unix.dup2 writer Unix.stdout;
    Unix.close writer

(* Do something useful that writes to standard output, then
   close the stream and wait for the pager to finish. *)
let () =
  (* ... *)
  close_out stdout;
  ignore (Unix.wait ())

```

## 16.5 Filtering Your Own Output

```

#load "unix.cma";;

(* Fork a process that calls f to post-process standard output. *)
let push_output_filter f =
  let reader, writer = Unix.pipe () in
  match Unix.fork () with
  | 0 ->
    Unix.close writer;
    Unix.dup2 reader Unix.stdin;
    Unix.close reader;
    f ();
    exit 0
  | pid ->

```

```

        Unix.close reader;
        Unix.dup2 writer Unix.stdout;
        Unix.close writer

(* Only display a certain number of lines of output. *)
let head ?(lines=20) () =
  push_output_filter
  (fun () ->
    let lines = ref lines in
    try
      while !lines > 0 do
        print_endline (read_line ());
        decr lines
      done
    with End_of_file -> ())

(* Prepend line numbers to each line of output. *)
let number () =
  push_output_filter
  (fun () ->
    let line_number = ref 0 in
    try
      while true do
        let line = read_line () in
        incr line_number;
        Printf.printf "%d: %s\n" !line_number line
      done
    with End_of_file -> ())

(* Prepend "> " to each line of output. *)
let quote () =
  push_output_filter
  (fun () ->
    try
      while true do
        let line = read_line () in
        Printf.printf "> %s\n" line
      done
    with End_of_file -> ())

let () =
  head ~lines:100 (); (* push head filter on STDOUT *)
  number ();          (* push number filter on STDOUT *)
  quote ();           (* push quote filter on STDOUT *)

(* act like /bin/cat *)
begin
  try
    while true do
      print_endline (read_line ())
    done
  with End_of_file -> ()
end;

```

```

(* tell kids we're done--politely *)
close_out stdout;
ignore (Unix.waitpid [] (-1));
exit 0

```

## 16.6 Preprocessing Input

```

#load "unix.cma";;
#load "str.cma";;

(* Tagged filename or URL type. *)
type filename =
  | Uncompressed of string
  | Compressed of string
  | URL of string

(* try/finally-
like construct to ensure we dispose of resources properly. *)
let finally handler f x =
  let result = try f x with e -> handler (); raise e in handler (); result

(* Call f with an in_channel given a tagged filename. If the filename is
tagged Uncompressed, open it normally. If it is tagged Compressed then
pipe it through gzip. If it is tagged URL, pipe it through "lynx -dump".
Ensure that the channel is closed and any created processes have
terminated before returning. As a special case, a filename of
Uncompressed "-" will result in stdin being passed, and no channel
will be closed. *)
let with_in_channel filename f =
  let pipe_input args f =
    let reader, writer = Unix.pipe () in
    let pid =
      Unix.create_process args.(0) args Unix.stdin writer Unix.stderr in
    Unix.close writer;
    let in_channel = Unix.in_channel_of_descr reader in
    finally
      (fun () -> close_in in_channel; ignore (Unix.waitpid [] pid))
      f in_channel in
  match filename with
  | Uncompressed "-" ->
    f stdin
  | Uncompressed filename ->
    let in_channel = open_in filename in
    finally
      (fun () -> close_in in_channel)
      f in_channel
  | Compressed filename ->
    pipe_input [| "gzip"; "-dc"; filename |] f
  | URL url ->
    pipe_input [| "lynx"; "-dump"; url |] f

(* Return true if the string s starts with the given prefix. *)

```

```

let starts_with s prefix =
  try Str.first_chars s (String.length prefix) = prefix
  with Invalid_argument _ -> false

(* Return true if the string s ends with the given suffix. *)
let ends_with s suffix =
  try Str.last_chars s (String.length suffix) = suffix
  with Invalid_argument _ -> false

(* Return true if the string s contains the given substring. *)
let contains s substring =
  try ignore (Str.search_forward (Str.regexp_string substring) s 0); true
  with Not_found -> false

(* Tag the filename depending on its contents or extension. *)
let tag_filename filename =
  if contains filename "://"
  then URL filename
  else if List.exists (ends_with filename) [".gz"; ".Z"]
  then Compressed filename
  else Uncompressed filename

(* Process a tagged filename. *)
let process filename =
  with_in_channel
    filename
    (fun in_channel ->
      try
        while true do
          let line = input_line in_channel in
            (* ... *)
            ()
        done
      with End_of_file -> ())

(* Parse the command-line arguments and process each file or URL. *)
let () =
  let args =
    if Array.length Sys.argv > 1
    then (List.tl (Array.to_list Sys.argv))
    else ["-"] in
  List.iter process (List.map tag_filename args)

```

## 16.7 Reading STDERR from a Program

```

#load "unix.cma";;

(* Read STDERR and STDOUT at the same time. *)
let () =
  let ph = Unix.open_process_in "cmd 2>&1" in
  while true do
    let line = input_line ph in
    (* ... *)

```

```

    ()
done

(*-----*)

(* Read STDOUT and discard STDERR. *)
let output = read_process "cmd 2>/dev/null"
(* or *)
let () =
  let ph = Unix.open_process_in "cmd 2>/dev/null" in
  while true do
    let line = input_line ph in
    (* ... *)
    ()
  done

(*-----*)

(* Read STDERR and discard STDOUT. *)
let output = read_process "cmd 2>&1 1>/dev/null"
(* or *)
let () =
  let ph = Unix.open_process_in "cmd 2>&1 1>/dev/null" in
  while true do
    let line = input_line ph in
    (* ... *)
    ()
  done

(*-----*)

(* Swap STDOUT with STDERR and read original STDERR. *)
let output = read_process "cmd 3>&1 1>&2 2>&3 3>&-"
(* or *)
let () =
  let ph = Unix.open_process_in "cmd 3>&1 1>&2 2>&3 3>&-" in
  while true do
    let line = input_line ph in
    (* ... *)
    ()
  done

(*-----*)

(* Redirect STDOUT and STDERR to temporary files. *)
let () =
  ignore
  (Sys.command
   "program args 1>/tmp/program.stdout 2>/tmp/program.stderr")

(*-----*)

(* If the following redirections were done in OCaml... *)

```

```

let output = read_process "cmd 3>&1 1>&2 2>&3 3>&-"

(* ...they would look something like this: *)
let fd3 = fd1
let fd1 = fd2
let fd2 = fd3
let fd3 = undef

(*-----*)

(* Send STDOUT and STDERR to a temporary file. *)
let () = ignore (Sys.command "prog args 1>tmpfile 2>&1")

(* Send STDOUT to a temporary file and redirect STDERR to STDOUT. *)
let () = ignore (Sys.command "prog args 2>&1 1>tmpfile")

(*-----*)

(* If the following redirections were done in OCaml... *)
let () = ignore (Sys.command "prog args 1>tmpfile 2>&1")

(* ...they would look something like this: *)
let fd1 = "tmpfile"      (* change stdout destination first *)
let fd2 = fd1           (* now point stderr there, too *)

(*-----*)

(* If the following redirections were done in OCaml... *)
let () = ignore (Sys.command "prog args 2>&1 1>tmpfile")

(* ...they would look something like this: *)
let fd2 = fd1           (* stderr same destination as stdout *)
let fd1 = "tmpfile"    (* but change stdout destination *)

```

## 16.8 Controlling Input and Output of Another Program

```

#load "unix.cma";;

let () =
  let (readme, writeme) = Unix.open_process program in
    output_string writeme "here's your input\n";
    close_out writeme;
    let output = input_line readme in
      ignore (Unix.close_process (readme, writeme))

```

## 16.9 Controlling the Input, Output, and Error of Another Program

```

#load "unix.cma";;
let () =
  let proc =
    Unix.open_process_in
      ("(" ^ cmd ^ " | sed -e 's/^/stdout: /' ) 2>&1") in
  try

```

```

while true do
  let line = input_line proc in
  if String.length line >= 8
    && String.sub line 0 8 = "stdout: "
  then Printf.printf "STDOUT: %s\n"
    (String.sub line 8 (String.length line - 8))
  else Printf.printf "STDERR: %s\n" line
  done
with End_of_file ->
  ignore (Unix.close_process_in proc)

(*-----*)

#!/usr/bin/ocaml
(* cmd3sel - control all three of kids in, out, and error. *)
#load "unix.cma";;

let cmd = "grep vt33 /none/such - /etc/termcap"
let cmd_out, cmd_in, cmd_err = Unix.open_process_full cmd [| |]

let () =
  output_string cmd_in "This line has a vt33 lurking in it\n";
  close_out cmd_in;
  let cmd_out_descr = Unix.descr_of_in_channel cmd_out in
  let cmd_err_descr = Unix.descr_of_in_channel cmd_err in
  let selector = ref [cmd_err_descr; cmd_out_descr] in
  while !selector <> [] do
    let can_read, _, _ = Unix.select !selector [] [] 1.0 in
    List.iter
      (fun fh ->
        try
          if fh = cmd_err_descr
            then Printf.printf "STDERR: %s\n" (input_line cmd_err)
          else Printf.printf "STDOUT: %s\n" (input_line cmd_out)
        with End_of_file ->
          selector := List.filter (fun fh' -> fh <> fh') !selector)
      can_read
  done;
  ignore (Unix.close_process_full (cmd_out, cmd_in, cmd_err))

```

## 16.10 Communicating Between Related Processes

```

(* pipe1 - use pipe and fork so parent can send to child *)
#load "unix.cma"
open Unix

let reader, writer = pipe ()

let () =
  match fork () with
  | 0 ->
    close writer;
    let input = in_channel_of_descr reader in

```

```

        let line = input_line input in
        Printf.printf "Child Pid %d just read this: '%s'\n" (get-
pid ()) line;
        close reader; (* this will happen anyway *)
        exit 0
    | pid ->
        close reader;
        let output = out_channel_of_descr writer in
        Printf.fprintf output "Parent Pid %d is sending this\n" (get-
pid ());
        flush output;
        close writer;
        ignore (waitpid [] pid)

(*-----*)

(* pipe2 - use pipe and fork so child can send to parent *)
#load "unix.cma"
open Unix

let reader, writer = pipe ()

let () =
    match fork () with
    | 0 ->
        close reader;
        let output = out_channel_of_descr writer in
        Printf.fprintf output "Child Pid %d is sending this\n" (getpid ());
        flush output;
        close writer; (* this will happen anyway *)
        exit 0
    | pid ->
        close writer;
        let input = in_channel_of_descr reader in
        let line = input_line input in
        Printf.printf "Parent Pid %d just read this: '%s'\n" (get-
pid ()) line;
        close reader;
        ignore (waitpid [] pid)

(*-----*)

(* pipe3 and pipe4 demonstrate the use of perl's "forking open" feature to
* reimplement pipe1 and pipe2. Since OCaml does not support such a fea-
* ture,
* these are skipped here. *)

(*-----*)

(* pipe5 - bidirectional communication using two pipe pairs
    designed for the socketpair-challenged *)
#load "unix.cma"
open Unix

```

```

let parent_rdr, child_wtr = pipe ()
let child_rdr, parent_wtr = pipe ()

let () =
  match fork () with
  | 0 ->
    close child_rdr;
    close child_wtr;
    let input = in_channel_of_descr parent_rdr in
    let output = out_channel_of_descr parent_wtr in
    let line = input_line input in
    Printf.printf "Child Pid %d just read this: '%s'\n" (get-
pid ()) line;
    Printf.fprintf output "Child Pid %d is sending this\n" (getpid ());
    flush output;
    close parent_rdr;
    close parent_wtr;
    exit 0
  | pid ->
    close parent_rdr;
    close parent_wtr;
    let input = in_channel_of_descr child_rdr in
    let output = out_channel_of_descr child_wtr in
    Printf.fprintf output "Parent Pid %d is sending this\n" (getpid ());
    flush output;
    let line = input_line input in
    Printf.printf "Parent Pid %d just read this: '%s'\n" (get-
pid ()) line;
    close child_rdr;
    close child_wtr;
    ignore (waitpid [] pid)

(*-----*)

(* pipe6 - bidirectional communication using socketpair
   "the best ones always go both ways" *)
#load "unix.cma"
open Unix

let child, parent = socketpair PF_UNIX SOCK_STREAM 0

let () =
  match fork () with
  | 0 ->
    close child;
    let input = in_channel_of_descr parent in
    let output = out_channel_of_descr parent in
    let line = input_line input in
    Printf.printf "Child Pid %d just read this: '%s'\n" (get-
pid ()) line;
    Printf.fprintf output "Child Pid %d is sending this\n" (getpid ());
    flush output;

```

```

        close parent;
        exit 0
    | pid ->
        close parent;
        let input = in_channel_of_descr child in
        let output = out_channel_of_descr child in
        Printf.fprintf output "Parent Pid %d is sending this\n" (get-
pid ());
        flush output;
        let line = input_line input in
        Printf.printf "Parent Pid %d just read this: '%s'\n" (get-
pid ()) line;
        close child;
        ignore (waitpid [] pid)

(*-----*)

(* Simulating a pipe using a socketpair. *)
let reader, writer = socketpair PF_UNIX SOCK_STREAM 0 in
shutdown reader SHUTDOWN_SEND;      (* no more writing for reader *)
shutdown writer SHUTDOWN_RECEIVE;   (* no more reading for writer *)

```

## 16.11 Making a Process Look Like a File with Named Pipes

```

% mkfifo /path/to/named.pipe

(*-----*)

let () =
  let fifo = open_in "/path/to/named.pipe" in
  try
    while true do
      let line = input_line fifo in
      Printf.printf "Got: %s\n" line
    done
  with End_of_file ->
    close_in fifo

(*-----*)

let () =
  let fifo = open_out "/path/to/named.pipe" in
  output_string fifo "Smoke this.\n";
  close_out fifo

(*-----*)

% mkfifo ~/.plan          # isn't this everywhere yet?
% mknod  ~/.plan p       # in case you don't have mkfifo

(*-----*)

(* dateplan - place current date and time in .plan file *)

```

```

#load "unix.cma";;
let () =
  while true do
    let home = Unix.getenv "HOME" in
    let fifo = open_out (home ^ "/.plan") in
    Printf.fprintf fifo "The current time is %s\n"
      (format_time (Unix.time ()));
    close_out fifo;
    Unix.sleep 1
  done

(*-----*)

#!/usr/bin/ocaml
(* fifolog - read and record log msgs from fifo *)
#load "unix.cma";;

let fifo = ref None

let handle_alarm signal =
  match !fifo with
  | Some channel ->
    (* move on to the next queued process *)
    close_in channel;
    fifo := None
  | None -> ()

let () =
  Sys.set_signal Sys.sigalrm (Sys.Signal_handle handle_alarm)

let read_fifo () =
  try
    match !fifo with
    | Some channel -> Some (input_line channel)
    | None -> None
  with
  | End_of_file ->
    None
  | Sys_error e ->
    Printf.eprintf "Error reading fifo: %s\n%!" e;
    fifo := None;
    None

let days = [| "Sun"; "Mon"; "Tue"; "Wed"; "Thu"; "Fri"; "Sat" |]
let months = [| "Jan"; "Feb"; "Mar"; "Apr"; "May"; "Jun";
  "Jul"; "Aug"; "Sep"; "Oct"; "Nov"; "Dec" |]

let format_time time =
  let tm = Unix.localtime time in
  Printf.sprintf "%s %s %2d %02d:%02d:%02d %04d"
    days.(tm.Unix.tm_wday)
    months.(tm.Unix.tm_mon)
    tm.Unix.tm_mday

```

```

tm.Unix.tm_hour
tm.Unix.tm_min
tm.Unix.tm_sec
(tm.Unix.tm_year + 1900)

let () =
  while true do
    (* turn off alarm for blocking open *)
    ignore (Unix.alarm 0);
    begin
      try fifo := Some (open_in "/tmp/log")
      with Sys_error e ->
        Printf.eprintf "Can't open /tmp/log: %s\n%!" e;
        exit 1
    end;

    (* you have 1 second to log *)
    ignore (Unix.alarm 1);

    let service = read_fifo () in
    let message = read_fifo () in

    (* turn off alarms for message processing *)
    ignore (Unix.alarm 0);

    begin
      match service, message with
      | None, _ | _, None ->
        (* interrupted or nothing logged *)
        ()
      | Some service, Some message ->
        if service = "http"
        then () (* ignoring *)
        else if service = "login"
        then
          begin
            (* log to /tmp/login *)
            try
              let log =
                open_out_gen
                  [Open_wronly; Open_creat; Open_append]
                  0o666
                  "/tmp/login" in
                Printf.fprintf log "%s %s %s\n%!"
                  (format_time (Unix.time ())) service message;
                close_out log
            with Sys_error e ->
              Printf.eprintf "Couldn't log %s %s to /tmp/login: %s\n%!"
                service message e
          end
        end
    end
  end
done

```

## 16.12 Sharing Variables in Different Processes

(\* OCaml does not currently support SysV IPC. \*)

## 16.13 Listing Available Signals

```
% echo 'module M = Sys;;' | ocaml | grep 'val sig'
  val sigabrt : int
  val sigalrm : int
  val sigfpe : int
  val sighup : int
  val sigill : int
  val sigint : int
  val sigkill : int
  val sigpipe : int
  val sigquit : int
  val sigsegv : int
  val sigterm : int
  val sigusr1 : int
  val sigusr2 : int
  val sigchld : int
  val sigcont : int
  val sigstop : int
  val sigtstp : int
  val sigttin : int
  val sigttou : int
  val sigvtalrm : int
  val sigprof : int

% grep -A1 'val sig' sys.mli
val sigabrt : int
(** Abnormal termination *)
--
val sigalrm : int
(** Timeout *)
--
val sigfpe : int
(** Arithmetic exception *)
--
val sighup : int
(** Hangup on controlling terminal *)
--
val sigill : int
(** Invalid hardware instruction *)
--
val sigint : int
(** Interactive interrupt (ctrl-C) *)
--
val sigkill : int
(** Termination (cannot be ignored) *)
--
val sigpipe : int
(** Broken pipe *)
```

```

--
val sigquit : int
(** Interactive termination *)
--
val sigsegv : int
(** Invalid memory reference *)
--
val sigterm : int
(** Termination *)
--
val sigusr1 : int
(** Application-defined signal 1 *)
--
val sigusr2 : int
(** Application-defined signal 2 *)
--
val sigchld : int
(** Child process terminated *)
--
val sigcont : int
(** Continue *)
--
val sigstop : int
(** Stop *)
--
val sigtstp : int
(** Interactive stop *)
--
val sigttin : int
(** Terminal read from background process *)
--
val sigttou : int
(** Terminal write from background process *)
--
val sigvtalrm : int
(** Timeout in virtual time *)
--
val sigprof : int
(** Profiling interrupt *)

```

## 16.14 Sending a Signal

```

#load "unix.cma";;
let () =
  (* send pid a signal 9 *)
  Unix.kill pid 9;
  (* send whole job a signal 1 *)
  Unix.kill pgrp (-1);
  (* send myself a SIGUSR1 *)
  Unix.kill (Unix.getpid ()) Sys.sigusr1;
  (* send a SIGHUP to processes in pids *)
  List.iter (fun pid -> Unix.kill pid Sys.sighup) pids

```

```
(*-----*)

(* Use kill with pseudo-signal 0 to see if process is alive. *)
let () =
  try
    Unix.kill minion 0;
    Printf.printf "%d is alive!\n" minion
  with
  | Unix.Unix_error (Unix.EPERM, _, _) -> (* changed uid *)
    Printf.printf "%d has escaped my control!\n" minion
  | Unix.Unix_error (Unix.ESRCH, _, _) ->
    Printf.printf "%d is deceased.\n" (* or zombied *) minion
  | e ->
    Printf.printf "Odd; I couldn't check on the status of %d: %s\n"
      minion
      (Printexc.to_string e)
```

## 16.15 Installing a Signal Handler

```
let () =
  (* call got_sig_quit for every SIGQUIT *)
  Sys.set_signal Sys.sigquit (Sys.Signal_handle got_sig_quit);
  (* call got_sig_pipe for every SIGPIPE *)
  Sys.set_signal Sys.sigpipe (Sys.Signal_handle got_sig_pipe);
  (* increment ouch for every SIGINT *)
  Sys.set_signal Sys.sigint (Sys.Signal_handle (fun _ -> incr ouch));
  (* ignore the signal INT *)
  Sys.set_signal Sys.sigint Sys.Signal_ignore;
  (* restore default STOP signal handling *)
  Sys.set_signal Sys.sigstop Sys.Signal_default
```

## 16.16 Temporarily Overriding a Signal Handler

```
let finally handler f x =
  let result = try f x with e -> handler (); raise e in handler (); result

(* call f with signal behavior temporarily set *)
let local_set_signal signal behavior f =
  let old_behavior = Sys.signal signal behavior in
  finally (fun () -> Sys.set_signal signal old_behavior) f ()

(* the signal handler *)
let rec ding _ =
  Sys.set_signal Sys.sigint (Sys.Signal_handle ding);
  prerr_endline "\x07Enter your name!"

(* prompt for name, overriding SIGINT *)
let get_name () =
  local_set_signal
  Sys.sigint (Sys.Signal_handle ding)
  (fun () ->
    print_string "Kindly Stranger, please enter your name: ";
    read_line ())
```

## 16.17 Writing a Signal Handler

```
let rec got_int _ =
  Sys.set_signal Sys.sigint (Sys.Signal_handle got_int);
  (* but not for SIGCHLD! *)
  (* ... *)
  ()

(*-----*)

let rec got_int _ =
  Sys.set_signal Sys.sigint Sys.Signal_default; (* or Signal_ignore *)
  failwith "interrupted"

let () =
  Sys.set_signal Sys.sigint (Sys.Signal_handle got_int);
  try
    (* ... long-running code that you don't want to restart *)
    ()
  with Failure "interrupted" ->
    (* deal with the signal *)
    ()
```

## 16.18 Catching Ctrl-C

```
let () =
  (* ignore signal INT *)
  Sys.set_signal Sys.sigint Sys.Signal_ignore;

  (* install signal handler *)
  let rec tsksk signal =
    Sys.set_signal Sys.sigint (Sys.Signal_handle tsksk);
    print_endline "\x07The long habit of living indisposeth us for dy-
ing." in
    Sys.set_signal Sys.sigint (Sys.Signal_handle tsksk)
```

## 16.19 Avoiding Zombie Processes

```
#load "unix.cma";;

let () =
  Sys.set_signal Sys.sigchld Sys.Signal_ignore

(*-----*)

let rec reaper signal =
  try while true do ignore (Unix.waitpid [Unix.WNOHANG] (-1)) done
  with Unix.Unix_error (Unix.ECHILD, _, _) -> ();
  Sys.set_signal Sys.sigchld (Sys.Signal_handle reaper)

let () =
  Sys.set_signal Sys.sigchld (Sys.Signal_handle reaper)
```

```

(*-----*)

let rec reaper signal =
  begin try
    let pid, status = Unix.waitpid [Unix.WNOHANG] (-1) in begin
      match status with
      | Unix.WEXITED _ ->
          Printf.printf "Process %d exited.\n" pid
      | _ ->
          Printf.printf "False alarm on %d.\n" pid;
    end;
    reaper signal
  with Unix.Unix_error (Unix.ECHILD, _, _) ->
    () (* No child waiting. Ignore it. *)
  end;
  Sys.set_signal Sys.sigchld (Sys.Signal_handle reaper)

let () =
  Sys.set_signal Sys.sigchld (Sys.Signal_handle reaper)

```

## 16.20 Blocking Signals

```

#load "unix.cma";;

(* define the signals to block *)
let sigset = [Sys.sigint; Sys.sigkill]

let () =
  (* block signals *)
  let old_sigset = Unix.sigprocmask Unix.SIG_BLOCK sigset in

  (* ... *)

  (* unblock signals *)
  (* the origi-
nal recipe uses SIG_UNBLOCK, but that doesn't seem right... *)
  ignore (Unix.sigprocmask Unix.SIG_SETMASK old_sigset)

```

## 16.21 Timing Out an Operation

```

#load "unix.cma";;
let () =
  Sys.set_signal Sys.sigalrm
    (Sys.Signal_handle (fun _ -> failwith "timeout"));

  ignore (Unix.alarm 3600);
  try
    (* long-time operations here *)
    ignore (Unix.alarm 0)
  with
    | Failure "timeout" ->
        (* timed out; do what you will here *)
        ()

```

```

| e ->
  (* clear the still-pending alarm *)
  ignore (Unix.alarm 0);
  (* propagate unexpected exception *)
  raise e

```

## 16.22 Program: sigrand

```

#!/usr/bin/ocaml
(* sigrand - supply random fortunes for .signature file *)
#load "str.cma";;
#load "unix.cma";;

(* globals *)

let pwd = Unix.getpwuid (Unix.getuid ())

let home =
  try Unix.getenv "HOME" with Not_found ->
  try Unix.getenv "LOGDIR" with Not_found ->
  pwd.Unix.pw_dir

let fortune_path = ref ""

(*****)
(* begin configuration section *)

(* for rec/humor/funny instead of rec.humor.funny *)
let ng_is_dir      = true

let fullname      = home ^ "/.fullname"
let fifo          = home ^ "/.signature"
let art           = home ^ "/.article"
let news          = home ^ "/News"
let sigs          = news ^ "/SIGNATURES"
let sema         = home ^ "/.sigrandpid"
let globrand     = 0.25 (* chance to use global sigs anyway *)

(* name should be (1) left None to have program guess
   read address for signature maybe looking in ~/.fullname,
   (2) set to an exact address, or (3) set to empty string
   to be omitted entirely. *)

(* let name       = ref None *)
(* let name       = ref (Some ("me@home.org")) *)
let name          = ref (Some "")

(* end configuration section *)
(*****)

let read_process_lines command =
  let lines = ref [] in
  let in_channel = Unix.open_process_in command in

```

```

begin
  try
    while true do
      lines := input_line in_channel :: !lines
    done;
  with End_of_file ->
    ignore (Unix.close_process_in in_channel)
end;
List.rev !lines

let line_stream_of_channel channel =
  Stream.from
    (fun _ -> try Some (input_line channel) with End_of_file -> None)

let delimited_stream_of_channel delim channel =
  let lines = line_stream_of_channel channel in
  let rec next para_lines i =
    match Stream.peek lines, para_lines with
    | None, [] -> None
    | Some delim', [] when delim' = delim ->
      Stream.junk lines; next para_lines i
    | Some delim', _ when delim' = delim ->
      Some (String.concat "\n" (List.rev para_lines))
    | None, _ ->
      Some (String.concat "\n" (List.rev para_lines))
    | Some line, _ -> Stream.junk lines; next (line :: para_lines) i in
  Stream.from (next [])

(* Make sure there's a fortune program. Search
   for its full path and set global to that. *)
let check_fortunes () =
  if !fortune_path <> ""
  then () (* already set *)
  else
    let path = Str.split (Str.regexp ":") (Unix.getenv "PATH") in
    let rec check = function
      | [] ->
        Printf.eprintf
          "Need either %s or a fortune program, bailing out\n"
          sigs;
        exit 1
      | dir :: dirs ->
        let p = Filename.concat dir "fortune" in
        if Sys.file_exists p then p else check dirs in
    fortune_path := check (path @ ["/usr/games"])

(* Call the fortune program with -s for short flag until
   we get a small enough fortune or ask too much. *)
let fortune () =
  let cmd = !fortune_path ^ " -s" in
  let rec loop tries =
    let lines = read_process_lines cmd in
    if List.length lines < 5 then lines

```

```

    else if tries < 20 then loop (tries + 1)
    else [] in
match loop 0 with
| [] ->
    [" SIGRAND: deliver random signals to all processes."]
| lines ->
    List.map (( ^ ) " ") lines

(* See whether ~/.article contains a Newsgroups line. if so, see the
first group posted to and find out whether it has a dedicated set of
fortunes. otherwise return the global one. Also, return the global
one randomly now and then to spice up the sigs. *)
let signame () =
  if Random.float 1.0 > globrand
  then
    begin
      try
        let channel = open_in art in
        let regexp = Str.regexp "Newsgroups:[ \t]*\\\[^\, \r\n\t]*\\" in
        let ng = ref "" in
        begin
          try
            while true do
              let line = input_line channel in
              if Str.string_match regexp line 0
              then ng := Str.matched_group 1 line
            done
            with End_of_file ->
              close_in channel
          end;
          if ng_is_dir
          then ng := Str.global_replace (Str.regexp "\\.") "/" !ng;
          ng := news ^ "/" ^ !ng ^ "/" ^ "SIGNATURES";
          if Sys.file_exists !ng then !ng else sigs
        with Sys_error e ->
          sigs
        end
      else sigs
    end

(* choose a random signature *)
let pick_quote () =
  let sigfile = signame () in
  if not (Sys.file_exists sigfile)
  then fortune ()
  else
    begin
      let channel = open_in sigfile in
      let stream = delimited_stream_of_channel "%%" channel in
      let quip = ref [] in
      let num = ref 1 in
      Stream.iter
        (fun chunk ->
          if Random.int !num = 0

```

```

        then quip := Str.split (Str.regexp "\n") chunk;
        incr num)
    stream;
    close_in channel;
    if !quip <> []
    then List.map (( ^ ) " ") !quip
    else [" ENOSIG: This signature file is empty."]
    end
end

(* Ignore SIGPIPE in case someone opens us up and then closes the fifo
without reading it; look in a .fullname file for their login name.
Try to determine the fully qualified hostname. Make sure we have
signatures or fortunes. Build a fifo if we need to. *)

let setup () =
  Sys.set_signal Sys.sigpipe Sys.Signal_ignore;

  if !name = Some "" then
    begin
      try
        let channel = open_in fullname in
          name := Some (input_line channel);
          close_in channel
        with Sys_error _ ->
          name := Some (Str.global_replace (Str.regexp ",.*") ""
            pwd.Unix.pw_gecos)
      end;
    end;

  if not (Sys.file_exists sigs) then check_fortunes ();

  if Sys.file_exists fifo
  then (if (Unix.stat fifo).Unix.st_kind = Unix.S_FIFO
        then (Printf.eprintf "%s: using existing named pipe %s\n"
          Sys.argv.(0) fifo)
        else (Printf.eprintf "%s: won't overwrite file %s\n"
          Sys.argv.(0) fifo;
          exit 1))
  else (Unix.mkfifo fifo 0o666;
        Printf.eprintf "%s: created %s as a named pipe\n"
          Sys.argv.(0) fifo);

  Random.self_init ()

(* "There can be only one." --the Highlander *)
let justme () =
  let channel =
    try Some (open_in sema)
    with Sys_error _ -> None in
  match channel with
  | Some channel ->
    begin
      let pid = int_of_string (input_line channel) in
      try

```

```

        Unix.kill pid 0;
        Printf.eprintf "%s already running (pid %d), bailing out\n"
            Sys.argv.(0) pid;
        exit 1
    with _ ->
        close_in channel
    end
| None -> ()

let () =
    setup ();                (* pull in inits *)
    justme ();               (* make sure program not already running *)
    match Unix.fork () with (* background ourself and go away *)
    | 0 ->
        let channel = open_out sema in
        output_string channel (string_of_int (Unix.getpid ()));
        output_string channel "\n";
        close_out channel;

        (* now loop forever, writing a signature into the
           fifo file.  if you don't have real fifos, change
           sleep time at bottom of loop to like 10 to update
           only every 10 seconds. *)

        while true do
            let channel = open_out fifo in
            let sig' = pick_quote () in
            let sig' = Array.of_list sig' in

            (* trunc to 4 lines *)
            let sig' =
                if Array.length sig' > 4
                then Array.sub sig' 0 4
                else sig' in

            (* trunc long lines *)
            let sig' =
                Array.map
                    (fun line ->
                        if String.length line > 80
                        then String.sub line 0 80
                        else line)
                    sig' in

            (* print sig, with name if present, padded to four lines *)
            begin
                match !name with
                | None | Some "" ->
                    Array.iter
                        (fun line ->
                            output_string channel line;
                            output_string channel "\n")
                        sig'
            end
        end

```

```

    | Some name ->
      output_string channel name;
      for i = 4 downto Array.length sig' do
        output_string channel "\n";
      done;
      Array.iter
        (fun line ->
          output_string channel line;
          output_string channel "\n")
          sig'
end;
close_out channel;

(* Without a microsleep, the reading process doesn't finish
before the writer tries to open it again, which since the
reader exists, succeeds. They end up with multiple
signatures. Sleep a tiny bit between opens to give readers
a chance to finish reading and close our pipe so we can
block when opening it the next time. *)

ignore (Unix.select [] [] [] 0.2) (* sleep 1/5 second *)
done
| _ ->
  exit 0

```

## 17 Sockets

```
open Unix

(* Convert human readable form to 32 bit value *)
let packed_ip = inet_addr_of_string "208.146.240.1" in

let host = gethostbyname "www.oreilly.com" in
let packed_ip = host.h_addr_list.(0) in

(* Convert 32 bit value to ip adress *)
let ip_address = string_of_inet_addr (packed_ip) in

(* Create socket object *)
let sock = socket PF_INET SOCK_STREAM 0 in

(* Get socketname *)
let saddr = getsockname sock ;;
```

### 17.1 Writing a TCP Client

```
(* For real applications you should the SMTP module in Ocamlnet. *)
open Unix

let sock_send sock str =
    let len = String.length str in
    send sock str 0 len []

let sock_recv sock maxlen =
    let str = String.create maxlen in
    let recvlen = recv sock str 0 maxlen [] in
    String.sub str 0 recvlen

let client_sock = socket PF_INET SOCK_STREAM 0 in
let hentry = gethostbyname "coltrane" in
connect client_sock (ADDR_INET (hentry.h_addr_list.(0), 25)) ; (* SMTP *)

sock_recv client_sock 1024 ;

sock_send client_sock "mail from: <pleac@localhost>\n" ;
sock_recv client_sock 1024 ;

sock_send client_sock "rcpt to: <erikd@localhost>\n" ;
sock_recv client_sock 1024;

sock_send client_sock "data\n" ;
sock_recv client_sock 1024 ;

sock_send client_sock "From: Ocaml whiz\nSubject: Ocaml rulez!\n\nYES!\n.\n" ;
sock_recv client_sock 1024 ;

close client_sock ;;
```

## 17.2 Writing a TCP Server

```
(* Writing a TCP Server *)
(* Run this and then telnet <machinename> 1027 *)

#load "unix.cma" ;;
open Unix ;;

let server_sock = socket PF_INET SOCK_STREAM 0 in

(* so we can restart our server quickly *)
setsockopt server_sock SO_REUSEADDR true ;

(* build up my socket address *)
let address = (gethostbyname(gethostname())).h_addr_list.(0) in
bind server_sock (ADDR_INET (address, 1029)) ;

(* Listen on the socket. Max of 10 incoming connections. *)
listen server_sock 10 ;

(* accept and process connections *)
while true do
  let (client_sock, client_addr) = accept server_sock in
  let str = "Hello\n" in
  let len = String.length str in
  let x = send client_sock str 0 len [] in
  shutdown client_sock SHUTDOWN_ALL
done ;;
```

## 17.3 Communicating over TCP

```
#load "unix.cma";;

let () =
  let server_in = Unix.in_channel_of_descr server in
  let server_out = Unix.out_channel_of_descr server in
  output_string server_out "What is your name?\n";
  flush server_out;
  let response = input_line server_in in
  print_endline response

(*-----*)

let () =
  try
    ignore
      (Unix.send server data_to_send 0 (String.length data_to_send) flags)
  with Unix.Unix_error (e, _, _) ->
    Printf.eprintf "Can't send: %s\n%!"
      (Unix.error_message e);
  exit 1

let data_read =
```

```

let data_read = String.create maxlen in
let data_length =
  try
    Unix.recv server data_read 0 maxlen flags
  with Unix.Unix_error (e, _, _) ->
    Printf.eprintf "Can't receive: %s\n%!"
      (Unix.error_message e);
    exit 1 in
String.sub data_read 0 data_length

(*-----*)

let () =
  let read_from, _, _ =
    Unix.select [from_server; to_client] [] [] timeout in
  List.iter
    (fun socket ->
      (* read the pending data from socket *)
      ())
    read_from

(*-----*)

(* Requires OCaml 3.11 or newer. *)
let () =
  try Unix.setsockopt server Unix.TCP_NODELAY true
  with Unix.Unix_error (e, _, _) ->
    Printf.eprintf "Couldn't disable Nagle's algorithm: %s\n%!"
      (Unix.error_message e)

(*-----*)

(* Requires OCaml 3.11 or newer. *)
let () =
  try Unix.setsockopt server Unix.TCP_NODELAY false
  with Unix.Unix_error (e, _, _) ->
    Printf.eprintf "Couldn't enable Nagle's algorithm: %s\n%!"
      (Unix.error_message e)

```

## 17.4 Setting Up a UDP Client

```

#load "unix.cma";;

(* Create a UDP socket. *)
let socket =
  Unix.socket Unix.PF_INET Unix.SOCK_DGRAM
    (Unix.getprotobyname "udp").Unix.p_proto

(*-----*)

(* Send a UDP message. *)
let ipaddr = (Unix.gethostbyname hostname).Unix.h_addr_list.(0)
let portaddr = Unix.ADDR_INET (ipaddr, portno)

```

```

let len = Unix.sendto socket msg 0 (String.length msg) [] portaddr

(*-----*)

(* Receive a UDP message. *)
let msg = String.create maxlen
let len, portaddr = Unix.recvfrom socket msg 0 maxlen []

(*-----*)

#!/usr/bin/ocaml
(* clockdrift - compare another system's clock with this one *)
#load "unix.cma";;

let secs_of_70_years = 2_208_988_800L

let msgbox =
  Unix.socket Unix.PF_INET Unix.SOCK_DGRAM
  (Unix.getprotobyname "udp").Unix.p_proto

let him =
  Unix.ADDR_INET ((Unix.gethostbyname
    (if Array.length Sys.argv > 1
      then Sys.argv.(1)
      else "127.1")).Unix.h_addr_list.(0),
    (Unix.getservbyname "time" "udp").Unix.s_port)

let () = ignore (Unix.sendto msgbox "" 0 0 [] him)

let ptime = String.create 4

let host =
  match Unix.recvfrom msgbox ptime 0 4 [] with
  | _, Unix.ADDR_INET (addr, port) ->
    (Unix.gethostbyaddr addr).Unix.h_name
  | _ -> assert false

let delta =
  Int64.to_float
  (Int64.sub
    (Int64.of_string (Printf.sprintf "0x%02x%02x%02x%02x"
      (int_of_char ptime.[0])
      (int_of_char ptime.[1])
      (int_of_char ptime.[2])
      (int_of_char ptime.[3])))
    secs_of_70_years)
  -. (Unix.time ())

let () =
  Printf.printf "Clock on %s is %d seconds ahead of this one.\n"
    host (int_of_float delta)

```

## 17.5 Setting Up a UDP Server

```
#load "unix.cma";;
let () =
  begin
    try
      Unix.bind socket (Unix.ADDR_INET (Unix.inet_addr_any, server_port));
    with Unix.Unix_error (e, _, _) ->
      Printf.eprintf "Couldn't be a udp server on port %d: %s\n"
        server_port (Unix.error_message e);
      exit 1
    end;
  let him = String.create max_to_read in
  while true do
    ignore (Unix.recvfrom socket him 0 max_to_read []);
    (* do something *)
  done

(*-----*)

#!/usr/bin/ocaml
(* udpqotd - UDP message server *)
#load "unix.cma";;

let maxlen = 1024
let portno = 5151

let sock =
  Unix.socket Unix.PF_INET Unix.SOCK_DGRAM
  (Unix.getprotobyname "udp").Unix.p_proto

let () =
  Unix.bind sock (Unix.ADDR_INET (Unix.inet_addr_any, portno));
  Printf.printf "Awaiting UDP messages on port %d\n%!" portno

let oldmsg = ref "This is the starting message."

let () =
  let newmsg = String.create maxlen in
  while true do
    let newmsg, hishost, sockaddr =
      match Unix.recvfrom sock newmsg 0 maxlen [] with
      | len, (Unix.ADDR_INET (addr, port) as sockaddr) ->
        String.sub newmsg 0 len,
        (Unix.gethostbyaddr addr).Unix.h_name,
        sockaddr
      | _ -> assert false in
    Printf.printf "Client %s said '%s'\n%!" hishost newmsg;
    ignore
      (Unix.sendto sock !oldmsg 0 (String.length !oldmsg) [] sockaddr);
    oldmsg := Printf.sprintf "[%s] %s" hishost newmsg
  done
```

```

(*-----*)

#!/usr/bin/ocaml
(* udpmsg - send a message to the udpqtd server *)
#load "unix.cma";;

let maxlen = 1024
let portno = 5151
let timeout = 5

let server_host, msg =
  match Array.to_list Sys.argv with
  | _ :: head :: tail -> head, String.concat " " tail
  | _ ->
    Printf.eprintf "Usage: %s server_host msg ...\n" Sys.argv.(0);
    exit 1

let sock =
  Unix.socket Unix.PF_INET Unix.SOCK_DGRAM
  (Unix.getprotobyname "udp").Unix.p_proto

let sockaddr =
  let addr = (Unix.gethostbyname server_host).Unix.h_addr_list.(0) in
  Unix.ADDR_INET (addr, portno)

let handle_alarm signal =
  Printf.eprintf "recv from %s timed out after %d seconds.\n"
    server_host timeout;
  exit 1

let () =
  ignore (Unix.sendto sock msg 0 (String.length msg) [] sockaddr);
  Sys.set_signal Sys.sigalrm (Sys.Signal_handle handle_alarm);
  ignore (Unix.alarm timeout);
  let msg = String.create maxlen in
  let msg, hishost =
    match Unix.recvfrom sock msg 0 maxlen [] with
    | len, Unix.ADDR_INET (addr, port) ->
      String.sub msg 0 len,
      (Unix.gethostbyaddr addr).Unix.h_name
    | _ -> assert false in
  ignore (Unix.alarm 0);
  Printf.printf "Server %s responded '%s'\n" hishost msg

```

## 17.6 Using UNIX Domain Sockets

```

#load "unix.cma";;

(* Create a Unix domain socket server - you can also use SOCK_STREAM. *)
let server = Unix.socket Unix.PF_UNIX Unix.SOCK_DGRAM 0
let () = try Unix.unlink "/tmp/mysock" with Unix.Unix_error _ -> ()
let () = Unix.bind server (Unix.ADDR_UNIX "/tmp/mysock")

```

```

(* Create a Unix domain socket client - you can also use SOCK_STREAM. *)
let client = Unix.socket Unix.PF_UNIX Unix.SOCK_DGRAM 0
let () = Unix.connect client (Unix.ADDR_UNIX "/tmp/mysock")

```

## 17.7 Identifying the Other End of a Socket

```

#load "unix.cma";;

(* Get the remote IP address. *)
let () =
  let other_end = Unix.getpeername socket in
  let name_info = Unix.getnameinfo other_end [Unix.NI_NUMERICHOST] in
  let ip_address = name_info.Unix.ni_hostname in
  (* ... *)
  ()

(*-----*)

(* Attempt to determine the remote host name, with forward and reverse
   DNS lookups to detect spoofing. *)
let () =
  let other_end = Unix.getpeername socket in
  let name_info = Unix.getnameinfo other_end [Unix.NI_NUMERICHOST] in
  let actual_ip = name_info.Unix.ni_hostname in
  let claimed_hostname =
    (Unix.gethostbyaddr (Unix.inet_addr_of_string actual_ip))
    .Unix.h_name in
  let name_lookup = Unix.gethostbyname claimed_hostname in
  let resolved_ips =
    Array.to_list (Array.map
      Unix.string_of_inet_addr
      name_lookup.Unix.h_addr_list) in
  (* ... *)
  ()

```

## 17.8 Finding Your Own Name and Address

```

(*-----*)

(*
** Finding Your Own Name and Address.
** The Unix module to the rescue again.
*)

#load "unix.cma" ;;
open Unix ;;

let hostname = gethostname () in
Printf.printf "hostname : %s\n" hostname ;;

(*-----*)

(*

```

```

** Unfortunately there is no easy way of retrieving the
** uname without using Unix.open_process_in.
*)

(*-----*)

let hentry = gethostbyname hostname in
let address = hentry.h_addr_list.(0) in
Printf.printf "address : %s\n" (string_of_inet_addr address) ;;

let hentry = gethostbyaddr address in
Printf.printf "hostname : %s\n" hentry.h_name ;;

```

## 17.9 Closing a Socket After Forking

```

(* Closing a Socket After Forking *)

(*-----*)
shutdown sock SHUTDOWN_RECEIVE ;    (* I/we have stopped reading data *)
shutdown sock SHUTDOWN_SEND ;       (* I/we have stopped writing data *)
shutdown sock SHUTDOWN_ALL ;        (* I/we have stopped us-
ing this socket *)

(*-----*)
(* Using the sock_send and sock_rcv functions from above. *)

sock_send sock "my request\n" ;     (* send some data *)
shutdown sock SHUTDOWN_SEND ;       (* send eof; no more writing *)
let answer = sock_rcv sock 4096 ;;  (* but you can still read *)

```

## 17.10 Writing Bidirectional Clients

```

#!/usr/bin/ocaml
(* biclient - bidirectional forking client *)
#load "unix.cma";;

let host, port =
  match Array.to_list Sys.argv with
  | [_; host; port] -> host, int_of_string port
  | _ -> Printf.eprintf "usage: %s host port\n" Sys.argv.(0); exit 1

let sockaddr =
  let addr = (Unix.gethostbyname host).Unix.h_addr_list.(0) in
  Unix.ADDR_INET (addr, port)

let () =
  let socket = Unix.socket Unix.PF_INET Unix.SOCK_STREAM 0 in
  Unix.connect socket sockaddr;
  Printf.eprintf "[Connected to %s:%d]\n%! " host port;

  (* split the program into two processes, identical twins *)
  match Unix.fork () with
  | 0 ->

```

```

(* child copies standard input to the socket *)
let output = Unix.out_channel_of_descr socket in
while true do
  let line = input_line stdin in
  output_string output line;
  output_string output "\n";
  flush output
done
| kidpid ->
(* parent copies the socket to standard output *)
let input = Unix.in_channel_of_descr socket in
try
  while true do
    let line = input_line input in
    output_string stdout line;
    output_string stdout "\n";
    flush stdout
  done
with End_of_file ->
  Unix.kill kidpid Sys.sigterm

let () = exit 0

```

## 17.11 Forking Servers

```

(* set up the socket SERVER, bind and listen ... *)
#load "unix.cma";;

let rec reaper signal =
  try while true do ignore (Unix.waitpid [Unix.WNOHANG] (-1)) done
  with Unix.Unix_error (Unix.ECHILD, _, _) -> ();
  Sys.set_signal Sys.sigchld (Sys.Signal_handle reaper)

let () =
  Sys.set_signal Sys.sigchld (Sys.Signal_handle reaper)

let () =
  while true do
    try
      let (client, addr) = Unix.accept server in
      let pid = Unix.fork () in
      if pid = 0 then
        (* parent *)
        begin
          Unix.close server;
          (* no use to child *)
          (* ... do something *)
          exit 0
          (* child leaves *)
        end
      else
        begin
          Unix.close client
          (* no use to parent *)
        end
      end
    with Unix.Unix_error (Unix.EINTR, _, _) -> ()
  done

```

## 17.12 Pre-Forking Servers

```
#!/usr/bin/ocaml
(* preforker - server who forks first *)
#load "unix.cma";;

(* global variables *)
let prefork = 5
let max_clients_per_child = 5
module PidSet = Set.Make(struct type t = int let compare = compare end)
let children = ref PidSet.empty

(* takes care of dead children *)
let rec reaper _ =
  Sys.set_signal Sys.sigchild (Sys.Signal_handle reaper);
  match Unix.wait ()
  with (pid, _) -> children := PidSet.remove pid !children

(* signal handler for SIGINT *)
let rec huntsman _ =
  (* we're going to kill our children *)
  Sys.set_signal Sys.sigchild Sys.Signal_ignore;
  PidSet.iter
    (fun pid ->
      try Unix.kill Sys.sigint pid with Unix.Unix_error _ -> ())
    !children;
  (* clean up with dignity *)
  exit 0

let make_new_child server =
  (* block signal for fork *)
  let sigset = [Sys.sigint] in
  ignore (Unix.sigprocmask Unix.SIG_BLOCK sigset);

  match Unix.fork () with
  | 0 ->
    (* Child can *not* return from this subroutine. *)
    (* make SIGINT kill us as it did before *)
    Sys.set_signal Sys.sigint Sys.Signal_default;

    (* unblock signals *)
    ignore (Unix.sigprocmask Unix.SIG_UNBLOCK sigset);

    (* handle connections until we've reached max_clients_per_child *)
    for i = 1 to max_clients_per_child do
      let (client, _) = Unix.accept server in
        (* do something with the connection *)
        ()
    done;

    (* tidy up gracefully and finish *)

    (* this exit is VERY important, otherwise the child will become
```

```

        a producer of more and more children, forking yourself into
        process death. *)
    exit 0
  | pid ->
    (* Parent records the child's birth and returns. *)
    ignore (Unix.sigprocmask Unix.SIG_UNBLOCK sigset);
    children := PidSet.add pid !children

let () =
  (* establish SERVER socket, bind and listen. *)
  let server = Unix.socket Unix.PF_INET Unix.SOCK_STREAM 0 in
  Unix.setsockopt server Unix.SO_REUSEADDR true;
  Unix.bind server (Unix.ADDR_INET (Unix.inet_addr_any, 6969));
  Unix.listen server 10;

  (* Fork off our children. *)
  for i = 1 to prefork do
    make_new_child server
  done;

  (* Install signal handlers. *)
  Sys.set_signal Sys.sigchild (Sys.Signal_handle reaper);
  Sys.set_signal Sys.sigint (Sys.Signal_handle huntsman);

  (* And maintain the population. *)
  while true do
    (* wait for a signal (i.e., child's death) *)
    Unix.pause ();
    for i = (PidSet.cardinal !children) to (prefork - 1) do
      (* top up the child pool *)
      make_new_child server
    done
  done

```

### 17.13 Non-Forking Servers

```

#!/usr/bin/ocaml
(* nonforker - server who multiplexes without forking *)
#load "unix.cma";;

let port = 1685 (* change this at will *)

(* Listen to port. *)
let server = Unix.socket Unix.PF_INET Unix.SOCK_STREAM 0
let () =
  Unix.setsockopt server Unix.SO_REUSEADDR true;
  Unix.bind server (Unix.ADDR_INET (Unix.inet_addr_any, port));
  Unix.listen server 10;
  Unix.set_nonblock server

module FDSet =
  Set.Make(struct type t = Unix.file_descr let compare = compare end)
let clients = ref (FDSet.singleton server)

```

```

(* begin with empty buffers *)
let inbuffer = Hashtbl.create 0
let outbuffer = Hashtbl.create 0
let ready = Hashtbl.create 0

let buffer_size = 8192
let buffer = String.make buffer_size '\000'

(* handle deals with all pending requests for client *)
let handle client requests =
  (* requests are in ready[client] *)
  (* send output to outbuffer[client] *)
  List.iter
    (fun request ->
      (* request is the text of the request *)
      let data = Printf.sprintf "You said: %s\n" request in
      (* put text of reply into outbuffer[client] *)
      Hashtbl.replace outbuffer client
        (try Hashtbl.find outbuffer client ^ data
         with Not_found -> data))
    requests

(* Main loop: check reads/accepts, check writes, check ready to process *)
let () =
  while true do
    (* check for new information on the connections we have *)

    let (can_read, _, _) =
      Unix.select (FDSet.elements !clients) [] [] 1.0 in
    List.iter
      (fun client ->
        if client = server
        then
          begin
            (* accept a new connection *)
            let (client, addr) = Unix.accept server in
              clients := FDSet.add client !clients;
              Unix.set_nonblock client
            end
          else
            begin
              (* read data *)
              let chars_read =
                try
                  Some (Unix.read client buffer 0 buffer_size)
                with Unix.Unix_error (error, _, _) ->
                  prerr_endline (Unix.error_message error);
                  None in

              match chars_read with
              | None | Some 0 ->
                (* This would be the end of file, so close the client *)

```

```

        Hashtbl.remove inbuffer client;
        Hashtbl.remove outbuffer client;
        Hashtbl.remove ready client;

        clients := FDSets.remove client !clients;
        Unix.close client

    | Some chars_read ->
        let data = String.sub buffer 0 chars_read in
        Hashtbl.replace inbuffer client
            (try Hashtbl.find inbuffer client ^ data
             with Not_found -> data);

        (* test whether the data in the buffer or the data we *)
        (* just read means there is a complete request wait-
ing *)

        (* to be fulfilled. If there is, set ready[client] *)
        (* to the requests waiting to be fulfilled. *)
        try
            while true do
                let data = Hashtbl.find inbuffer client in
                let index = String.index data '\n' in
                Hashtbl.replace inbuffer client
                    (String.sub data
                     (index + 1)
                     (String.length data - index - 1));
                Hashtbl.replace ready client
                    ((try Hashtbl.find ready client
                     with Not_found -> [])
                     @ [String.sub data 0 index])
                done
            with Not_found -> ()
        end)
    can_read;

    (* Any complete requests to process? *)
    Hashtbl.iter handle ready;
    Hashtbl.clear ready;

    (* Buffers to flush? *)
    let (_, can_write, _) =
        Unix.select [] (FDSets.elements !clients) [] 1.0 in
    (* Skip client if we have nothing to say *)
    let can_write =
        List.filter (Hashtbl.mem outbuffer) can_write in
    List.iter
        (fun client ->
            let data = Hashtbl.find outbuffer client in
            let chars_written =
                try
                    Some (Unix.single_write client data 0 (String.length data))
                with
                    | Unix.Unix_error (Unix.EAGAIN, _, _)

```

```

    | Unix.Unix_error (Unix.EWOULDBLOCK, _, _) ->
      prerr_endline "I was told I could write, but I can't.";
      Some 0
    | Unix.Unix_error (error, _, _) ->
      prerr_endline (Unix.error_message error);
      None in

  match chars_written with
  | Some chars_written ->
    if chars_written = String.length data
    then Hashtbl.remove outbuffer client
    else Hashtbl.replace outbuffer client
      (String.sub data chars_written
       (String.length data - chars_written))
  | None ->
    (* Couldn't write all the data, and it wasn't because *)
    (* it would have blocked. Shutdown and move on. *)
    Hashtbl.remove inbuffer client;
    Hashtbl.remove outbuffer client;
    Hashtbl.remove ready client;

    clients := FDSet.remove client !clients;
    Unix.close client)
  can_write;

  let (_, _, has_exception) =
    Unix.select [] [] (FDSet.elements !clients) 0.0 in
  List.iter
    (fun client ->
      (* Deal with out-of-band data here, if you want to. *)
      ())
    has_exception;
done

```

## 17.14 Writing a Multi-Homed Server

```

#load "unix.cma";;

let server =
  Unix.socket Unix.PF_INET Unix.SOCK_STREAM
  (Unix.getprotobyname "tcp").Unix.p_proto

let () =
  Unix.setsockopt server Unix.SO_REUSEADDR true;
  Unix.bind server (Unix.ADDR_INET (Unix.inet_addr_any, server_port));
  Unix.listen server 10;

  (* accept loop *)
  while true do
    let client, sockaddr = Unix.accept server in
      match Unix.getsockname client with
      | Unix.ADDR_INET (addr, port) ->
        print_endline (Unix.string_of_inet_addr addr)

```

```

    | _ -> assert false
done

(*-----*)

#load "unix.cma";;

let port = 4269 (* port to bind to *)
let host = "specific.host.com" (* virtual host to listen on *)

let server =
  Unix.socket Unix.PF_INET Unix.SOCK_STREAM
  (Unix.getprotobyname "tcp").Unix.p_proto

let () =
  let addr = (Unix.gethostbyname host).Unix.h_addr_list.(0) in
  Unix.bind server (Unix.ADDR_INET (addr, port));
  Unix.listen server 10;
  while true do
    let client, sockaddr = Unix.accept server in
      (* ... *)
    ()
  done

```

## 17.15 Making a Daemon Server

```

#load "unix.cma";;

let () =
  (* for the paranoid *)
  (* Unix.handle_unix_error Unix.chroot "/var/daemon"; *)

  (* fork and let parent exit *)
  let pid = Unix.fork () in
  if pid > 0 then exit 0;

  (* create a new session and abandon the controlling process *)
  ignore (Unix.setsid ())

  (* flag indicating it is time to exit *)
  let time_to_die = ref false

  (* trap fatal signals *)
  let () =
    let signal_handler _ = time_to_die := true in
    List.iter
      (fun signal ->
        Sys.set_signal signal (Sys.Signal_handle signal_handler))
      [Sys.sigint; Sys.sigterm; Sys.sighup]
    (* trap or ignore Sys.sigpipe *)

  (* server loop *)
  let () =

```

```

while not !time_to_die do
  (* ... *)
  ()
done

```

## 17.16 Restarting a Server on Demand

```

#load "unix.cma";;

let self = "/usr/bin/ocaml"
let args = self :: Array.to_list Sys.argv

let phoenix _ =
  (* close all your connections, kill your children, and *)
  (* generally prepare to be reincarnated with dignity. *)
  try
    ignore (Unix.sigprocmask Unix.SIG_UNBLOCK [Sys.sighup]);
    Unix.execv self (Array.of_list args)
  with Unix.Unix_error (e, _, _) ->
    Printf.eprintf "Couldn't restart: %s\n%!"
      (Unix.error_message e)

let () =
  Sys.set_signal Sys.sighup (Sys.Signal_handle phoenix)

(*-----*)

(* This recipe uses the Ocaml-Syck YAML parser available at:
   http://ocaml-syck.sourceforge.net/ *)

#directory "+yaml";;
#load "yaml.cma";;
#load "unix.cma";;

let yaml_parser = YamlParser.make ()

let config_file = "/usr/local/etc/myprog/server_conf.yaml"
let config = ref (YamlNode.SCALAR ("", ""))

let read_config _ =
  let in_channel = open_in config_file in
  let lines = ref [] in
  try
    while true do
      let line = input_line in_channel in
      lines := line :: !lines
    done
  with End_of_file ->
    close_in in_channel;
    config :=
      YamlParser.parse_string yaml_parser
        (String.concat "\n" (List.rev !lines))

```

```

let () =
  read_config ();
  Sys.set_signal Sys.sighup (Sys.Signal_handle read_config)

```

### 17.17 Program: backsniff

```

Oct  4 11:01:16 pedro sniffer: Connection from 10.0.0.4 to 10.0.0.1:echo

(*-----*)

echo stream tcp nowait nobody /usr/bin/ocaml ocaml /path/to/backsniff.ml

(*-----*)

(* backsniff - log attempts to connect to particular ports *)
#load "unix.cma";;

(* This recipe uses syslog-ocaml, which is available at:
   http://www.cs.cmu.edu/~ecc/software.html *)
#directory "+syslog";;
#load "syslog.cma";;

(* identify my port and address *)
let sockname =
  try Unix.getsockname Unix.stdin
  with Unix.Unix_error (e, _, _) ->
    Printf.eprintf "Couldn't identify myself: %s\n%!"
      (Unix.error_message e);
    exit 1
let iaddr, port =
  match sockname with
  | Unix.ADDR_INET (iaddr, port) -> iaddr, port
  | _ -> assert false
let my_address = Unix.string_of_inet_addr iaddr

(* get a name for the service *)
let service =
  try (Unix.getservbyport port "tcp").Unix.s_name
  with Not_found -> string_of_int port

(* now identify remote address *)
let sockname =
  try Unix.getpeername Unix.stdin
  with Unix.Unix_error (e, _, _) ->
    Printf.eprintf "Couldn't identify other end: %s\n%!"
      (Unix.error_message e);
    exit 1
let iaddr, port =
  match sockname with
  | Unix.ADDR_INET (iaddr, port) -> iaddr, port
  | _ -> assert false
let ex_address = Unix.string_of_inet_addr iaddr

```

```

(* and log the information *)
let () =
  let log = Syslog.openlog ~flags:[] ~facility:'LOG_DAEMON "sniffer" in
  Syslog.syslog log 'LOG_NOTICE
    (Printf.sprintf "Connection from %s to %s:%s\n"
      ex_address my_address service);
  Syslog.closelog log;
  exit 0

```

## 17.18 Program: fwdport

```

#!/usr/bin/ocaml
(* fwdport -- act as proxy forwarder for dedicated services *)

#load "str.cma";;
#load "unix.cma";;

let children = Hashtbl.create 0 (* hash of outstanding child processes *)
let remote = ref "" (* whom we connect to on the outside *)
let local = ref "" (* where we listen to on the inside *)
let service = ref "" (* our service name or port number *)
let proxy_server = ref Unix.stdin (* the socket we accept() from *)

(* process command line switches *)
let check_args () =
  Arg.parse
  [
    "-r", Arg.Set_string remote, "Remote host";
    "-remote", Arg.Set_string remote, "Remote host";
    "-l", Arg.Set_string local, "Local interface";
    "-local", Arg.Set_string local, "Local interface";
    "-s", Arg.Set_string service, "Service";
    "-service", Arg.Set_string service, "Service";
  ]
  (fun s ->
    raise (Arg.Bad (Printf.sprintf "unexpected argument '%s'" s))
    (Printf.sprintf "usage: %s [ -remote host ] [ -local interface ] [ -service service ]" Sys.argv.(0)));
  if !remote = ""
  then (prerr_endline "Need remote"; exit 1);
  if !local = "" && !service = ""
  then (prerr_endline "Need local or service"; exit 1);
  if !local = ""
  then local := "localhost"

let parse_host host =
  match Str.split (Str.regexp ":") host with
  | [] -> "", ""
  | host :: [] -> host, ""
  | host :: service :: _ -> host, service

let resolve_host host =

```

```

try (Unix.gethostbyname host).Unix.h_addr_list.(0)
with Not_found ->
  Printf.eprintf "Host not found: %s\n" host;
  exit 1

let resolve_service service =
  try int_of_string service
  with Failure _ ->
    try (Unix.getservbyname service "tcp").Unix.s_port
    with Not_found ->
      Printf.eprintf "Service not found: %s\n" service;
      exit 1

(* begin our server *)
let start_proxy () =
  try
    let proto = (Unix.getprotobyname "tcp").Unix.p_proto in
    let addr, port =
      match parse_host (!local ^ ":" ^ !service) with
      | host, service ->
        (resolve_host host,
         resolve_service service) in
    proxy_server := Unix.socket Unix.PF_INET Unix.SOCK_STREAM proto;
    Unix.setsockopt !proxy_server Unix.SO_REUSEADDR true;
    Unix.bind !proxy_server (Unix.ADDR_INET (addr, port));
    Unix.listen !proxy_server 128;
    Printf.printf "[Proxy server on %s initialized.]\n%!";
    (if !local <> "" then !local else !service)
  with Unix.Unix_error (e, _, _) ->
    Printf.eprintf "Can't create proxy server: %s\n%!";
    (Unix.error_message e);
    exit 1

(* helper function to produce a nice string in the form HOST:PORT *)
let peerinfo sock =
  match Unix.getpeername sock with
  | Unix.ADDR_INET (addr, port) ->
    let hostinfo = Unix.gethostbyaddr addr in
    Printf.sprintf "%s:%d" hostinfo.Unix.h_name port
  | _ -> assert false

(* somebody just died. keep harvesting the dead until *)
(* we run out of them. check how long they ran. *)
let rec reaper signal =
  begin
    let result =
      try Some (Unix.waitpid [Unix.WNOHANG] (-1))
      with Unix.Unix_error (Unix.ECHILD, _, _) -> None in
    match result with
    | Some (child, status) when Hashtbl.mem children child ->
      let start = Hashtbl.find children child in
      let runtime = Unix.time () -. start in
      Printf.printf "Child %d ran %dm%fs\n%!"

```

```

        child
        (int_of_float (runtime /. 60.))
        (mod_float runtime 60.);
        Hashtbl.remove children child;
        reaper signal
    | Some (child, status) ->
        Printf.printf "Bizarre kid %d exited with %s\n%!"
            child
            (match status with
            | Unix.WEXITED code ->
                "code " ^ string_of_int code
            | Unix.WSTOPPED signal
            | Unix.WSIGNALED signal ->
                "signal " ^ string_of_int signal);
        reaper signal
    | None -> ()
end;
(* If I had to choose between System V and 4.2, I'd resign. *)
(* --Peter Honeyman *)
Sys.set_signal Sys.sigchld (Sys.Signal_handle reaper)

let service_clients () =
    (* harvest the moribund *)
    Sys.set_signal Sys.sigchld (Sys.Signal_handle reaper);

    (* an accepted connection here means someone inside wants out *)
    while true do
        try
            begin
                let local_client = fst (Unix.accept !proxy_server) in
                let lc_info = peerinfo local_client in
                Printf.printf "[Connect from %s]\n%!" lc_info;

                let proto = (Unix.getprotobyname "tcp").Unix.p_proto in
                let addr, port =
                    match parse_host (!remote ^ ":" ^ !service) with
                    | host, service ->
                        (resolve_host host,
                         resolve_service service) in
                Printf.printf "[Connecting to %s...%]" !remote;
                let remote_server = Unix.socket Unix.PF_INET Unix.SOCK_STREAM proto in
                Unix.connect remote_server (Unix.ADDR_INET (addr, port));
                Printf.printf "done]\n%!";

                let local_in = Unix.in_channel_of_descr local_client in
                let local_out = Unix.out_channel_of_descr local_client in
                let remote_in = Unix.in_channel_of_descr remote_server in
                let remote_out = Unix.out_channel_of_descr remote_server in

                match Unix.fork () with
                | 0 ->
                    (* at this point, we are the forked child process dedi-
```

```

cated *)
    (* to the incoming client.  but we want a twin to make i/o *)
    (* easier. *)

    Unix.close !proxy_server;  (* no use to slave *)

    (* now each twin sits around and ferries lines of data. *)
    (* see how simple the algorithm is when you can have *)
    (* multiple threads of control? *)

    (match Unix.fork () with
    | 0 ->
        (* this is the fork's child, the master's grand-
child *)
        (try
            while true do
                let line = input_line local_in in
                Printf.fprintf remote_out "%s\n%!" line
            done
            with End_of_file ->
                (* kill my twin cause we're done *)
                Unix.kill (Unix.getppid ()) Sys.sigterm)
    | kidpid ->
        (* this is the fork's parent, the master's child *)
        (try
            while true do
                let line = input_line remote_in in
                Printf.fprintf local_out "%s\n%!" line
            done
            with End_of_file ->
                (* kill my twin cause we're done *)
                Unix.kill kidpid Sys.sigterm));

    exit 0  (* whoever's still alive bites it *)

    | kidpid ->
        (* remember his start time *)
        Hashtbl.replace children kidpid (Unix.time ());
        Unix.close remote_server;  (* no use to master *)
        Unix.close local_client;  (* likewise *)
    end
    with Unix.Unix_error (Unix.EINTR, "accept", _) -> ()
done

let () =
    check_args ();          (* processing switches *)
    start_proxy ();        (* launch our own server *)
    service_clients ();    (* wait for incoming *)
    prerr_endline "NOT REACHED"; (* you can't get here from there *)
    exit 1

```

## 18 Internet Services

### 18.1 Simple DNS Lookups

```
#load "unix.cma";;

let () =
  try
    let addresses = Unix.gethostbyname name in
    let addresses =
      Array.map Unix.string_of_inet_addr addresses.Unix.h_addr_list in
    (* addresses is an array of IP addresses *)
    Array.iter print_endline addresses
  with Not_found ->
    Printf.printf "Can't resolve %s\n" name

(*-----*)

let () =
  try
    let host = Unix.gethostbyaddr (Unix.inet_addr_of_string address) in
    let name = host.Unix.h_name in
    (* name is the hostname ("www.perl.com") *)
    print_endline name
  with Not_found ->
    Printf.printf "Can't resolve %s\n" address

(*-----*)

let () =
  try
    let host = Unix.gethostbyaddr (Unix.inet_addr_of_string address) in
    let name = host.Unix.h_name in
    try
      let addresses = Unix.gethostbyname name in
      let addresses =
        Array.map Unix.string_of_inet_addr addresses.Unix.h_addr_list in
      Array.iter print_endline addresses;
      let found = List.mem address (Array.to_list addresses) in
      print_endline (if found then "found" else "not found")
    with Not_found ->
      Printf.printf "Can't look up %s\n" name
  with Not_found ->
    Printf.printf "Can't look up %s\n" address

(*-----*)

#!/usr/bin/ocaml
(* mxhost - find mx exchangers for a host *)

(* Though there is an experimental new DNS resolver for OCaml called
Netdns, it does not yet support resolving MX records. For now, we'll
use Net::DNS through perl4caml until a better solution is available.
```

```

*)
#directory "+perl";;
#load "perl4caml.cma";;
let _ = Perl.eval "use Net::DNS"

let host = Sys.argv.(1)
let res = Perl.call_class_method "Net::DNS::Resolver" "new" []
let mx = Perl.call_array ~fn:"mx" [res; Perl.sv_of_string host]
let () =
  if mx = [] then
    Printf.eprintf "Can't find MX records for %s (%s)\n"
      host (Perl.string_of_sv (Perl.call_method res "errorstring" []))

let () =
  List.iter
    (fun record ->
      let preference = Perl.call_method record "preference" [] in
      let exchange = Perl.call_method record "exchange" [] in
      Printf.printf "%s %s\n"
        (Perl.string_of_sv preference)
        (Perl.string_of_sv exchange))
    mx

(*-----*)

#!/usr/bin/ocaml
(* hostaddr - canonize name and show addresses *)
#load "unix.cma";;
let name = Sys.argv.(1)
let hent = Unix.gethostbyname name
let () =
  Printf.printf "%s => %s\n"
    hent.Unix.h_name (* in case different *)
    (String.concat " "
      (Array.to_list
        (Array.map
          Unix.string_of_inet_addr
          hent.Unix.h_addr_list)))

```

## 18.2 Being an FTP Client

(\* The Netclient package from Ocamlnet provides an event-driven FTP client. This client does not currently support uploading.

Ocamlnet is available here:  
<http://projects.camlcity.org/projects/ocamlnet.html>

This recipe assumes it has been installed with findlib. \*)

```

#use "topfind";;
#require "netclient";;

(* Create an FTP client instance. *)

```

```

let ftp = new Ftp_client.ftp_client ()

(* Build and execute a chain of FTP methods. *)
let () =
  ftp#add (new Ftp_client.connect_method ~host:"127.0.0.1" ());
  ftp#add (new Ftp_client.login_method
    ~user:"anonymous"
    ~get_password:(fun () -> "user@example.com")
    ~get_account:(fun () -> "anonymous") ());
  ftp#add (new Ftp_client.walk_method ('Dir "/pub"));
  let ch = new Netchannels.output_channel (open_out "output.txt") in
  ftp#add (new Ftp_client.get_method
    ~file:(`Verbatim "index.txt")
    ~representation:`Image
    ~store:(fun _ -> `File_structure ch) ());
  ftp#run ()

(*-----*)

(* If an error occurs, it will be exposed by the "state" property. *)
let () =
  match ftp#state with
  | `Error (Ftp_client.FTP_error (Unix.Unix_error (e, _, _))) ->
    Printf.eprintf "Error: %s\n%!"
      (Unix.error_message e)
  | _ -> ()

(*-----*)

(* To determine the current working directory, send invoke the 'PWD
  command and inspect the result in a callback. *)
let () =
  ftp#add (new Ftp_client.invoke_method
    ~command:`PWD
    ~process_result:(fun state (code, message) ->
      Printf.printf
        "I'm in the directory %s\n%!"
        message) ());

(*-----*)

(* Use mkdir_method and rmdir_method to make and remove directories from
  the remote server. Use the optional ~onerror argument to specify an
  error handler. *)
let () =
  ftp#add
    ~onerror:(fun e ->
      Printf.eprintf "Can't create /ocaml: %s\n%!"
        (Printexc.to_string e)
      (new Ftp_client.mkdir_method (`Verbatim "/pub/ocaml")))

(*-----*)

```

```

(* Use a list_method to get a list of files in a remote directory. *)
let () =
  let buffer = Buffer.create 256 in
  let ch = new Netchannels.output_buffer buffer in
  ftp#add
    ~onsuccess:(fun () -> print_endline (Buffer.contents buffer))
    ~onerror:(fun e ->
      Printf.eprintf "Can't get a list of files in /pub: %s\n%!"
        (Printexc.to_string e))
    (new Ftp_client.list_method
      ~dir:(`Verbatim "/pub")
      ~representation:`Image
      ~store:(fun _ -> `File_structure ch) ())

(*-----*)

(* Use 'QUIT followed by ftp#abort to close the connection and exit
the event loop. *)
let () =
  ftp#add (new Ftp_client.invoke_method
    ~command:`QUIT
    ~process_result:(fun _ _ -> ftp#abort ()) ())

```

### 18.3 Sending Mail

```

(* Use Netsendmail, part of the Netstring package that comes with
Ocamlnet, to send mail through a command-line mailer program. *)

#use "topfind";;
#require "netstring";;

let () =
  Netsendmail.sendmail
    ~mailer:"/usr/sbin/sendmail" (* defaults to "/usr/lib/sendmail" *)
    (Netsendmail.compose
      ~from_addr:(from_name, from_address)
      ~to_addrs:[(to_name, to_address)]
      ~subject:subject
      body)

(*-----*)

(* You can also open a pipe directly to sendmail. *)

#load "unix.cma";;

let () =
  let sendmail =
    Unix.open_process_out "/usr/lib/sendmail -oi -t -odq" in
  output_string sendmail "\
From: User Originating Mail <me@host>
To: Final Destination <you@otherhost>
Subject: A relevant subject line

```

```

Body of the message goes here, in as many lines as you like.
";
  ignore (Unix.close_process_out sendmail)

```

## 18.4 Reading and Posting Usenet News Messages

```

(* There is no NNTP library available for OCaml. With a little
preparation, we can easily use the one that comes with Perl
using perl4caml (http://merjis.com/developers/perl4caml) *)

```

```

#directory "+perl";
#load "perl4caml.cma";

module NNTP = struct
  open Perl
  let _ = eval "use Net::NNTP"

  (* Returned by "list" method so that newsgroups stay sorted. *)
  module GroupMap = Map.Make(String)

  (* Wrapper for Net::NNTP class. *)
  class nntp host =
    let nntp =
      call_class_method "Net::NNTP" "new" [sv_of_string host] in

    (* Raise a Failure exception if we couldn't connect. *)
    let () =
      if sv_is_undef nntp
      then failwith (string_of_sv (eval "$!")) in

    (* Helper function to transform nullable string arrays to OCaml. *)
    let maybe_string_list sv =
      if sv_is_undef sv
      then raise Not_found
      else List.map string_of_sv (list_of_av (deref_array sv)) in

  object (self)
    val nntp = nntp

    method group name =
      match call_method_array nntp "group" [sv_of_string name] with
      | [narticles; first; last; name] ->
          (int_of_sv narticles, int_of_sv first,
           int_of_sv last, string_of_sv name)
      | _ -> raise Not_found

    method head msgid =
      maybe_string_list (call_method nntp "head" [sv_of_int msgid])

    method body msgid =
      maybe_string_list (call_method nntp "body" [sv_of_int msgid])
  end
end

```

```

method article msgid =
  maybe_string_list (call_method nntp "article" [sv_of_int msgid])

method postok () =
  bool_of_sv (call_method nntp "postok" [])

method post lines =
  let lines = List.map sv_of_string lines in
  if (sv_is_undef (call_method nntp "post" lines))
  then failwith (string_of_sv (eval "$!"))

method list () =
  let hv = deref_hash (call_method nntp "list" []) in
  let map = ref GroupMap.empty in
  List.iter
    (fun (name, info) ->
      map :=
        GroupMap.add
          name
          (match list_of_av (deref_array info) with
           | [last; first; flags] ->
             (int_of_sv last, int_of_sv first,
              string_of_sv flags)
           | _ -> assert false)
          !map)
    (assoc_of_hv hv);
  !map

method quit () =
  ignore (call_method nntp "quit" [])
end
end

(*-----*)

(* Connect to an NNTP server by creating an "nntp" object. *)
let server =
  try new NNTP.nntp "news.west.cox.net"
  with Failure s ->
    Printf.eprintf "Can't connect to news server: %s\n" s;
    exit 1

(*-----*)

(* Select a newsgroup and retrieve its stats. *)
let (narticles, first, last, name) =
  try server#group "misc.test"
  with Not_found ->
    Printf.eprintf "Can't select misc.test\n";
    exit 1

(*-----*)

```

```

(* Get the headers from the last article. *)
let headers =
  try server#head last
  with Not_found ->
    Printf.eprintf "Can't get headers from article %d in %s\n"
      last name;
    exit 1

(*-----*)

(* Get the body from the last article. *)
let body =
  try server#head last
  with Not_found ->
    Printf.eprintf "Can't get body from article %d in %s\n"
      last name;
    exit 1

(*-----*)

(* Get the headers and body from the last article. *)
let article =
  try server#head last
  with Not_found ->
    Printf.eprintf "Can't get article from article %d in %s\n"
      last name;
    exit 1

(*-----*)

(* Determine if posting is allowed with this server. *)
let () =
  if not (server#postok ())
  then Printf.eprintf "Server didn't tell me I could post.\n"

(*-----*)

(* Post a message. *)
let () =
  begin
    try server#post lines
    with Failure s ->
      Printf.eprintf "Can't post: %s\n" s;
      exit 1
  end

(*-----*)

(* Get the complete list of newsgroups. *)
let () =
  let groupmap = server#list () in
  NNTP.GroupMap.iter
    (fun group (last, first, flags) ->

```

```

    if flags = "y"
    then (* I can post to [group] *) ()
groupmap

```

## 18.5 Reading Mail with POP3

```

(* Use Netpop, which is part of Ocamlnet. *)
#use "topfind";;
#require "pop";;

(* To create a Netpop client, you need to look up the server address
   and build a network connection first. Netpop uses wrappers called
   Netchannels to abstract the input and output channels. *)
let inet_addr =
  (Unix.gethostbyname mail_server).Unix.h_addr_list.(0)
let addr = Unix.ADDR_INET (inet_addr, Netpop.tcp_port)
let ic, oc = Unix.open_connection addr
let pop =
  new Netpop.client
    (new Netchannels.input_channel ic)
    (new Netchannels.output_channel oc)
let () =
  pop#user username;
  pop#pass password

(* Messages are retrieved as a hashtable from message IDs to tuples,
   each tuple containing the message size in bytes and a string of
   server-specific extension data. *)
let messages = pop#list ()
let () =
  Hashtbl.iter
    (fun msgid (size, ext) ->
      let message = pop#retr msgid in
      (* message is a Netchannels.in_obj_channel *)
      pop#dele msgid)
    messages

(*-----*)

(* Use pop#apop instead of pop#user/pop#pass to avoid sending passwords
   in plaintext across the network. *)
let () = pop#apop username password

(*-----*)

(* Get a message by number and print it to the console. *)
let () =
  Printf.printf "Retrieving %d : %!" msgnum;
  try
    let message = pop#retr msgnum in
    print_newline ();
    print_endline
      (Netchannels.string_of_in_obj_channel message)
  with _ -> ()

```

```

with Netpop.Err_status e ->
  Printf.printf "failed (%s)\n%!" e

(*-----*)

(* Gracefully tear down the connection. *)
let () =
  pop#quit ();
  Unix.shutdown_connection ic;
  close_out oc

```

## 18.6 Simulating Telnet from a Program

```

(* To simulate a Telnet client with OCaml, you can use the
   Telnet_client module from Ocamlnet's "netclient" package.

```

```

This module is written in an asynchronous style, so you
will need to create event handlers to process the Telnet
events that occur: data, end of file, timeout, and the
sending and receiving of options (also known as "do",
"don't", "will", and "won't". *)

```

```

#use "topfind";;
#require "netclient";;

open Telnet_client

(* This class wraps the Telnet session for convenience in
   defining event handlers and chaining them together. *)
class session ~host ~port ~username ~password ~prompt ~timeout =
object (self)
  (* Telnet_client.telnet_session instance to wrap. *)
  val telnet = new telnet_session

  (* Initial on-data handler, which will be redefined later. *)
  val mutable process = fun _ -> ()

  (* Initialize the Telnet session. *)
  initializer
    telnet#set_connection (Telnet_connect (host, port));
    telnet#set_options {connection_timeout=timeout;
                        verbose_connection=false;
                        verbose_input=false;
                        verbose_output=false};
    telnet#set_callback self#on_input;
    telnet#set_exception_handler self#on_exception;
    telnet#attach ();
    process <- self#start

  (* Build an input callback that checks for a regular
     expression match in the input and calls a callback
     function if the match is positive. *)
  method waitfor pat cb =

```

```

let rex = Pcre.regexp pat in
fun data -> if Pcre.pmatch ~rex data then cb data

(* Enqueue a line of data and flush the output queue. *)
method write data =
  Queue.add (Telnet_data data) telnet#output_queue;
  Queue.add (Telnet_data "\n") telnet#output_queue;
  telnet#update ()

(* Handle first input: wait for a login prompt and then
   invoke self#send_username to send the username. *)
method start =
  self#waitfor "ogin:" self#send_username

(* Send the username and wait for the password prompt. *)
method send_username data =
  self#write username;
  process <- self#waitfor "assword:" self#send_password

(* Send the password and wait to see if we succeeded. *)
method send_password data =
  self#write password;
  process <- self#verify_login

(* Determine if the login was a success or a failure.
   Abort with an exception on failure; call self#logged_in
   on success. *)
method verify_login data =
  if Pcre.pmatch ~pat:"incorrect" data
  then failwith "Login failed"
  else if Pcre.pmatch ~pat:"^\s*$" data
  then () (* ignore blank lines *)
  else self#logged_in data

(* Logged in successfully. Wait for a prompt if necessary
   and call self#run_ls to send the first command. *)
method logged_in data =
  process <- self#waitfor prompt self#run_ls;
  self#waitfor prompt self#run_ls data

(* Do a directory listing and wait for results. *)
method run_ls data =
  self#write "/bin/ls -l";
  process <- self#gather_files

(* This variable will buffer the results of the "ls" command. *)
val mutable files = ""

(* Buffer the filenames printed out from the "ls" command and
   print them out once we get a prompt. *)
method gather_files data =
  if Pcre.pmatch ~pat:prompt data
  then

```

```

begin
  files <- Pcre.replace ~pat:"~/bin/ls -l\\s*" files;
  Printf.printf
    "Files: %s\n%!"
    (String.concat ", "
     (Pcre.split ~pat:"\\s+" files));
  self#run_top data
end
else files <- files ^ data

(* Run another command until we get a prompt and then call
  self#close to close the connection. *)
method run_top data =
  self#write "top -n1 -b";
  process <- self#waitfor prompt self#close

(* Close the connection by sending an EOF. *)
method close data =
  Queue.add Telnet_eof telnet#output_queue

(* When we receive an EOF, exit the program. *)
method on_eof () =
  prerr_endline "EOF";
  exit 0

(* If a timeout event is received, exit with an error code. *)
method on_timeout () =
  prerr_endline "Timeout";
  exit 1

(* Print any thrown exceptions to standard error. *)
method on_exception exn =
  prerr_endline (Printexc.to_string exn)

(* This is the main error handler, which dispatches on
  Telnet_client events. *)
method on_input got_synch =
  while not (Queue.is_empty telnet#input_queue) do
    let tc = Queue.take telnet#input_queue in
    match tc with
    | Telnet_data data -> process data
    | Telnet_eof -> self#on_eof ()
    | Telnet_timeout -> self#on_timeout ()
    | Telnet_will _
    | Telnet_wont _
    | Telnet_do _
    | Telnet_dont _ ->
      (* The telnet_session handles these events.
        Calling this method is necessary. *)
      telnet#process_option_command tc
    | _ -> ()
  done

```

```

    (* Run the Telnet session by calling the "run" method on
       the underlying telnet_session instance. *)
    method run = telnet#run
end

(* Create an instance of our custom session class. *)
let session =
  new session
    ~host:"localhost"
    ~port:23
    ~username:"test"
    ~password:"pleac"
    ~prompt:"\\$ $"
    ~timeout:10.

(* Start the session. *)
let () = session#run ()

```

## 18.7 Pinging a Machine

```

#!/usr/bin/ocaml
(* ping - send and receive ICMP echo packets *)

(* There do not appear to be any libraries available for pingging
   servers from OCaml, ICMP or otherwise. In this recipe, we will
   make a diversion from the Perl recipe, which simply determines
   if a host is up, and instead write a lookalike for the "ping"
   shell command. We might as well, if we're going to all the
   trouble of building ICMP packets directly. *)

(* Import Unix and enable threads using findlib for convenience. *)
#use "topfind";;
#require "unix";;
#thread;;

(* The Packet module defines a data type and operations for building,
   parsing, and checking the integrity of ICMP packets. *)
module Packet = struct
  exception Invalid_length of int
  exception Invalid_checksum of int * int

  (* type' and code define the ICMP message type. An echo message
     has type'=8, code=0, and an echo reply has type'=0, code=0.
     The id is a unique identifier for the current process to help
     distinguish between replies for other processes. seq is the
     sequence number, which is usually incremented with each message.
     data is the message body whose contents depend on the type of
     message. *)
  type t = { type' : int;
             code : int;
             id : int;
             seq : int;
             data : string }

```

```

(* Define a convenience function for constructing packets. *)
let make ?(type'=8) ?(code=0) ~id ~seq data =
  {type'=type'; code=code; id=id; seq=seq; data=data}

(* Calculate a checksum for a message by adding its contents, two
   bytes at a time, folding the high order bits into the low order
   bits, and taking the logical complement. The result will be an
   int with 16-bit precision. *)
let checksum s =
  let num_bytes = String.length s in
  let num_shorts = num_bytes / 2 in
  let rec sum_shorts i sum =
    if i < num_shorts then
      let short = Int32.of_int (int_of_char s.[i * 2] lsl 8
                               + int_of_char s.[i * 2 + 1]) in
      sum_shorts (i + 1) (Int32.add sum short)
    else sum in
  let sum = sum_shorts 0 0l in
  let sum =
    if num_bytes mod 2 = 1 then
      Int32.add sum
        (Int32.of_int (int_of_char s.[num_bytes - 1] lsl 8))
    else sum in
  let sum =
    Int32.add
      (Int32.shift_right sum 16)
      (Int32.logand sum 0xffffl) in
  Int32.to_int
    (Int32.logand
      (Int32.lognot (Int32.add (Int32.shift_right sum 16) sum))
      0xffffl)

(* Convert a packet to a string that can be sent over a socket. *)
let to_string {type'=type'; code=code; id=id; seq=seq; data=data} =
  let b = Buffer.create 20 in
  Buffer.add_char b (char_of_int type');
  Buffer.add_char b (char_of_int code);
  Buffer.add_char b '\000'; (* checksum hi *)
  Buffer.add_char b '\000'; (* checksum lo *)
  Buffer.add_char b (char_of_int (id lsr 8 land 0xff));
  Buffer.add_char b (char_of_int (id land 0xff));
  Buffer.add_char b (char_of_int (seq lsr 8 land 0xff));
  Buffer.add_char b (char_of_int (seq land 0xff));
  Buffer.add_string b data;
  let packet = Buffer.contents b in
  let sum = checksum packet in
  packet.[2] <- char_of_int (sum lsr 8 land 0xff);
  packet.[3] <- char_of_int (sum land 0xff);
  packet

(* Parse a string into a packet structure. If the string is less than
   8 bytes long, an Invalid_length exception will be raised. If the

```

```

checksum does not match the contents, an Invalid_checksum
exception will be raised. *)
let of_string s =
  if String.length s < 8 then raise (Invalid_length (String.length s));
  let s' = String.copy s in
  s'.[2] <- '\000';
  s'.[3] <- '\000';
  let sum = int_of_char s.[2] lsl 8 + int_of_char s.[3] in
  let sum' = checksum s' in
  if sum <> sum' then raise (Invalid_checksum (sum, sum'));
  {type=int_of_char s.[0];
   code=int_of_char s.[1];
   id=int_of_char s.[4] lsl 8 + int_of_char s.[5];
   seq=int_of_char s.[6] lsl 8 + int_of_char s.[7];
   data=String.sub s 8 (String.length s - 8)}
end

(* Define a data structure for the message body of our echo requests. *)
type payload = { timestamp : float; data : string }

(* Send a single ICMP echo request to the given socket and address. *)
let ping socket sockaddr id seq =
  let payload =
    Marshal.to_string {timestamp=Unix.gettimeofday ();
                      data="abcdefghijklmnopqrstuvwxyz0123456"} [] in
  let message = Packet.to_string (Packet.make ~id ~seq payload) in
  ignore
    (Unix.sendto socket message 0 (String.length message) [] sockaddr)

(* Loop forever waiting for echo replies, printing them to the
   console along with their hostname, IP, and round-trip time. *)
let pong socket id =
  let buffer = String.make 256 '\000' in
  while true do
    let length, sockaddr =
      Unix.recvfrom socket buffer 0 (String.length buffer) [] in
    let response =
      Packet.of_string (String.sub buffer 20 (length - 20)) in
    match sockaddr, response with
    | Unix.ADDR_INET (addr, port),
      {Packet.type'=0; code=0; id=id'; seq=seq; data=data}
      when id = id' ->
      let host_entry = Unix.gethostbyaddr addr in
      let payload = Marshal.from_string data 0 in
      Printf.printf
        "%d bytes from %s (%s): icmp_seq=%d time=%.3f ms\n%!"
        (String.length data)
        host_entry.Unix.h_name
        (Unix.string_of_inet_addr addr)
        seq
        ((Unix.gettimeofday () -. payload.timestamp) *. 1000.)
    | _ -> ()
  done

```

```

(* Read hostname from command line. *)
let host =
  if Array.length Sys.argv <> 2
  then (Printf.eprintf "Usage: %s host\n" Sys.argv.(0); exit 1)
  else Sys.argv.(1)

(* Use DNS to find the IP address and canonical name. *)
let name, addr =
  try
    let h = Unix.gethostbyname host in
    h.Unix.h_name, h.Unix.h_addr_list.(0)
  with Not_found ->
    Printf.eprintf "%s: unknown host %s\n" Sys.argv.(0) host;
    exit 2

(* Make sure we are running as root, since this is required to
   open a socket with SOCK_RAW and send ICMP packets. *)
let () =
  if Unix.getuid () <> 0
  then (Printf.eprintf "%s: icmp ping requires root privilege\n"
        Sys.argv.(0);
        exit 3)

(* Start the ping loop. *)
let () =
  Printf.printf "PING %s (%s)\n" name (Unix.string_of_inet_addr addr);

  (* Build a socket and destination address. *)
  let proto = (Unix.getprotobyname "icmp").Unix.p_proto in
  let socket = Unix.socket Unix.PF_INET Unix.SOCK_RAW proto in
  let sockaddr = Unix.ADDR_INET (addr, 0) in

  (* Use the PID as the ID for packets, and create a counter for
     the sequence number. *)
  let id = Unix.getpid () in
  let seq = ref 0 in

  (* Start a background thread to print the echo replies. *)
  ignore (Thread.create (pong socket) id);

  (* Loop forever sending echo requests and sleeping. *)
  while true do
    incr seq;
    ping socket sockaddr id !seq;
    Unix.sleep 1
  done

```

## 18.8 Using Whois to Retrieve Information from the InterNIC

```

(* WHOIS servers depend on the TLD, and their output formats are
   informal, inconsistent, and completely different from server
   to server. This makes a general solution very large and ad-hoc.

```

The Net::Whois package, on which the original Perl recipe was based, no longer works since WHOIS servers started redirecting to other servers for most of the information.

Since no libraries are available for this task, we will do a WHOIS lookup manually using sockets. This example shows how to perform a WHOIS lookup for the "sourceforge.net" domain, and probably will not work without modification for domains under any other TLD. \*)

```
#load "unix.cma";;
#load "str.cma";;

let domain_name = "sourceforge.net"
let whois_server = "whois.internic.net"
let service = Unix.getservbyname "whois" "tcp"

let ltrim =
  let re = Str.regexp "^[\r\n\t\x00\x0B]*" in
  Str.global_replace re ""

let () =
  (* Connect to the parent server to find the redirect. *)

  let host = Unix.gethostbyname whois_server in
  let socket_in, socket_out =
    Unix.open_connection
      (Unix.ADDR_INET (host.Unix.h_addr_list.(0),
                      service.Unix.s_port)) in

  output_string socket_out domain_name;
  output_string socket_out "\n";
  flush socket_out;

  let whois_redirect_regexp = Str.regexp "Whois Server: \\.*(\\)" in
  let whois_redirect = ref "" in

  begin
    try
      while true do
        let line = ltrim (input_line socket_in) in
        if Str.string_match whois_redirect_regexp line 0
        then whois_redirect := Str.matched_group 1 line
        done
      with End_of_file ->
        Unix.shutdown_connection socket_in
    end;

    if !whois_redirect = ""
    then failwith "Couldn't find WHOIS redirect";

    (* Connect to the real server and get the WHOIS data. *)
```

```

let host = Unix.gethostbyname !whois_redirect in
let socket_in, socket_out =
  Unix.open_connection
    (Unix.ADDR_INET (host.Unix.h_addr_list.(0),
                     service.Unix.s_port)) in

output_string socket_out domain_name;
output_string socket_out "\n";
flush socket_out;

let domain_name_regexp = Str.regexp "Domain name: \\(.*\\" in
let domain_name = ref "" in

let registrant_regexp = Str.regexp "Registrant:" in
let registrant_name = ref "" in
let registrant_address = ref [] in
let registrant_country = ref "" in

let contact_regexp = Str.regexp "\\(.*\\" Contact:" in
let contacts = ref [] in

begin
  try
    while true do
      let line = ltrim (input_line socket_in) in
      if Str.string_match domain_name_regexp line 0
      then domain_name := Str.matched_group 1 line
      else if Str.string_match registrant_regexp line 0
      then
        begin
          (* Read registrant data. *)
          registrant_name := ltrim (input_line socket_in);
          let finished = ref false in
          while not !finished do
            let line = ltrim (input_line socket_in) in
            if String.length line > 2
            then registrant_address := !registrant_address @ [line]
            else if String.length line = 2
            then registrant_country := line
            else finished := true
          done
        end
      else if Str.string_match contact_regexp line 0
      then
        begin
          (* Read contact data. *)
          let contact_type = Str.matched_group 1 line in
          let contact_info = ref [] in
          for i = 1 to 6 do
            let line = ltrim (input_line socket_in) in
            contact_info := !contact_info @ [line]
          done;
          contacts := (contact_type, !contact_info) :: !contacts
        end
      end
    end
  with _ :> ()
end

```

```

        end
    done
    with End_of_file ->
        Unix.shutdown_connection socket_in
    end;

    (* Display the results. *)

    Printf.printf "The domain is called %s\n" !domain_name;
    Printf.printf "Mail for %s should be sent to:\n" !registrant_name;
    List.iter (Printf.printf "\t%s\n") !registrant_address;
    Printf.printf "\t%s\n" !registrant_country;

    if !contacts = []
    then Printf.printf "No contact information.\n"
    else
        begin
            Printf.printf "Contacts:\n";
            List.iter
                (fun (contact_type, contact_info) ->
                    Printf.printf " %s\n" contact_type;
                    List.iter (Printf.printf " %s\n") contact_info)
                !contacts
        end
    end
end

```

## 18.9 Program: expn and vrfy

```

#!/usr/bin/ocaml
(* expn -- convince smtp to divulge an alias expansion *)

#use "topfind";;                                (* Findlib *)
#require "str";;                                  (* Stdlib *)
#require "unix";;                                 (* Stdlib *)
#require "perl";;                                 (* Perl4caml *)
#require "smtp";;                                 (* Ocamlnet *)
let _ = Perl.eval "use Net::DNS"                 (* Net::DNS *)

let selfname = Unix.gethostname ()

let () =
    if Array.length Sys.argv < 2
    then (Printf.eprintf "usage: %s address@host ...\n" Sys.argv.(0);
         exit 1)

let () =
    List.iter
        (fun combo ->
            let name, host =
                match Str.bounded_split (Str.regexp "@") combo 2 with
                | [] -> "", ""
                | [name] -> name, "localhost"
                | [name; host] -> name, host
                | _ -> assert false in

```

```

let hosts =
  Perl.call_array ~fn:"mx" [Perl.sv_of_string host] in
let hosts =
  List.map (fun mx -> Perl.call_method mx "exchange" []) hosts in
let hosts =
  if hosts = [] then [Perl.sv_of_string host] else hosts in
List.iter
  (fun host ->
    let host = Perl.string_of_sv host in
    Printf.printf "Expanding %s at %s (%s): %!"
      name host combo;
    let inet_addr =
      (Unix.gethostbyname host).Unix.h_addr_list.(0) in
    let addr = Unix.ADDR_INET (inet_addr, Netsmtp.tcp_port) in
    try
      let ic, oc = Unix.open_connection addr in
      let smtp =
        new Netsmtp.client
          (new Netchannels.input_channel ic)
          (new Netchannels.output_channel oc) in
      ignore (smtp#helo ~host:selfname ());
      print_endline
        (match smtp#expn name with
         | None -> "None"
         | Some results -> String.concat ", " results);
      smtp#quit ();
      Unix.shutdown_connection ic;
      close_out oc
    with Unix.Unix_error (Unix.ECONNREFUSED, _, _) ->
      Printf.eprintf "cannot connect to %s\n" host)
  hosts)
(List.tl (Array.to_list Sys.argv))

```

## 19 CGI Programming

(\* If you've never seen a URL before, here are a few examples. \*)

```
http://caml.inria.fr/  
http://www.ocaml-tutorial.org/  
http://en.wikipedia.org/wiki/Ocaml  
http://pleac.sourceforge.net/pleac_ocaml/index.html
```

(\* The URL for a form submission using the GET method will contain a query string (the sequence of characters after the '?') with named parameters of the form: key1=value1&key2=value2&... \*)

```
http://caml.inria.fr/cgi-bin/search.en.cgi?corpus=hump&words=cgi
```

(\* The URL for a form submission using POST will not usually contain a query string, so it will appear cleaner. \*)

```
http://caml.inria.fr/cgi-bin/hump.cgi
```

(\* GET requests are assumed to be "idempotent", meaning they can be requested many times without any different effect than if they were only requested once. This has the practical difference of making GET requests easy to cache, and POST requests nearly impossible (since there is no guarantee that a POST is non-destructive). It is considered best practice to use POST, not GET, for side-effecting operations such as deleting or modifying a record. \*)

### 19.1 Writing a CGI Script

```
#!/usr/bin/env ocaml  
(* hiweb - load CGI module to decode information given by web server *)  
  
#use "topfind";; (* Findlib *)  
#require "netcgi2";; (* Ocamlnet *)  
  
(* Create an HTML escaping function for the UTF-8 encoding. *)  
let escape_html = Netencoding.Html.encode ~in_enc:'Enc_utf8 ()  
  
(* Construct the beginning of an (X)HTML document. *)  
let start_html title =  
  Printf.sprintf "\n  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transi-  
tional//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">  
<html xmlns=\"http://www.w3.org/1999/xhtml\">  
  <head>  
    <title>%s</title>  
    <meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-  
8\" />  
  </head>  
  <body>  
  
  \" (escape_html title)  
  
(* Construct the end of an (X)HTML document. *)  
let end_html = "
```

```

    </body>
</html>
"

(* Construct a few common elements. *)
let p contents =
  Printf.sprintf "<p>%s</p>" (String.concat "" contents)
let tt contents =
  Printf.sprintf "<tt>%s</tt>" (String.concat "" contents)

(* Process a page request. *)
let process (cgi : Netcgi.cgi) =
  (* Get a parameter from a form. *)
  let value = cgi#argument_value "PARAM_NAME" in

  (* Output a document. *)
  let out = cgi#out_channel#output_string in
  out (start_html "Howdy there!");
  out (p ["You typed: "; tt [escape_html value]]);
  out end_html;

  (* Flush the output buffer. *)
  cgi#out_channel#commit_work ()

(* Initialize and run the Netcgi process. *)
let () =
  let config = Netcgi.default_config in
  let buffered _ ch = new Netchannels.buffered_trans_channel ch in
  Netcgi.cgi.run ~config ~output_type:(`Transactional buffered) process

(*-----*)

(* Set the output mime-type and expiration time. *)
cgi#set_header ~content_type:"text/html" ~cache:(`Max_age 3600) ()

(*-----*)

(* Read multiple form fields, one containing multiple values. *)
let who = cgi#argument_value "Name" in
let phone = cgi#argument_value "Number" in
let picks =
  List.map
    (fun arg -> arg#value)
    (cgi#multiple_argument "Choices") in
(* ... *)

```

## 19.2 Redirecting Error Messages

```

(* The default Netcgi configuration sends all exceptions to the browser
   in nicely formatted error pages. This is helpful during development
   but may be inappropriate for production. The exception pages can be
   disabled by setting the "default_exn_handler" configuration field: *)

```

```

let config = {Netcgi.default_config with
              Netcgi.default_exn_handler=false}

(* Most web servers send standard error to the error log, which is
   typically /var/log/apache2/error.log for a default Apache 2
   configuration. You can define a "warn" function to include the
   script name in warning messages: *)
let warn = Printf.eprintf "%s: %s\n" (Filename.basename Sys.argv.(0))
let () =
  warn "This goes to the error log."

(* You can also use Printf.kprintf to define a fancier warning function
   that supports Printf formatting. *)
let warn =
  Printf.kprintf
    (Printf.eprintf "%s: %s\n" (Filename.basename Sys.argv.(0)))
let () =
  warn "So does %s." "this"

```

### 19.3 Fixing a 500 Server Error

```

#!/usr/bin/env ocaml
(* webwhoami - show web users id *)

#use "topfind";;
#require "netcgi2";;
#require "unix";;

let process (cgi : Netcgi.cgi) =
  cgi#set_header ~content_type:"text/plain" ();
  cgi#out_channel#output_string
    (Printf.sprintf "Running as %s\n"
      (Unix.getpwuid (Unix.geteuid ())).Unix.pw_name);
  cgi#out_channel#commit_work ()

let () =
  let config = Netcgi.default_config in
  let buffered _ ch = new Netchannels.buffered_trans_channel ch in
  Netcgi.cgi.run ~config ~output_type:(`Transactional buffered) process

(*-----*)

(* By using Netcgi_test.run instead of Netcgi_run, you can enable a
   command-line testing mechanism. *)

let () =
  let config = Netcgi.default_config in
  let buffered _ ch = new Netchannels.buffered_trans_channel ch in
  let output_type = `Transactional buffered in
  if Unix.isatty Unix.stdin
  then Netcgi_test.run ~config ~output_type process
  else Netcgi.cgi.run ~config ~output_type process

```

```
(* Now, you can run the CGI script from the command line to test for
  compilation and runtime errors. *)
$ ./webwhoami -help
ocaml [options] name1=value1 ... nameN=valueN
  -get           Set the method to GET (the default)
  -head         Set the method to HEAD
  -post        Set the method to POST
  -put file     Set the method to PUT with the file as argument
  -delete      Set the method to DELETE
  -mimetype type Set the MIME type for the next file argument(s) (de-
  fault: text/plain)
  -
filename path  Set the filename property for the next file argument(s)
  -filearg name=file Specify a file argument whose contents are in the file
  -user name    Set REMOTE_USER to this name
  -prop name=value Set the environment property
  -header name=value Set the request header field
  -o file       Set the output file (default: stdout)
  -help        Display this list of options
  --help       Display this list of options
```

## 19.4 Writing a Safe CGI Program

```
(* There is no feature in OCaml resembling Perl's "taint mode". *)
```

## 19.5 Making CGI Scripts Efficient

```
(* Ocamlnet provides an Apache 2 module called netcgi_apache that allows
  Netcgi scripts to run inside the Apache process. To load the module,
  put something like the following in your Apache configuration file: *)
```

```
LoadModule netcgi_module /usr/lib/apache2/modules/mod_netcgi_apache.so
NetcgiLoad pcre/pcre.cma
NetcgiLoad netsys/netsys.cma
NetcgiLoad netstring/netstring.cma
NetcgiLoad str.cma
NetcgiLoad netcgi2/netcgi.cma
NetcgiLoad netcgi_apache/netcgi_apache.cma
```

```
(* Extra libraries can be added with additional "NetcgiLoad" directives.
  The following will enable netcgi_apache for *.cma files: *)
```

```
NetcgiHandler Netcgi_apache.bytecode
AddHandler ocaml-bytecode .cma
```

```
(* Or, if you prefer, you can enable netcgi_apache for a directory: *)
```

```
<Location /caml-bin>
  SetHandler ocaml-bytecode
  NetcgiHandler Netcgi_apache.bytecode
  Options ExecCGI
  Allow from all
</Location>
```

```

(* Each script contains code similar to other Netcgi examples but uses
   Netcgi_apache.run to run the process. *)

let process (cgi : Netcgi_apache.cgi) =
  cgi#set_header ~content_type:"text/html" ();
  (* ... *)
  cgi#out_channel#commit_work ()

let () =
  let config = Netcgi.default_config in
  let buffered _ ch = new Netchannels.buffered_trans_channel ch in
  let output_type = 'Transactional buffered in
  Netcgi_apache.run ~config ~output_type process

(* Scripts need to be compiled into bytecode libraries before Apache can
   execute them. If you have findlib installed, you can compile them as
   follows: *)

ocamlfind ocamlc -package netcgi_apache -c myscript.ml
ocamlfind ocamlc -a -o myscript.cma myscript.cmo

(* Here is a Makefile to automate the build process. *)

RESULTS = myscript.cma another.cma
PACKS = netcgi_apache,anotherlib

%.cma : %.ml
  ocamlfind ocamlc -package $(PACKS) -c $<

%.cma : %.cmo
  ocamlfind ocamlc -a -o $@ $<

all: $(RESULTS)

clean:
  rm -f *.cma *.cmi *.cmo $(RESULTS)

```

## 19.6 Executing Commands Without Shell Escapes

```

(* UNSAFE *)
let status =
  Unix.system
    (command ^ " " ^ input ^ " " ^ String.concat " " files)

(* safer *)
let pid =
  Unix.create_process command (Array.of_list ([command; input] @ files))
    Unix.stdin Unix.stdout Unix.stderr
let _, status = Unix.waitpid [] pid

```

## 19.7 Formatting Lists and Tables with HTML Shortcuts

```
open Printf

(* Define some HTML helper functions. *)
let ol contents = sprintf "<ol>%s</ol>" (String.concat "" contents)
let ul contents = sprintf "<ul>%s</ul>" (String.concat "" contents)
let li ?(typ="") content =
  if typ = ""
  then sprintf "<li>%s</li>" content
  else sprintf "<li type=\"%s\">%s</li>" typ content
let tr contents = sprintf "<tr>%s</tr>" (String.concat "" contents)
let th content = sprintf "<th>%s</th>" content
let td content = sprintf "<td>%s</td>" content

(* Main CGI process. *)
let process (cgi : Netcgi.cgi) =

  (* Define a print function for convenience. *)
  let print s =
    cgi#out_channel#output_string s;
    cgi#out_channel#output_string "\n" in

  (* Print a numbered list. *)
  print (ol (List.map li ["red"; "blue"; "green"]));

  (* Print a bulleted list. *)
  let names = ["Larry"; "Moe"; "Curly"] in
  print (ul (List.map (li ~typ:"disc") names));

  (* The "li" function gets applied to a single item. *)
  print (li "alpha");

  (* If there are multiple items, use List.map. *)
  print (String.concat " " (List.map li ["alpha"; "omega"]));

  (* Build a table of states and their cities. *)
  let ( => ) k v = (k, v) in
  let state_cities =
    [
      "Wisconsin" => [ "Superior"; "Lake Geneva"; "Madison" ];
      "Colorado"  => [ "Denver"; "Fort Collins"; "Boulder" ];
      "Texas"     => [ "Plano"; "Austin"; "Fort Stockton" ];
      "California" => [ "Sebastopol"; "Santa Rosa"; "Berkeley" ];
    ] in

  (* Print the table in sorted order. *)
  print "<TABLE> <CAPTION>Cities I Have Known</CAPTION>";
  print (tr (List.map th ["State"; "Cities"]));
  List.iter
    (fun (state, cities) ->
      print (tr (th state :: List.map td (List.sort compare cities))))
    (List.sort compare state_cities);
```

```

print "</TABLE>";

(* Flush the output buffer. *)
cgi#out_channel#commit_work ()

(*-----*)

(* salcheck - check for salaries *)

(* Requires ocaml-mysql, available here:
   http://raevnos.pennmush.org/code/ocaml-mysql/

   For netcgi_apache, the following configuration directive is needed:
   NetcgiLoad mysql/mysql.cma *)

open Printf

let escape_html = Netencoding.Html.encode ~in_enc:'Enc_utf8 ()

let start_html title =
  sprintf "\
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transi-
tional//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">
<html xmlns=\"http://www.w3.org/1999/xhtml\">
  <head>
    <title>%s</title>
    <meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-
8\" />
  </head>
  <body>

" (escape_html title)

let end_html = "

  </body>
</html>
"

let start_form ?(action="") ?(method='get') () =
  sprintf "<form action=\"%s\" method=\"%s\">"
    (escape_html action) (escape_html method')
let end_form = "</form>"

let p contents = sprintf "<p>%s</p>" (String.concat "" contents)
let h1 contents = sprintf "<h1>%s</h1>" (String.concat "" contents)

let textfield ?(name="") ?(value="") () =
  sprintf "<input type=\"text\" name=\"%s\" value=\"%s\" />"
    (escape_html name) (escape_html value)

let submit ?(name="") ?(value="") () =
  sprintf "<input type=\"submit\" name=\"%s\" value=\"%s\" />"

```

```

    (escape_html name) (escape_html value)

let tr contents = sprintf "<tr>%s</tr>" (String.concat "" contents)
let td content = sprintf "<td>%s</td>" content

let process (cgi : Netcgi.cgi) =
  let limit = cgi#argument_value "LIMIT" in

  cgi#set_header ~content_type:"text/html" ();

  let print s =
    cgi#out_channel#output_string s;
    cgi#out_channel#output_string "\n" in

  print (start_html "Salary Query");
  print (h1 ["Search"]);
  print (start_form ());
  print (p ["Enter minimum salary ";
            textfield ~name:"LIMIT" ~value:limit ()]);
  print (submit ~value:"Submit" ());
  print end_form;

  if limit <> "" then
    begin
      let db =
        Mysql.quick_connect
          ~user:"username"
          ~password:"password"
          ~database:"somedb"
          ~host:"localhost"
          ~port:3306 () in
      let sql =
        sprintf "
          SELECT name, salary
          FROM   employees
          WHERE  salary > %s
        " (Mysql.ml2float (float_of_string limit)) in
      let result = Mysql.exec db sql in
      print (h1 ["Results"]);
      print "<table border=\\\"1\\\">";
      print (String.concat "\\n"
             (Mysql.map result
              (fun values ->
                tr [td (escape_html
                      (Mysql.not_null
                       Mysql.str2ml values.(0))));
                  td (sprintf "%.2f"
                          (Mysql.not_null
                           Mysql.float2ml values.(1)))))]));
      print "</table>";
      Mysql.disconnect db;
    end;
end;

```

```

print end_html;
cgi#out_channel#commit_work ()

let () =
  let buffered _ ch = new Netchannels.buffered_trans_channel ch in
  Netcgi_apache.run
    ~output_type:(`Transactional buffered)
    (fun cgi -> process (cgi :> Netcgi.cgi))

```

## 19.8 Redirecting to a Different Location

```

let process (cgi : Netcgi.cgi) =
  let url = "http://caml.inria.fr/cgi-bin/hump.cgi" in
  cgi#set_redirection_header url;
  (* By default, the above will send a 302 Found. To instead send
     a 301 Moved Permanently, use the following command. *)
  cgi#set_header ~status:`Moved_permanently ()

  (*-----*)

  (* oreobounce - set a cookie and redirect the browser *)

  let process (cgi : Netcgi.cgi) =
    let oreo =
      Netcgi_common.Cookie.make
        ~max_age:(60 * 60 * 24 * 30 * 3) (* 3 months *)
        ~domain:".sourceforge.nett"
        "filling" "vanilla crême" in
    let whither = "http://somewhere.sourceforge.net/nonesuch.html" in
    cgi#set_redirection_header ~set_cookies:[oreo] whither

  let () =
    let buffered _ ch = new Netchannels.buffered_trans_channel ch in
    Netcgi_apache.run
      ~output_type:(`Transactional buffered)
      (fun cgi -> process (cgi :> Netcgi.cgi))

  (*
  HTTP/1.1 302 Found
  Date: Thu, 06 Nov 2008 04:39:53 GMT
  Server: Apache/2.2.9 (Debian) Netcgi_apache/2.2.9 PHP/5.2.6-5 with Suhosin-
  Patch
  Set-Cookie: filling=vanilla%20cr%E8me;Version=1;Domain=.sourceforge.nt;Max-
  Age=7776000;Expires=Wed, 04 Feb 2009 04:39:55 +0000
  Location: http://somewhere.sourceforge.net/nonesuch.html
  Status: 302
  Transfer-Encoding: chunked
  Content-Type: text/html
  *)

  (*-----*)

  (* os_snipe - redirect to a Jargon File entry about current OS *)

```

```

let process (cgi : Netcgi.cgi) =
  let dir = "http://www.wins.uva.nl/%7Emes/jargon" in
  let page =
    match cgi#environment#user_agent with
    | s when Str.string_match
      (Str.regexp ".*Mac") s 0 ->
      "m/Macintrash.html"
    | s when Str.string_match
      (Str.regexp ".*Win\\(dows \\)?NT") s 0 ->
      "e/evilandrude.html"
    | s when Str.string_match
      (Str.regexp ".*\\(Win\\|MSIE\\|WebTV\\)") s 0 ->
      "m/MicroslothWindows.html"
    | s when Str.string_match
      (Str.regexp ".*Linux") s 0 ->
      "l/Linux.html"
    | s when Str.string_match
      (Str.regexp ".*HP-UX") s 0 ->
      "h/HP-SUX.html"
    | s when Str.string_match
      (Str.regexp ".*SunOS") s 0 ->
      "s/ScumOS.html"
    | _ ->
      "a/AppendixB.html" in
  cgi#set_redirection_header (dir ^ "/" ^ page)

let () =
  let buffered _ ch = new Netchannels.buffered_trans_channel ch in
  Netcgi_apache.run
  ~output_type:(`Transactional buffered)
  (fun cgi -> process (cgi :> Netcgi.cgi))

(*-----*)

let process (cgi : Netcgi.cgi) =
  cgi#environment#set_status `No_content

(*
HTTP/1.1 204 No Content
Date: Thu, 06 Nov 2008 05:25:46 GMT
Server: Apache/2.2.9 (Debian) Netcgi_apache/2.2.9 PHP/5.2.6-5 with Suhosin-
Patch
Status: 204
Content-Type: text/html
*)

```

## 19.9 Debugging the Raw HTTP Exchange

```

#!/usr/bin/ocaml
(* dummyhttpd - start an HTTP daemon and print what the client sends *)

#load "unix.cma";;

```

```

let host = "localhost"
let port = 8989

let () =
  Printf.printf "Please contact me at: http://%s:%d/\n%!" host port;
  let addr = (Unix.gethostbyname host).Unix.h_addr_list.(0) in
  let server = Unix.socket Unix.PF_INET Unix.SOCK_STREAM 0 in
  Unix.setsockopt server Unix.SO_REUSEADDR true;
  Unix.bind server (Unix.ADDR_INET (addr, port));
  Unix.listen server 10;
  while true do
    begin
      let client, sockaddr = Unix.accept server in
      let in_channel = Unix.in_channel_of_descr client in
      try
        while true do
          let line = input_line in_channel in
          print_endline line
        done
        with End_of_file ->
          print_endline "EOF";
          close_in in_channel
      end
    end
  done

```

## 19.10 Managing Cookies

```

(* Read a cookie: *)
Netcgi_common.Cookie.value (cgi#environment#cookie "preference name")

(* Make a cookie: *)
let cookie =
  Netcgi_common.Cookie.make
    ~max_age:(60 * 60 * 24 * 365 * 2) (* 2 years *)
    "preference name" (* name *)
    "whatever you'd like" (* value*)

(* Write a cookie: *)
cgi#set_header ~set_cookies:[cookie] ()

(*-----*)

#!/usr/bin/env ocaml
(* ic_cookies - sample CGI script that uses a cookie *)

#use "topfind";;
#require "netcgi2";;

open Printf

let escape_html = Netencoding.Html.encode ~in_enc:'Enc_utf8 ()

```

```

let start_html title =
  sprintf "\
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transi-
tional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>%s</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8" />
  </head>
  <body>
" (escape_html title)

let end_html = "
  </body>
</html>
"

let h1 contents = sprintf "<h1>%s</h1>" (String.concat "" contents)
let hr = "<hr />"
let p contents = sprintf "<p>%s</p>" (String.concat "" contents)

let start_form ?(action="") ?(method="get") () =
  sprintf "<form action=\"%s\" method=\"%s\">"
    (escape_html action) (escape_html method')
let end_form = "</form>"

let textfield ?(name="") ?(value="") () =
  sprintf "<input type=\"text\" name=\"%s\" value=\"%s\" />"
    (escape_html name) (escape_html value)

let process (cgi : Netcgi.cgi) =
  let cookname = "favorite ice cream" in
  let favorite = cgi#argument_value "flavor" in
  let tasty =
    try Netcgi_common.Cookie.value (cgi#environment#cookie cookname)
    with Not_found -> "mint" in
  let print s =
    cgi#out_channel#output_string s;
    cgi#out_channel#output_string "\n" in

  cgi#set_header ~content_type:"text/html" ();
  if favorite = ""
  then
  begin
    print (start_html "Ice Cookies");
    print (h1 ["Hello Ice Cream"]);
    print hr;
    print (start_form ~method':"post" ());
    print (p ["Please select a flavor: ";
              textfield ~name:"flavor" ~value:tasty ()]);
    print end_form;
    print hr;

```

```

    print end_html;
  end
else
  begin
    let cookie =
      Netcgi_common.Cookie.make
        ~max_age:(60 * 60 * 24 * 365 * 2) (* 2 years *)
        cookname favorite in
      cgi#set_header ~set_cookies:[cookie] ();
      print (start_html "Ice Cookies, #2");
      print (h1 ["Hello Ice Cream"]);
      print (p ["You chose as your favorite flavor '";
        escape_html favorite; "'."]);
      print end_html;
    end;
    cgi#out_channel#commit_work ()

let () =
  let config = Netcgi.default_config in
  let buffered _ ch = new Netchannels.buffered_trans_channel ch in
  let output_type = 'Transactional buffered in
  if Unix.isatty Unix.stdin
  then Netcgi_test.run ~config ~output_type process
  else Netcgi_cgi.run ~config ~output_type process

```

## 19.11 Creating Sticky Widgets

```

#!/usr/bin/env ocaml
(* who.cgi - run who(1) on a user and format the results nicely *)

#use "topfind";;
#require "netcgi2";;
#require "str";;

open Printf

let escape_html = Netencoding.Html.encode ~in_enc:'Enc_utf8 ()

let start_html title =
  sprintf "\
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transi-
tional//EN" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">
<html xmlns=\"http://www.w3.org/1999/xhtml\">
  <head>
    <title>%s</title>
    <meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-
8\" />
  </head>
  <body>
" (escape_html title)

let end_html = "
  </body>

```

```

</html>
"

let h1 contents = sprintf "<h1>%s</h1>" (String.concat "" contents)
let p contents = sprintf "<p>%s</p>" (String.concat "" contents)
let pre contents = sprintf "<pre>%s</pre>" (String.concat "" contents)

let start_form ?(action="") ?(method='get') () =
  sprintf "<form action=\"%s\" method=\"%s\">"
    (escape_html action) (escape_html method')
let end_form = "</form>"

let textfield ?(name="") ?(value="") () =
  sprintf "<input type=\"text\" name=\"%s\" value=\"%s\" />"
    (escape_html name) (escape_html value)

let submit ?(name="") ?(value="") () =
  sprintf "<input type=\"submit\" name=\"%s\" value=\"%s\" />"
    (escape_html name) (escape_html value)

let process (cgi : Netcgi.cgi) =
  let print s =
    cgi#out_channel#output_string s;
    cgi#out_channel#output_string "\n" in

  let name = cgi#argument_value "WHO" in

  (* print search form *)
  cgi#set_header ~content_type:"text/html" ();
  print (start_html "Query Users");
  print (h1 ["Search"]);
  print (start_form ~method':"post" ());
  print (p ["Which user? ";
    textfield ~name:"WHO" ~value:name ()]);
  print (submit ~value:"Query" ());
  print end_form;

  (* print results of the query if we have someone to look for *)
  if name <> "" then
    begin
      print (h1 ["Results"]);
      let regexp = Str.regexp name in
      let proc = Unix.open_process_in "who" in
      let found = ref false in
      let html = Buffer.create 0 in
      begin
        (* call who and build up text of response *)
        try
          while true do
            let line = input_line proc in
            (* only lines matching [name] *)
            if Str.string_match regexp line 0 then
              begin

```

```

        Buffer.add_string html (escape_html line ^ "\n");
        found := true;
    end
done
with End_of_file ->
    close_in proc;
    (* nice message if we didn't find anyone by that name *)
    if not !found
    then Buffer.add_string html
        (escape_html name ^ " is not logged in");
    end;
    print (pre [Buffer.contents html]);
end;

print end_html

let () =
    let config = Netcgi.default_config in
    let buffered _ ch = new Netchannels.buffered_trans_channel ch in
    let output_type = 'Transactional buffered in
    if Unix.isatty Unix.stdin
    then Netcgi_test.run ~config ~output_type process
    else Netcgi_cgi.run ~config ~output_type process

```

## 19.12 Writing a Multiscreen CGI Script

```

#!/usr/bin/env ocaml

#use "topfind";;
#require "netcgi2";;

open Printf

let ( => ) k v = (k, v)

let escape_html = Netencoding.Html.encode ~in_enc:'Enc_utf8 ()

let start_html title =
    sprintf "\
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transi-
tional//EN" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">
<html xmlns=\"http://www.w3.org/1999/xhtml\">
  <head>
    <title>%s</title>
    <meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-
8\" />
  </head>
  <body>
" (escape_html title)

let end_html = "
  </body>
</html>

```

```

"

let h1 contents = sprintf "<h1>%s</h1>" (String.concat "" contents)
let p contents = sprintf "<p>%s</p>" (String.concat "" contents)
let pre contents = sprintf "<pre>%s</pre>" (String.concat "" contents)

let start_form ?(action="") ?(method="get") () =
  sprintf "<form action=\"%s\" method=\"%s\">"
    (escape_html action) (escape_html method')
let end_form = "</form>"

let hidden ?(name="") ?(value="") () =
  sprintf "<input type=\"hidden\" name=\"%s\" value=\"%s\" />"
    (escape_html name) (escape_html value)

let submit ?(name="") ?(value="") () =
  sprintf "<input type=\"submit\" name=\"%s\" value=\"%s\" />"
    (escape_html name) (escape_html value)

let popup_menu ?(name="") ?(value="") values =
  let options =
    List.map
      (fun (value', label) ->
        sprintf "<option %s value=\"%s\">%s</option>"
          (if value = value' then "selected=\"selected\"" else "")
          (escape_html value')
          (escape_html label))
    values in
  sprintf "<select name=\"%s\">\n%s\n</select>"
    (escape_html name) (String.concat "\n" options)

let standard_header () = h1 ["Program Title"]
let standard_footer () = "<hr />"

let to_page value = submit ~name:".State" ~value ()

(* when we get a .State that doesn't exist *)
let no_such_page (cgi : Netcgi.cgi) print = ()

let front_page (cgi : Netcgi.cgi) print active = ()
let sweater (cgi : Netcgi.cgi) print active = ()
let checkout (cgi : Netcgi.cgi) print active = ()
let credit_card (cgi : Netcgi.cgi) print active = ()
let order (cgi : Netcgi.cgi) print active = ()

let t_shirt (cgi : Netcgi.cgi) print active =
  let size = cgi#argument_value "size" in
  let color = cgi#argument_value "color" in
  if active then
    begin
      print (p ["You want to buy a t-shirt?"]);
      print (p ["Size: ";
        popup_menu ~name:"size" ~value:size

```

```

        ["XL" => "X-Large";
         "L"  => "Large";
         "M"  => "Medium";
         "S"  => "Small";
         "XS" => "X-Small"]]);
    print (p ["Color: ";
             popup_menu ~name:"color" ~value:color
             ["Black" => "Black"; "White" => "White"]]);
    print (p [to_page "Shoes"; to_page "Checkout"]);
end
else
begin
    print (hidden ~name:"size" ~value:size ());
    print (hidden ~name:"color" ~value:color ());
end

let states =
[
    "Default" => front_page;
    "Shirt"   => t_shirt;
    "Sweater" => sweater;
    "Checkout" => checkout;
    "Card"     => credit_card;
    "Order"    => order;
    "Cancel"   => front_page;
]

let process (cgi : Netcgi.cgi) =
    let page = cgi#argument_value ~default:"Default" ".State" in
    cgi#set_header ~content_type:"text/html" ();
    let print s =
        cgi#out_channel#output_string s;
        cgi#out_channel#output_string "\n" in
    print (start_html "Program Title");
    print (standard_header ());
    print (start_form ());
    if List.mem_assoc page states
    then List.iter (fun (state, sub) ->
                    sub cgi print (page = state)) states
    else no_such_page cgi print;
    print (standard_footer ());
    print end_form;
    print end_html;
    cgi#out_channel#commit_work ()

let () =
    let config = Netcgi.default_config in
    let buffered _ ch = new Netchannels.buffered_trans_channel ch in
    let output_type = 'Transactional buffered in
    if Unix.isatty Unix.stdin
    then Netcgi_test.run ~config ~output_type process
    else Netcgi.cgi.run ~config ~output_type process

```

## 19.13 Saving a Form to a File or Mail Pipe

```
#!/usr/bin/env ocaml

#use "topfind";;
#require "netcgi2";;

let escape  = Netencoding.Url.encode ~plus:false
let unescape = Netencoding.Url.decode ~plus:false

let save_arguments (ch : Netchannels.out_obj_channel) args =
  List.iter
    (fun arg ->
      ch#output_string (escape arg#name);
      ch#output_char '=';
      ch#output_string (escape arg#value);
      ch#output_char '\n')
    args;
  ch#output_string "=\n"

let process (cgi : Netcgi.cgi) =
  (* first open and exclusively lock the file *)
  let ch = open_out_gen [Open_append; Open_creat] 0o666 "/tmp/formlog" in
  Unix.lockf (Unix.descr_of_out_channel ch) Unix.F_LOCK 0;

  (* locally set some additional arguments *)
  let arguments =
    Netcgi.Argument.set
      [
        Netcgi.Argument.simple "_timestamp"
          (string_of_float (Unix.time ()));
        Netcgi.Argument.simple "_environs"
          (String.concat "\n" (Array.to_list (Unix.environment ()))));
      ]
    cgi#arguments in

  (* wrap output in a Netchannel and save *)
  let ch = new Netchannels.output_channel ch in
  save_arguments ch arguments;
  ch#close_out ();

  (* send in an email *)
  let body = Buffer.create 256 in
  let ch = new Netchannels.output_buffer body in
  save_arguments ch arguments;
  Netsendmail.sendmail
    (Netsendmail.compose
     ~from_addr:("your cgi script", Sys.argv.(0))
     ~to_addrs:[("hisname", "hisname@hishost.com")]
     ~subject:"mailed form submission"
     (Buffer.contents body))

let () =
```

```

let config = Netcgi.default_config in
let buffered _ ch = new Netchannels.buffered_trans_channel ch in
let output_type = 'Transactional buffered in
if Unix.isatty Unix.stdin
then Netcgi_test.run ~config ~output_type process
else Netcgi_cgi.run ~config ~output_type process

(*-----*)

#!/usr/bin/ocaml

#use "topfind";;
#require "str";;
#require "unix";;
#require "netstring";;

let escape = Netencoding.Url.encode ~plus:false
let unescape = Netencoding.Url.decode ~plus:false

let parse_env data =
  let result = Hashtbl.create 16 in
  List.iter
    (fun line ->
      try
        let index = String.index line '=' in
        Hashtbl.add result
          (String.sub line 0 index)
          (String.sub line (index + 1) (String.length line - index - 1))
        with Not_found -> ())
    (Str.split (Str.regexp "\n") data);
  result

let ends_with suffix s =
  try Str.last_chars s (String.length suffix) = suffix
  with Invalid_argument _ -> false

let () =
  let forms = open_in "/tmp/formlog" in
  let args = Hashtbl.create 8 in
  let count = ref 0 in
  Unix.lockf (Unix.descr_of_in_channel forms) Unix.F_RLOCK 0;
  try
    while true do
      let line = input_line forms in
      if line = "=" then
        begin
          let his_env = parse_env (Hashtbl.find args "_environs") in
          let host =
            try Hashtbl.find his_env "REMOTE_HOST"
            with Not_found -> "" in
          if host <> "perl.com" && not (ends_with ".perl.com" host)
          then (count :=
              (!count +

```

```

                int_of_string
                (try Hashtbl.find args "items requested"
                 with Not_found -> "0"));
        Hashtbl.clear args
    end
else
    begin
        let index = String.index line '=' in
            Hashtbl.add args
                (unescape (String.sub line 0 index))
                (unescape
                 (String.sub
                  line
                  (index + 1)
                  (String.length line - index - 1)))
        end
    done
with End_of_file ->
    close_in forms;
    Printf.printf "Total orders: %d\n" !count

```

## 19.14 Program: chemiserie

```

#!/usr/bin/env ocaml
(* chemiserie - simple CGI shopping for shirts and sweaters *)

#use "topfind";;
#require "netcgi2";;

open Printf

let ( => ) k v = (k, v)

let escape_html = Netencoding.Html.encode ~in_enc:'Enc_utf8 ()

let start_html title =
    sprintf "\
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transi-
tional//EN\" \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">
<html xmlns=\"http://www.w3.org/1999/xhtml\">
  <head>
    <title>%s</title>
    <meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-
8\" />
  </head>
  <body>
" (escape_html title)

let end_html = "
  </body>
</html>
"

```

```

let h1 contents = sprintf "<h1>%s</h1>" (String.concat "" contents)
let h2 contents = sprintf "<h2>%s</h2>" (String.concat "" contents)
let p contents = sprintf "<p>%s</p>" (String.concat "" contents)
let pre contents = sprintf "<pre>%s</pre>" (String.concat "" contents)

let start_form ?(action="") ?(method="get") () =
  sprintf "<form action=\"%s\" method=\"%s\">"
    (escape_html action) (escape_html method')
let end_form = "</form>"

let hidden ?(name="") ?(value="") () =
  sprintf "<input type=\"hidden\" name=\"%s\" value=\"%s\" />"
    (escape_html name) (escape_html value)

let submit ?(name="") ?(value="") () =
  sprintf "<input type=\"submit\" name=\"%s\" value=\"%s\" />"
    (escape_html name) (escape_html value)

let textfield ?(name="") ?(value="") () =
  sprintf "<input type=\"text\" name=\"%s\" value=\"%s\" />"
    (escape_html name) (escape_html value)

let popup_menu ?(name="") ?(value="") values =
  let options =
    List.map
      (fun (value', label) ->
        sprintf "<option %s value=\"%s\">%s</option>"
          (if value = value' then "selected=\"selected\"" else "")
          (escape_html value')
          (escape_html label))
      values in
  sprintf "<select name=\"%s\">\n%s\n</select>"
    (escape_html name) (String.concat "\n" options)

let defaults label =
  sprintf "<input type=\"button\" value=\"%s\" onclick=\"%s\" />"
    (escape_html label) "javascript:location.href='?'"

let to_page value = submit ~name:".State" ~value ()

(*****
 * header, footer, menu functions
 *****)

let standard_header print =
  print (start_html "Shirts");
  print (start_form ())

let standard_footer print =
  print end_form;
  print end_html

let shop_menu print =

```

```

print (p [defaults "Empty My Shopping Cart";
         to_page "Shirt";
         to_page "Sweater";
         to_page "Checkout"])

(*****
 * subroutines for each screen
 *****)

(* The default page. *)
let front_page cgi print active =
  if active then
    begin
      print (h1 ["Hi!"]);
      print "Welcome to our Shirt Shop! Please make your selection ";
      print "from the menu below.";
      shop_menu print;
    end

(* Page to order a shirt from. *)
let shirt (cgi : Netcgi.cgi) print active =
  let sizes = ["XL" => "X-Large";
              "L"  => "Large";
              "M"  => "Medium";
              "S"  => "Small";
              "XS" => "X-Small"] in
  let colors = ["Black" => "Black"; "White" => "White"] in

  let size, color, count =
    cgi#argument_value "shirt_size",
    cgi#argument_value "shirt_color",
    cgi#argument_value "shirt_count" in

  (* sanity check *)
  let size =
    if List.mem_assoc size sizes
    then size
    else fst (List.hd sizes) in
  let color =
    if List.mem_assoc color colors
    then color
    else fst (List.hd colors) in

  if active then
    begin
      print (h1 ["T-Shirt"]);
      print (p ["What a shirt! This baby is decked out with all the ";
              "options. It comes with full luxury interior, cotton ";
              "trim, and a collar to make your eyes water! ";
              "Unit price: $33.00"]);
      print (h2 ["Options"]);
      print (p ["How Many? ";
              textfield

```

```

        ~name:"shirt_count"
        ~value:count ());
    print (p ["Size? ";
        popup_menu ~name:"shirt_size" ~value:size sizes]);
    print (p ["Color? ";
        popup_menu ~name:"shirt_color" ~value:color colors]);
    shop_menu print;
end
else
begin
    if size <> ""
    then print (hidden ~name:"shirt_size" ~value:size ());
    if color <> ""
    then print (hidden ~name:"shirt_color" ~value:color ());
    if count <> ""
    then print (hidden ~name:"shirt_count" ~value:count ());
end

(* Page to order a sweater from. *)
let sweater (cgi : Netcgi.cgi) print active =
    let sizes = ["XL" => "X-Large";
        "L" => "Large";
        "M" => "Medium"] in
    let colors = ["Chartreuse" => "Chartreuse";
        "Puce" => "Puce";
        "Lavender" => "Lavender"] in

    let size, color, count =
        cgi#argument_value "sweater_size",
        cgi#argument_value "sweater_color",
        cgi#argument_value "sweater_count" in

    (* sanity check *)
    let size =
        if List.mem_assoc size sizes
        then size
        else fst (List.hd sizes) in
    let color =
        if List.mem_assoc color colors
        then color
        else fst (List.hd colors) in

    if active then
    begin
        print (h1 ["Sweater"]);
        print (p ["Nothing implies preppy elegance more than this fine ";
            "sweater. Made by peasant workers from black market ";
            "silk, it slides onto your lean form and cries out ";
            "'Take me, for I am a god!'. Unit price: $49.99."]);
        print (h2 ["Options"]);
        print (p ["How Many? ";
            textfield
            ~name:"sweater_count"

```

```

        ~value:count ());
    print (p ["Size? ";
        popup_menu ~name:"sweater_size" ~value:size sizes]);
    print (p ["Color? ";
        popup_menu ~name:"sweater_color" ~value:color colors]);
    shop_menu print;
end
else
begin
    if size <> ""
    then print (hidden ~name:"sweater_size" ~value:size ());
    if color <> ""
    then print (hidden ~name:"sweater_color" ~value:color ());
    if count <> ""
    then print (hidden ~name:"sweater_count" ~value:count ());
end

let calculate_price (cgi : Netcgi.cgi) =
    let shirts =
        try int_of_string (cgi#argument_value "shirt_count")
        with Failure _ -> 0 in
    let sweaters =
        try int_of_string (cgi#argument_value "shirt_count")
        with Failure _ -> 0 in
    sprintf "$%.2f" (float shirts *. 33.0 +. float sweaters *. 49.99)

(* Returns HTML for the current order ("You have ordered ...") *)
let order_text (cgi : Netcgi.cgi) =
    let shirt_count = cgi#argument_value "shirt_count" in
    let shirt_size = cgi#argument_value "shirt_size" in
    let shirt_color = cgi#argument_value "shirt_color" in

    let sweater_count = cgi#argument_value "sweater_count" in
    let sweater_size = cgi#argument_value "sweater_size" in
    let sweater_color = cgi#argument_value "sweater_color" in

    let html = Buffer.create 0 in

    if not (List.mem shirt_count [""; "0"]) then
        Buffer.add_string html
            (p ["You have ordered "; escape_html shirt_count;
                " shirts of size "; escape_html shirt_size;
                " and color "; escape_html shirt_color; "."]);

    if not (List.mem sweater_count [""; "0"]) then
        Buffer.add_string html
            (p ["You have ordered "; escape_html sweater_count;
                " sweaters of size "; escape_html sweater_size;
                " and color "; escape_html sweater_color; "."]);

    let html = Buffer.contents html in
    match html with
    | "" -> p ["Nothing!"]

```

```

| html -> html ^ p ["For a total cost of "; calculate_price cgi]

(* Page to display current order for confirmation. *)
let checkout (cgi : Netcgi.cgi) print active =
  if active then
    begin
      print (h1 ["Order Confirmation"]);
      print (p ["You ordered the following:"]);
      print (order_text cgi);
      print (p ["Is this right? Select 'Card' to pay for the items ";
        "or 'Shirt' or 'Sweater' to continue shopping."]);
      print (p [to_page "Card";
        to_page "Shirt";
        to_page "Sweater"]);
    end

  end

(* Page to gather credit-card information. *)
let credit_card (cgi : Netcgi.cgi) print active =
  let widgets = ["Name"; "Address1"; "Address2"; "City"; "Zip"; "State";
    "Phone"; "Card"; "Expiry"] in
  if active then
    begin
      print (pre [p ["Name:          ";
        textfield
          ~name:"Name"
          ~value:(cgi#argument_value "Name") ()];
        p ["Address:          ";
        textfield
          ~name:"Address1"
          ~value:(cgi#argument_value "Address1") ()];
        p ["          ";
        textfield
          ~name:"Address2"
          ~value:(cgi#argument_value "Address2") ()];
        p ["City:           ";
        textfield
          ~name:"City"
          ~value:(cgi#argument_value "City") ()];
        p ["Zip:            ";
        textfield
          ~name:"Zip"
          ~value:(cgi#argument_value "Zip") ()];
        p ["State:          ";
        textfield
          ~name:"State"
          ~value:(cgi#argument_value "State") ()];
        p ["Phone:          ";
        textfield
          ~name:"Phone"
          ~value:(cgi#argument_value "Phone") ()];
        p ["Credit Card *: ";
        textfield
          ~name:"Card"

```

```

        ~value:(cgi#argument_value "Card") ());
    p ["Expiry:      ";
      textfield
      ~name:"Expiry"
      ~value:(cgi#argument_value "Expiry") (]]]);

    print (p ["Click on 'Order' to order the items. ";
             "Click on 'Cancel' to return shopping."]);

    print (p [to_page "Order"; to_page "Cancel"]);
  end
else
  begin
    List.iter
      (fun widget ->
        print (hidden
              ~name:widget
              ~value:(cgi#argument_value widget) ()))
      widgets
  end

(* Page to complete an order. *)
let order cgi print active =
  if active then
    begin
      (* you'd check credit card values here *)
      print (h1 ["Ordered!"]);
      print (p ["You have ordered the following toppings:"]);
      print (order_text cgi);

      print (p [defaults "Begin Again"]);
    end
  end

(* state table mapping pages to functions *)
type page = Netcgi.cgi -> (string -> unit) -> bool -> unit
let (states : (string * page) list) =
  [
    "Default" => front_page;
    "Shirt"   => shirt;
    "Sweater" => sweater;
    "Checkout" => checkout;
    "Card"    => credit_card;
    "Order"   => order;
    "Cancel"  => front_page;
  ]

let no_such_page (cgi : Netcgi.cgi) print current_screen =
  print ("No screen for " ^ current_screen)

let process (cgi : Netcgi.cgi) =
  let current_screen = cgi#argument_value ~default:"Default" ".State" in

  let print s =

```

```

    cgi#out_channel#output_string s;
    cgi#out_channel#output_string "\n" in

(* Generate the current page. *)
cgi#set_header ~content_type:"text/html" ();
standard_header print;
if List.mem_assoc current_screen states
then List.iter (fun (state, sub) ->
                sub cgi print (current_screen = state)) states
else no_such_page cgi print current_screen;
standard_footer print;
cgi#out_channel#commit_work ()

let () =
  let config = Netcgi.default_config in
  let buffered _ ch = new Netchannels.buffered_trans_channel ch in
  let output_type = 'Transactional buffered in
  if Unix.isatty Unix.stdin
  then Netcgi_test.run ~config ~output_type process
  else Netcgi_cgi.run ~config ~output_type process

```



```

        call#response_status_code
        call#response_status_text
    | 'Http_protocol_error e ->
        Printf.eprintf "HTTP protocol error: %s\n"
        (Printexc.to_string e)
    | 'Redirection ->
        Printf.eprintf "Redirection\n"
    | 'Server_error ->
        Printf.eprintf "Server error\n"
    | 'Unservd ->
        assert false

```

## 20.2 Automating Form Submission

```

#use "topfind";;
#require "netclient";;
open Http_client.Convenience

(* Submit a form using GET. *)
let url = "http://www.perl.com/cgi-bin/cpan_mod?module=DB_File&readme=1"
let content = http_get url

(* Submit a form using POST. Since we need to follow a redirect here,
   we can't use the "Convenience" methods. *)
let url = "http://www.perl.com/cgi-bin/cpan_mod"
let params = ["module", "DB_File"; "readme", "1"]
let () =
    let call = new Http_client.post url params in
    call#set_redirect_mode Http_client.Redirect;
    let pipeline = new Http_client.pipeline in
    pipeline#add call;
    pipeline#run ()
let content = call#response_body#value

(* GET parameters can be URL encoded with Netencoding.Url.encode. *)
let arg = "\"this isn't <EASY> & <FUN>\""
Netencoding.Url.encode arg
(* - : string = "%22this+isn%27t+%3CEASY%3E+%26+%3CFUN%3E%22" *)
Netencoding.Url.encode ~plus:false arg
(* - : string = "%22this%20isn%27t%20%3CEASY%3E%20%26%20%3CFUN%3E%22" *)

(* To use a proxy, either set the "http_proxy" environment variable and
   call "set_proxy_from_environment" on the pipeline (done automatically
   for the "Convenience" methods) or set the proxy host and port using
   the "set_proxy" method: *)
let () = pipeline#set_proxy "localhost" 3128

```

## 20.3 Extracting URLs

```

(* The Nethtml library, part of Ocamlnet, can parse arbitrary HTML from
   files and web pages. *)

#use "topfind";;

```

```

#require "netstring";;

open Nhtml

(* Define a function to walk through all the elements in a document and
   accumulate the results of a user-supplied function for each element.
   This is known as a "fold" in functional programming. *)
let rec fold_elements f accu = function
  | Element (_, _, children) as element ->
    let accu =
      List.fold_right
        (fun child accu ->
          fold_elements f accu child)
        children
        accu in
    f accu element
  | other -> accu

(* Define a type for links so we can tell anchors and images apart. *)
type link = A of string | IMG of string

(* Using fold_elements, define a function that collects the URLs from
   all the "a" and "img" tags. *)
let find_links elements =
  List.flatten
    (List.map
      (fold_elements
        (fun accu element ->
          match element with
          | Element ("a", attribs, _) ->
            (try A (List.assoc "href" attribs) :: accu
             with Not_found -> accu)
          | Element ("img", attribs, _) ->
            (try IMG (List.assoc "src" attribs) :: accu
             with Not_found -> accu)
          | _ -> accu)
        []))
    elements)

(* Parse an HTML file. *)
let elements = parse (new Nchannels.input_channel (open_in filename))

(* Print the links we found. *)
let () =
  List.iter
    (function
     | A href -> Printf.printf "ANCHOR: %s\n" href
     | IMG src -> Printf.printf "IMAGE: %s\n" src)
    (find_links elements)

(*-----*)

#!/usr/bin/ocaml

```

```

(* xurl - extract unique, sorted list of links from URL *)

#use "topfind";;
#require "netclient";;

open Http_client.Convenience
open Nethtml

let rec fold_elements f accu = function
  | Element (_, _, children) as element ->
    let accu =
      List.fold_right
        (fun child accu ->
          fold_elements f accu child)
        children
        accu in
    f accu element
  | other -> accu

type link = A of string | IMG of string

let find_links elements =
  List.flatten
    (List.map
      (fold_elements
        (fun accu element ->
          match element with
          | Element ("a", attribs, _) ->
            (try A (List.assoc "href" attribs) :: accu
             with Not_found -> accu)
          | Element ("img", attribs, _) ->
            (try IMG (List.assoc "src" attribs) :: accu
             with Not_found -> accu)
          | _ -> accu)
        []))
      elements)

let base_url = Sys.argv.(1)
let elements = parse (new Netchannels.input_string (http_get base_url))
let url_syntax = Hashtbl.find Neturl.common_url_syntax "http"
let url_syntax =
  {url_syntax with
   Neturl.url_enable_fragment = Neturl.Url_part_allowed}
let url_syntax = Neturl.partial_url_syntax url_syntax

module StringSet = Set.Make(String)

let () =
  StringSet.iter print_endline
    (List.fold_left
      (fun accu s ->
        try
          StringSet.add

```

```

        (Neturl.string_of_url
         (Neturl.apply_relative_url
          (Neturl.url_of_string url_syntax base_url)
          (Neturl.url_of_string url_syntax s)))
      accu
    with Neturl.Malformed_URL ->
      Printf.eprintf "Malformed URL: %s\n%!" s;
      accu)
  StringSet.empty
  (List.map
   (function
    | A href -> href
    | IMG src -> src)
   (find_links elements)))

```

## 20.4 Converting ASCII to HTML

```

#!/usr/bin/ocaml
(* text2html - trivial html encoding of normal text *)

#use "topfind";;
#require "str";;
#require "netstring";;

let line_stream_of_channel channel =
  Stream.from
    (fun _ -> try Some (input_line channel) with End_of_file -> None)

let paragraph_stream_of_channel channel =
  let lines = line_stream_of_channel channel in
  let rec next para_lines i =
    match Stream.peek lines, para_lines with
    | None, [] -> None
    | Some "", [] -> Stream.junk lines; next para_lines i
    | Some "", _
    | None, _ -> Some (String.concat "\n" (List.rev para_lines))
    | Some line, _ -> Stream.junk lines; next (line :: para_lines) i in
  Stream.from (next [])

let chop s =
  if s = "" then s else String.sub s 0 (String.length s - 1);;

let substitutions =
  [
    (* embedded URL (good) or guessed URL (bad) *)
    Str.regexp "\\(<URL:[^>]+>\\)\\\\\\\\(http:[^ \\n\\r\\t]+\\\\)",
    (fun s ->
      let s =
        if s.[0] = '<'
        then String.sub s 5 (String.length s - 6)
        else s in
      [Nethtml.Element ("a", ["href", s], [Nethtml.Data s])]);
  ]

```

```

    (* this is bold here *)
    Str.regexp "\\*[^*]+\\*",
    (fun s -> [Nethtml.Element ("strong", [], [Nethtml.Data s])]);

    (* this is italics here *)
    Str.regexp "_[^_]+_",
    (fun s -> [Nethtml.Element ("em", [], [Nethtml.Data s])]);
]

let substitute regexp func data =
  List.flatten
    (List.map
      (function
        | Str.Text s -> [Nethtml.Data s]
        | Str.Delim s -> func s)
      (Str.full_split regexp data))

let rec map_data f list =
  List.flatten
    (List.map
      (function
        | Nethtml.Data data -> f data
        | Nethtml.Element (name, attrs, children) ->
          [Nethtml.Element (name, attrs, map_data f children)])
      list)

let text2html text =
  (* Create the initial HTML tree. *)
  let html = [Nethtml.Data text] in

  (* Split text into lines. *)
  let html =
    List.flatten
      (List.map
        (function
          | Nethtml.Data data ->
              List.map
                (fun line -> Nethtml.Data (line ^ "\n"))
                ("" :: Str.split (Str.regexp "\n") data)
          | Nethtml.Element _ as e -> [e])
        html) in

  (* Perform inline substitutions. *)
  let html =
    List.fold_right
      (fun (regexp, func) ->
        map_data (substitute regexp func))
      substitutions
      html in

  (* Add line breaks to quoted text. *)
  let html =
    List.flatten

```

```

(List.map
  (function
    | Nethtml.Data line when line.[0] = '>' ->
      [Nethtml.Data (chop line);
       Nethtml.Element ("br", [], []);
       Nethtml.Data "\n"]
    | Nethtml.Data line -> [Nethtml.Data line]
    | Nethtml.Element _ as e -> [e])
  html) in

  (* Return the finished document. *)
  html

let buffer = Buffer.create 0
let channel = new Netchannels.output_buffer buffer
let write html = Nethtml.write channel (Nethtml.encode html)
let paragraphs = paragraph_stream_of_channel stdin

(* Main loop *)
let () =
  let first = ref true in
  Stream.iter
    (fun para ->
      if !first then first := false
      else write [Nethtml.Data "\n\n"];
      (* Paragraphs beginning with whitespace are wrapped in <pre> *)
      let tag, body =
        if String.length para > 0 && String.contains " \t" para.[0]
        then "pre", [Nethtml.Data "\n";
                     Nethtml.Data para; (* indented verbatim *)
                     Nethtml.Data "\n"]
        else "p", text2html para in (* add paragraph tag *)
      write [Nethtml.Element (tag, [], body)])
    paragraphs;
  print_endline (Buffer.contents buffer)

(*-----*)

(* To format mail headers as a table, add the following just before
   the main loop. *)
let () =
  let colon_delim = Str.regexp "[ \t]*:[ \t]*" in
  let continuation = Str.regexp "\n[ \t]+" in
  try
    let headers = Stream.next paragraphs in
    let headers = Str.global_replace continuation " " headers in
    let lines = Str.split (Str.regexp "\n") headers in
    let rows =
      List.flatten
        (List.map
          (fun line ->
            (* parse heading *)
            let key, value =

```

```

        match Str.bounded_split_delim colon_delim line 2 with
        | [key; value] -> key, value
        | _ -> "", line in
[Nethtml.Element
  ("tr", [],
    [Nethtml.Element
      ("th", ["align", "left"], [Nethtml.Data key]);
      Nethtml.Element
        ("td", [], [Nethtml.Data value])]);
  Nethtml.Data "\n"])
lines) in
write [Nethtml.Element ("table", [], Nethtml.encode rows);
  Nethtml.Element ("hr", [], [])];
  Nethtml.Data "\n\n"]
with Stream.Failure -> ()

```

## 20.5 Converting HTML to ASCII

```

#load "unix.cma";;

let slurp_channel channel =
  let buffer_size = 4096 in
  let buffer = Buffer.create buffer_size in
  let string = String.create buffer_size in
  let chars_read = ref 1 in
  while !chars_read <> 0 do
    chars_read := input channel string 0 buffer_size;
    Buffer.add_substring buffer string 0 !chars_read
  done;
  Buffer.contents buffer

let () =
  let process = Unix.open_process_in ("lynx -dump " ^ filename) in
  let ascii = slurp_channel process in
  ignore (Unix.close_process_in process);
  (* ... *)

```

## 20.6 Extracting or Removing HTML Tags

```

(* Nethtml can be used to safely isolate and extract the text elements
   from an HTML document. *)
#use "topfind";;
#require "netstring";;

(* Load the HTML document. *)
let channel = new Netchannels.input_channel (open_in filename)
let html = Nethtml.parse channel
let () = channel#close_in ()

(* Convert the document to plain text. *)
let plain_text =
  let text = ref [] in
  let rec loop html =

```

```

List.iter
  (function
    | Nethtml.Data s -> text := s :: !text
    | Nethtml.Element (_, _, children) -> loop children)
  html in
loop (Nethtml.decode html);
String.concat "" (List.rev !text)

(*-----*)

#!/usr/bin/ocaml
(* htitle - get html title from URL *)

#use "topfind";;
#require "str";;
#require "netclient";;

open Http_client.Convenience

let ltrim = Str.global_replace (Str.regexp "^[\r\n\t\x00\x0B]*") ""
let rtrim = Str.global_replace (Str.regexp "[\r\n\t\x00\x0B]*$") ""
let trim s = rtrim (ltrim s)

let find_title html =
  let title = ref "" in
  let rec loop = function
    | Nethtml.Element ("title", _, Nethtml.Data data :: _) ->
      title := trim data; raise Exit
    | Nethtml.Element (_, _, children) -> List.iter loop children
    | _ -> () in
  (try List.iter loop html with Exit -> ());
  !title

let urls =
  if Array.length Sys.argv > 1
  then List.tl (Array.to_list Sys.argv)
  else (Printf.eprintf "usage: %s url ...\n" Sys.argv.(0); exit 1)

let () =
  List.iter
    (fun url ->
      print_string (url ^ ": ");
      try
        let res = http_get url in
        let ch = new Netchannels.input_string res in
        let html = Nethtml.parse ch in
        print_endline (find_title html)
      with
        | Http_client.Http_error (status, _) ->
          Printf.printf "%d %s\n" status
            (Nethhttp.string_of_http_status
             (Nethhttp.http_status_of_int status))
        | Failure s -> print_endline s)

```

```
urls
```

## 20.7 Finding Stale Links

```
#!/usr/bin/ocaml
(* churl - check urls *)

#use "topfind";;
#require "netclient";;

open Http_client.Convenience
open Nethtml

let rec fold_elements f accu = function
  | Element (_, _, children) as element ->
    let accu =
      List.fold_right
        (fun child accu ->
          fold_elements f accu child)
        children
        accu in
    f accu element
  | other -> accu

type link = A of string | IMG of string

let find_links elements =
  List.flatten
    (List.map
      (fold_elements
        (fun accu element ->
          match element with
          | Element ("a", attribs, _) ->
            (try A (List.assoc "href" attribs) :: accu
             with Not_found -> accu)
          | Element ("img", attribs, _) ->
            (try IMG (List.assoc "src" attribs) :: accu
             with Not_found -> accu)
          | _ -> accu)
        []))
      elements)

let check_url url =
  Printf.printf " %s: %s\n%!" url
  (match (http_head_message url)#status with
   | 'Successful -> "OK"
   | _ -> "BAD")

let () =
  if Array.length Sys.argv <> 2
  then (Printf.eprintf "usage: %s <start_url>\n" Sys.argv.(0); exit 1)

let base_url = Sys.argv.(1)
```

```

let elements = parse (new Netchannels.input_string (http_get base_url))
let url_syntax = Hashtbl.find Neturl.common_url_syntax "http"
let url_syntax =
  {url_syntax with
    Neturl.url_enable_fragment = Neturl.Url_part_allowed}
let url_syntax = Neturl.partial_url_syntax url_syntax

module StringSet = Set.Make(String)

let () =
  print_endline (base_url ^ ":");
  StringSet.iter check_url
    (List.fold_left
      (fun accu s ->
        try
          StringSet.add
            (Neturl.string_of_url
              (Neturl.apply_relative_url
                (Neturl.url_of_string url_syntax base_url)
                (Neturl.url_of_string url_syntax s)))
            accu
          with Neturl.Malformed_URL ->
            Printf.eprintf "Malformed URL: %s\n%!" s;
            accu)
        StringSet.empty
      (List.map
        (function
          | A href -> href
          | IMG src -> src)
        (find_links elements)))

```

## 20.8 Finding Fresh Links

```

#!/usr/bin/ocaml
(* surl - sort URLs by their last modification date *)

#use "topfind";;
#require "netclient";;

open Http_client.Convenience

let dates = ref []

let () =
  try
    while true do
      let url = input_line stdin in
      let call = http_head_message url in
      let date =
        try Some (Netdate.parse
          (call#response_header#field "Last-Modified"))
        with Not_found -> None in
      dates := (date, url) :: !dates

```

```

    done
  with End_of_file -> ()

let () =
  List.iter
    (fun (date, url) ->
      Printf.printf "%-25s %s\n"
        (match date with
         | Some date -> Netdate.format "%a %b %d %H:%M:%S %Y" date
         | None -> "<NONE SPECIFIED>")
        url)
    (List.rev (List.sort compare !dates))

```

## 20.9 Creating HTML Templates

```

(* Template replacement using regular expressions from the Str module. *)
#load "str.cma";;

let slurp_channel channel =
  let buffer_size = 4096 in
  let buffer = Buffer.create buffer_size in
  let string = String.create buffer_size in
  let chars_read = ref 1 in
  while !chars_read <> 0 do
    chars_read := input channel string 0 buffer_size;
    Buffer.add_substring buffer string 0 !chars_read
  done;
  Buffer.contents buffer

let slurp_file filename =
  let channel = open_in_bin filename in
  let result =
    try slurp_channel channel
    with e -> close_in channel; raise e in
  close_in channel;
  result

let template_regexp = Str.regexp "%%\([^\%]+\)\%"

let template filename fillings =
  let text = slurp_file filename in
  let eval s =
    try Hashtbl.find fillings s
    with Not_found -> "" in
  let replace _ =
    eval (Str.matched_group 1 text) in
  Str.global_substitute template_regexp replace text

(*-----*)

(* Alternative implementation: a hand-written stream parser. This version
avoids loading the whole template into memory, so it is efficient for
large files. *)

```

```

let template filename fillings =
  let f = open_in filename in
  try
    let buffer = Buffer.create (in_channel_length f) in
    let text = Stream.of_channel f in
    let eval s =
      try Hashtbl.find fillings s
      with Not_found -> "" in
    let rec search () =
      match Stream.peek text with
      | None -> ()
      | Some '%' ->
        Stream.junk text;
        (match Stream.peek text with
         | None ->
           Buffer.add_char buffer '%';
           search ()
         | Some '%' ->
           Stream.junk text;
           replace ""
         | Some c ->
           Stream.junk text;
           Buffer.add_char buffer '%';
           Buffer.add_char buffer c;
           search ())
      | Some c ->
        Stream.junk text;
        Buffer.add_char buffer c;
        search ()
    and replace acc =
      match Stream.peek text with
      | None ->
        Buffer.add_string buffer "%%";
        Buffer.add_string buffer acc
      | Some '%' ->
        Stream.junk text;
        (match Stream.peek text with
         | None ->
           Buffer.add_string buffer "%%";
           Buffer.add_string buffer acc;
           Buffer.add_char buffer '%'
         | Some '%' ->
           Stream.junk text;
           Buffer.add_string buffer (eval acc);
           search ()
         | Some c ->
           Stream.junk text;
           replace (acc ^ "%" ^ (String.make 1 c)))
      | Some c ->
        Stream.junk text;
        replace (acc ^ (String.make 1 c)) in
    search ();

```

```

        close_in f;
        Buffer.contents buffer
with e ->
    close_in f;
    raise e

(*-----*)

(* simple.template contains the following:

<!-- simple.template for internal template() function -->
<HTML><HEAD><TITLE>Report for %%username%%</TITLE></HEAD>
<BODY><H1>Report for %%username%%</H1>
%%username%% logged in %%count%% times, for a total of %%total%% minutes.

*)

let () =
    let fields = Hashtbl.create 3 in
    Hashtbl.replace fields "username" whats_his_name;
    Hashtbl.replace fields "count" (string_of_int login_count);
    Hashtbl.replace fields "total" (string_of_int minute_used);
    print_endline (template "simple.template" fields)

(* Output:

<!-- simple.template for internal template() function -->
<HTML><HEAD><TITLE>Report for ramen</TITLE></HEAD>
<BODY><H1>Report for ramen</H1>
ramen logged in 42 times, for a total of 123 minutes.

*)

(*-----*)

(* userrep - report duration of user logins using SQL database *)

let process (cgi : Netcgi.cgi) =
    cgi#set_header ~content_type:"text/html" ();
    begin
        match cgi#argument_value "username" with
        | "" ->
            cgi#out_channel#output_string "No username"
        | user ->
            let db =
                Mysql.quick_connect
                    ~user:"user"
                    ~password:"seekritpassword"
                    ~database:"connections" () in
            let sql = Printf.sprintf "
                SELECT COUNT(duration),SUM(duration)
                FROM logins WHERE username='%s'
            " (Mysql.escape user) in

```

```

let result = Mysql.exec db sql in
let default d = function Some x -> x | None -> d in
let (count, total) =
  match Mysql.fetch result with
  | None -> ("0", "0")
  | Some row ->
    (default "0" row.(0),
     default "0" row.(1)) in
(* template defined in the solution above *)
let tpl = template "report.tpl" in
let vars = Hashtbl.create 3 in
Hashtbl.replace vars "username" user;
Hashtbl.replace vars "count" count;
Hashtbl.replace vars "total" total;
cgi#out_channel#output_string (tpl vars)
end;
cgi#out_channel#commit_work ()

let () =
  let config = Netcgi.default_config in
  let buffered _ ch = new Netchannels.buffered_trans_channel ch in
  Netcgi.cgi.run ~config ~output_type:(`Transactional buffered) process

```

## 20.10 Mirroring Web Pages

```

#use "topfind";;
#require "unix";;
#require "netclient";;

let mirror url file =
  let call = new Http_client.get url in
  begin
    try
      let mtime = (Unix.stat file).Unix.st_mtime in
      let date = Netdate.mk_mail_date mtime in
      call#set_req_header "If-Modified-Since" date
      with Unix.Unix_error _ -> ()
    end;
    call#set_response_body_storage (`File (fun () -> file));
    let pipeline = new Http_client.pipeline in
    pipeline#add call;
    pipeline#run ();
    if call#response_status = `Ok
    then (let date =
          Netdate.parse
            (call#response_header#field "Last-Modified") in
         Unix.utimes file 0.0 (Netdate.since_epoch date));
        call#response_status

```

## 20.11 Creating a Robot

```

#load "str.cma";;

```

```

(* Parse "robots.txt" content from a stream of lines and return a
   list of user agents and a multi-valued hash table containing the
   rules for each user agent. *)
let parse_robots =
  let module S = Set.Make(struct
    type t = string
    let compare = compare
  end) in

  (* Precompile regular expressions. *)
  let comments = Str.regexp "#.*" in
  let leading_white = Str.regexp "[ \t]+" in
  let trailing_white = Str.regexp "[ \t\r]+$" in
  let colon_delim = Str.regexp "[ \t]*:[ \t]*" in

  fun stream ->
    let user_agent = ref "*" in
    let user_agents = ref (S.singleton "*") in
    let rules = Hashtbl.create 0 in

    Stream.iter
      (fun s ->
        let s = Str.replace_first comments "" s in
        let s = Str.replace_first leading_white "" s in
        let s = Str.replace_first trailing_white "" s in
        if String.length s > 0 then
          match Str.bounded_split_delim colon_delim s 2 with
          | ["User-agent"; value] ->
            (* Found a new User-agent. *)
            user_agent := value;
            user_agents := S.add value !user_agents
          | ["Sitemap"; value] ->
            (* Sitemaps are always global. *)
            Hashtbl.add rules "*" ("Sitemap", value)
          | [key; value] ->
            (* Found a rule for the current User-agent. *)
            Hashtbl.add rules !user_agent (key, value)
          | _ -> failwith s)
      stream;
    S.elements !user_agents, rules

  (* Produce a stream of lines from an input channel. *)
  let line_stream_of_channel channel =
    Stream.from
      (fun _ -> try Some (input_line channel) with End_of_file -> None)

  (* Produce a stream of lines from a string in memory. *)
  let line_stream_of_string string =
    Stream.of_list (Str.split (Str.regexp "\n") string)

  (*-----*)

  (* Use Ocamlnet to retrieve a "robots.txt" file and print its rules. *)

```

```

#use "topfind";;
#require "netclient";;
open Http_client.Convenience

let agents, rules =
  parse_robots
    (line_stream_of_string
      (http_get "http://sourceforge.net/robots.txt"))

let () =
  List.iter
    (fun agent ->
      Printf.printf "User-agent: %s\n" agent;
      List.iter
        (fun (key, value) ->
          Printf.printf "\t%s: %s\n" key value)
          (Hashtbl.find_all rules agent))
    agents

```

## 20.12 Parsing a Web Server Log File

```

(* Use the Weblogs library by Richard Jones:
   http://merjic.com/developers/weblogs

```

```

You will also need the HostIP library:
http://merjic.com/developers/hostip *)

```

```

let log = Weblogs.import_file "/var/log/apache2/access.log"

```

```

let () =
  Array.iter
    (fun {Weblogs.src_ip=client;
          remote_username=identuser;
          username=authuser;
          t=datetime;
          http_method=method';
          full_url=url;
          http_version=protocol;
          rcode=status;
          size=bytes;
          (* Many more fields are available.
             See Weblogs API documentation for details. *)
        } ->
      (* ... *)
    )
    log

```

## 20.13 Processing Server Logs

```

#!/usr/bin/ocaml
(* sumwww - summarize web server log activity *)

```

```

#use "topfind";;
#require "weblogs";;

open Weblogs

let file =
  if Array.length Sys.argv = 2
  then Sys.argv.(1)
  else (Printf.eprintf "usage: %s <logfile>\n" Sys.argv.(0);
        exit 1)

let format_date = CalendarLib.Printer.CalendarPrinter.sprint "%d/%b/%Y"

let incr_hash hash key by =
  Hashtbl.replace hash key
    (try Hashtbl.find hash key + by
     with Not_found -> by)

let count_hash hash =
  let count = ref 0 in
  Hashtbl.iter (fun _ _ -> incr count) hash;
  !count

let add_hash dest src =
  Hashtbl.iter (incr_hash dest) src

let lastdate = ref ""

let count = ref 0
let posts = ref 0
let homes = ref 0
let bytesum = ref 0l
let hosts = ref (Hashtbl.create 0)
let whats = ref (Hashtbl.create 0)

let sumcount = ref 0
let allposts = ref 0
let allhomes = ref 0
let bytesumsum = ref 0l
let allhosts = ref (Hashtbl.create 0)
let allwhats = ref (Hashtbl.create 0)

(* display the tallies of hosts and URLs *)
let write_report () =
  Printf.printf "%s %7d %8d %8d %7d %7d %14ld\n%!"
    !lastdate (count_hash !hosts) !count (count_hash !whats)
    !posts !homes !bytesum;

  (* add to summary data *)
  sumcount := !sumcount + !count;
  bytesumsum := Int32.add !bytesumsum !bytesum;
  allposts := !allposts + !posts;
  allhomes := !allhomes + !homes;

```

```

(* reset daily data *)
count := 0;
posts := 0;
homes := 0;
bytesum := 0;
add_hash !allhosts !hosts;
add_hash !allwhats !whats;
Hashtbl.clear !hosts;
Hashtbl.clear !whats

(* read log file and tally hits from the host and to the URL *)
let daily_logs () =
  let log = import_file file in
  print_endline
  "   Date      Hosts  Accesses  Unidocs  POST   Home      Bytes";
  print_endline
  "-----";
  Array.iter
    (fun row ->
      let date = format_date row.t in
      let host = row.src_ip in
      let what = row.url in
      let post = row.http_method = POST in
      let home = what = "/" in
      let bytes = match row.size with Some n -> n | None -> 0 in
      if !lastdate = "" then lastdate := date;
      if !lastdate <> date then write_report ();
      lastdate := date;
      incr count;
      if post then incr posts;
      if home then incr homes;
      incr_hash !hosts host 1;
      incr_hash !whats what 1;
      bytesum := Int32.add !bytesum (Int32.of_int bytes))
    log;
  if !count > 0 then write_report ()

let summary () =
  lastdate := "Grand Total";
  count := !sumcount;
  bytesum := !bytesumsum;
  hosts := !allhosts;
  posts := !allposts;
  whats := !allwhats;
  homes := !allhomes;
  write_report ()

let () =
  daily_logs ();
  summary ();

```

```

    exit 0

(*-----*)

#!/usr/bin/ocaml
(* aprept - report on Apache logs *)

#use "topfind";;
#require "weblogs";;

open Weblogs

let file =
  if Array.length Sys.argv = 2
  then Sys.argv.(1)
  else (Printf.eprintf "usage: %s <logfile>\n" Sys.argv.(0);
        exit 1)

let log = import_file file
let conn = HostIP.connection ()

let incr_hash hash key by =
  Hashtbl.replace hash key
    (try Hashtbl.find hash key + by
     with Not_found -> by)

let report_countries () =
  let total = ref 0 in
  let countries = Hashtbl.create 0 in
  Array.iter
    (fun row ->
      let country =
        match HostIP.get_country_name conn row.src_ip with
        | Some country -> country
        | None -> "UNKNOWN" in
      incr_hash countries country 1;
      incr total)
  log;
  let country_records = ref [] in
  Hashtbl.iter
    (fun country count ->
      country_records := (count, country) :: !country_records)
  countries;
  print_endline "Domain                Records";
  print_endline "=====";
  List.iter
    (fun (count, country) ->
      Printf.printf "%18s %5d %5.2f%%\n%!"
        country count (float count *. 100. /. float !total))
    (List.rev (List.sort compare !country_records))

let report_files () =
  let total = ref 0 in

```

```

let totalbytes = ref 0l in
let bytes = Hashtbl.create 0 in
let records = Hashtbl.create 0 in
Array.iter
  (fun row ->
    let file = row.url in
    let size = match row.size with Some n -> n | None -> 0 in
    incr_hash bytes file size;
    incr_hash records file 1;
    totalbytes := Int32.add !totalbytes (Int32.of_int size);
    incr total)
  log;
let file_records = ref [] in
Hashtbl.iter
  (fun file size ->
    let count = Hashtbl.find records file in
    file_records := (file, size, count) :: !file_records)
  bytes;
print_endline
  "File                Bytes                Records";
print_endline
  "=====";
List.iter
  (fun (file, size, count) ->
    Printf.printf "%-22s %10d %5.2f%% %9d %5.2f%%\n%"
      file size
      (float size *. 100. /. Int32.to_float !totalbytes)
      count
      (float count *. 100. /. float !total))
    (List.sort compare !file_records)

let () =
  report_countries ();
  print_newline ();
  report_files ()

```

## 20.14 Program: htmlsub

```

#!/usr/bin/ocaml
(* htmlsub - make substitutions in normal text of HTML files *)

#use "topfind";;
#require "str";;
#require "netstring";;

let usage () =
  Printf.eprintf "Usage: %s <from> <to> <file>...\n" Sys.argv.(0);
  exit 1

let from, to', files =
  match List.tl (Array.to_list Sys.argv) with
  | from :: to' :: files -> from, to', files
  | _ -> usage ()

```

```

let rec map_data f = function
  | Nethtml.Data data -> Nethtml.Data (f data)
  | Nethtml.Element (name, attribs, children) ->
      Nethtml.Element (name, attribs, List.map (map_data f) children)

let regexp = Str.regexp_string from
let buffer = Buffer.create 0
let out_channel = new Netchannels.output_buffer buffer
let write_html = Nethtml.write out_channel (Nethtml.encode html)

let () =
  List.iter
    (fun file ->
      let in_channel = new Netchannels.input_channel (open_in file) in
      let html = Nethtml.decode (Nethtml.parse in_channel) in
      in_channel#close_in ();
      write (List.map (map_data (Str.global_replace regexp to')) html))
    files;
  print_endline (Buffer.contents buffer)

```

## 20.15 Program: hrefsub

```

#!/usr/bin/ocaml
(* hrefsub - make substitutions in <A HREF="..."> fields of HTML files *)

#use "topfind";;
#require "str";;
#require "netstring";;

let usage () =
  Printf.eprintf "Usage: %s <from> <to> <file>...\n" Sys.argv.(0);
  exit 1

let from, to', files =
  match List.tl (Array.to_list Sys.argv) with
  | from :: to' :: files -> from, to', files
  | _ -> usage ()

let rec map_attr tag attr f = function
  | Nethtml.Data _ as d -> d
  | Nethtml.Element (name, attribs, children)
      when tag = name && List.mem_assoc attr attribs ->
      let value = List.assoc attr attribs in
      Nethtml.Element (name,
        (attr, f value)
        :: List.remove_assoc attr attribs,
        List.map (map_attr tag attr f) children)
  | Nethtml.Element (name, attribs, children) ->
      Nethtml.Element (name,
        attribs,
        List.map (map_attr tag attr f) children)

```

```

let regexp = Str.regexp_string from
let buffer = Buffer.create 0
let out_channel = new Netchannels.output_buffer buffer
let write_html = Nethtml.write out_channel (Nethtml.encode html)

let () =
  List.iter
    (fun file ->
      let in_channel = new Netchannels.input_channel (open_in file) in
      let html = Nethtml.decode (Nethtml.parse in_channel) in
      in_channel#close_in ();
      write (List.map (map_attr "a" "href"
                          (Str.global_replace regexp to')) html))
    files;
  print_endline (Buffer.contents buffer)

```

Generated by GNU enscript 1.6.4.