15.7 REGULAR EXPRESSIONS

Ever wanted to know what a regex looked like? You'll need perl compiled with the DEBUGGING flag for this one:

```
> perl -Dr -e '/^pe(a)*rl$/i'
Compiling REx `^pe(a)*rl$'
size 17 first at 2
rarest char
 at 0
   1: BOL(2)
   2: EXACTF <pe>(4)
   4: CURLYN[1] {0,32767}(14)
   6:   NOTHING(8)
   8:   EXACTF <a>(0)
  12:   WHILEM(0)
  13: NOTHING(14)
  14: EXACTF <rl>(16)
  16: EOL(17)
  17: END(0)
floating `'$ at 4..2147483647 (checking floating) stclass `EXACTF <pe>'
anchored(BOL) minlen 4
Omitting $` $& $' support.

EXECUTING...

Freeing REx: `^pe(a)*rl$'
```

Did you really want to know? :-) For more gory details on getting regular expressions to work, have a look at *perlre*, *perlretut*, and to decode the mysterious labels (BOL and CURLYN, etc. above), see *perldebguts*.

15.8 OUTPUT TIPS

To get all the output from your error log, and not miss any messages via helpful operating system buffering, insert a line like this, at the start of your script:

```
$|=1;
```

To watch the tail of a dynamically growing logfile, (from the command line):

```
tail -f $error_log
```

Wrapping all die calls in a handler routine can be useful to see how, and from where, they're being called, *perlvar* has more information:

```
BEGIN { $SIG{__DIE__} = sub { require Carp; Carp::confess(@_) } }
```

Various useful techniques for the redirection of STDOUT and STDERR filehandles are explained in *perlopentut* and *perlfaq8*.

15.9 CGI

Just a quick hint here for all those CGI programmers who can't figure out how on earth to get past that 'waiting for input' prompt, when running their CGI script from the command-line, try something like this:

```
> perl -d my_cgi.pl -nodebug
```

Of course *CGI* and *perlfaq9* will tell you more.