```
 1: // $Id: bcat.c,v 1.7 2012-10-23 17:42:37-07 - - $
 2:
 3: //
 4: // NAME
 5: //    bcat - concatenate and display files
 6: //
 7: // SYNOPSIS
 8: //    bcat [filename...]
 9: //
10: // DESCRIPTION
11: //    The bcat utility reads each file in sequence and writes it
12: //    to stdout.  If any filename is given as the single character
13: //    "-", stdin is read at that point.  If no filenames are given
14: //    then stdin is read as the only file.
15: //
16:
17: #include <errno.h>
18: #include <libgen.h>
19: #include <stdbool.h>
20: #include <stdio.h>
21: #include <stdlib.h>
22: #include <string.h>
23: #include <unistd.h>
24:
25: char *progname = NULL;          // Name of the program being run.
26: int exit_status = EXIT_SUCCESS; // Assume successful completion.
27:
28: struct options {                // Structure to hold options.
29:    bool moretitles;             // Print titles in more-style
30:    bool numberlines;            // Output line numbers for each line.
31:    bool squeeze;                // Squeeze multiple empty lines.
32: };
33:
```

```
34:
35: //
36: // cat -
37: // Copy the contents of an already-opened file to stdout.
38: //
39:
40: void catfile (FILE *input, char *filename, struct options *opts) {
41:    char buffer[4096];
42:    for (;;) {
43:       char *bufrc = fgets (buffer, sizeof buffer, input);
44:       if (bufrc == NULL) break;
45:       printf ("%s", buffer);
46:    };
47: }
48:
```

```
49:
50: // scanoptions -
51: // Scan the options and initialize a struct options.
52: //
53:
54: void scan_options (int argc, char **argv, struct options *opts) {
55:    memset (opts, 0, sizeof (struct options));
56:    opterr = false;
57:    for (;;) {
58:       int opt = getopt (argc, argv, "mns");
59:       if (opt == EOF) break;
60:       switch (opt) {
61:          case 'm':
62:             opts->moretitles = true;
63:             break;
64:          case 'n':
65:             opts->numberlines = true;
66:             break;
67:          case 's':
68:             opts->squeeze = true;
69:             break;
70:          default :
71:             fflush (NULL);
72:             fprintf (stderr, "%s: -%c: invalid option\n",
73:                      progname, optopt);
74:             fflush (NULL);
75:             exit_status = EXIT_FAILURE;
76:        }
77:    }
78: }
79:
```

```
 80:
 81: //
 82: // main -
 83: // Loop over files, if any, and cat each of them to stdout.
 84: // Print error messages if appropriate.
 85: //
 86:
 87: int main (int argc, char **argv) {
 88:    exit_status = EXIT_SUCCESS;
 89:    progname = basename (argv[0]);
 90:    struct options opts;
 91:    scan_options (argc, argv, &opts);
 92:    if (optind == argc) {
 93:       catfile (stdin, "-", &opts);
 94:    }else{
 95:       for (int argi = optind; argi < argc; ++argi) {
 96:          char *filename = argv[argi];
 97:          if (strcmp (filename, "-") == 0) {
 98:             catfile (stdin, "-", &opts);
 99:          }else{
100:             FILE *input = fopen (filename, "r");
101:             if (input == NULL) {
102:                fflush (NULL);
103:                fprintf (stderr, "%s: %s: %s\n", progname,
104:                         argv[argi], strerror (errno));
105:                fflush (NULL);
106:                exit_status = EXIT_FAILURE;
107:             }else{
108:                catfile (input, filename, &opts);
109:                fclose (input);
110:             };
111:          };
112:       };
113:    };
114:    return exit_status;
115: }
116:
```

```
 1: /*
 2: ****************************************************************
 3:
 4: Whenever a man page is referenced, read it online.  For example,
 5: when we refer to 'stdio(3c)', you can read it with ''man -s 3C
 6: stdio''.
 7:
 8: As described in stdio(3c), there are three FILE* handles that
 9: are always opened when a program starts:  'stdin', 'stdout', and
10: 'stderr'.  These are, respectively, the standard input, standard
11: output, and standard error.  Normal output is written to stdout,
12: while error messages are written to stderr.
13:
14: The usual format of an error message is something like:
15: .   progname: object_or_function: reason
16: The reason a system call has failed is given in the external
17: variable 'errno'.  This can be translated into English via
18: strerror(3c).
19:
20: 'fopen(3c)' opens a file and returns a pointer to a 'FILE',
21: given a filename.  'fclose(3c)' closes that file, given a
22: FILE*.  'putchar(3c)' writes one byte to stdout.  'getc(3c)'
23: reads one byte from the FILE* given as an argument and returns
24: an int containing the character, if one exists.  If not, returns
25: EOF (-1).  Note that end of line is signalled by '\n'.  To
26: signal EOF from a Unix terminal, type Control/D as the first
27: character on a line.
28:
29: Strings are represented as arrays of characters.  Each string
30: ends with a null plug ('\0').  'strcmp(3c)' compares two such
31: character strings and returns a number that is <, =, or > 0,
32: depending on the relationship.  See Java's compareTo function.
33:
34: The call fflush (NULL) causes all opened FILE* handles to be
35: flushed.  When a program writes data, it is buffered in memory
36: before being written to the disk.  This causes immediate writing
37: instead of waiting until the buffer is full.  We do this so that
38: anything written to stdout and stderr are properly interleaved.
39:
40: ****************************************************************
41: $Id: comments.txt,v 1.3 2012-10-23 12:24:40-07 - - $
42: */
```

```
1: #!/bin/sh -x
2: # $Id: mk,v 1.4 2012-10-23 12:35:42-07 - - $
3: cid + bcat.c
4: gcc -g -O0 -Wall -Wextra -std=gnu99 bcat.c -o bcat
```

```
 1: #!/usr/bin/perl
 2: # $Id: pcat.perl,v 1.1 2012-10-23 12:35:42-07 - - $
 3: use strict;
 4: use warnings;
 5: use Getopt::Std;
 6:
 7: $0 =~ s|^.*/||;
 8: my $exit_status = 0;
 9: END {exit $exit_status}
10: sub note(@) {print STDERR "@_"};
11: $SIG{'__WARN__'} = sub {note @_; $exit_status = 1};
12: $SIG{'__DIE__'} = sub {warn @_; exit};
13:
14: my %opts;
15: getopts "mns", \%opts;
16: my $colons = ":" x 64;
17: my $s_count = 0;
18:
19: for my $filename (@ARGV ? @ARGV : '-') {
20:    open my $file, "<$filename" or warn "$0: $filename: $!\n" and next;
21:    print "\n$colons\n$filename\n$colons\n\n" if $opts{'m'};
22:    while (defined (my $line = <$file>)) {
23:       next if $opts{'s'}
24:            and ($s_count = $line =~ m/^\n$/ ? $s_count + 1 : 0) >= 2;
25:       printf "%6d  ", $. if $opts{'n'};
26:       print $line;
27:    }
28:    close $file;
29: }
30:
```