```
 1: #!/usr/bin/perl
 2: # $Id: pfmt.perl,v 1.1 2015-12-11 17:06:45-08 - - $
 3: use strict;
 4: use warnings;
 5:
 6: $0 =~ s|^.*/||;
 7: my $exit_status = 0;
 8: END {exit $exit_status}
 9: sub note(@) {print STDERR "@_"};
10: $SIG{'__WARN__'} = sub {note @_; $exit_status = 1};
11: $SIG{'__DIE__'} = sub {warn @_; exit};
12:
13: my $linelen = 65;
14: if (@ARGV and $ARGV[0] =~ m/^-(.+)/) {
15:    $linelen = $1;
16:    die "Usage: $0 [-width] [filename...]\n" if $linelen =~ m/\D/;
17:    shift @ARGV
18: }
19:
20: sub print_paragraph (@) {
21:    my (@words) = @_;
22:    print "\n";
23:    my $char_count = 0;
24:    for my $word (@words) {
25:       if ($char_count == 0) {
26:          print $word;
27:          $char_count = length $word;
28:       }else {
29:          $char_count += 1 + length $word;
30:          if ($char_count > $linelen) {
31:             print "\n", $word;
32:             $char_count = length $word;
33:          }else {
34:             print " ", $word;
35:          }
36:       }
37:    }
38:    print "\n" if $char_count > 0;
39: }
40:
41: push @ARGV, "-" unless @ARGV;
42: for my $filename (@ARGV) {
43:    open my $file, "<$filename" or warn "$0: $filename: $!\n" and next;
44:    my @output_words;
45:    for (;;) {
46:       my $input_line = <$file>;
47:       last unless defined $input_line;
48:       my @input_words = split " ", $input_line;
49:       if (@input_words) {
50:          push @output_words, @input_words;
51:       }else {
52:          print_paragraph @output_words if @output_words;
53:          @output_words = ();
54:       }
55:    }
56:    print_paragraph @output_words;
57:    close $file;
58: }
```

```
59:
```

```
1: #!/bin/sh
2: # $Id: mkp,v 1.1 2015-12-11 17:06:45-08 - - $
3: ./pfmt.perl ../.score/*.dat >pfmt.output1
4: ./pfmt.perl -40 *.java >pfmt.output2
5: mkpspdf pfmt.listing.ps pfmt.perl $0 pfmt.output*
```

```
    1:
    2: This is test file #1. This is test file #1. This is test file #1.
    3: This is test file #1. This is test file #1. This is test file #1.
    4: This is test file #1. This is test file #1.
    5:
    6: It is very regular and is used to check to see if word wrap
    7: works. It is very regular and is used to check to see if word
    8: wrap works. It is very regular and is used to check to see if
    9: word wrap works. It is very regular and is used to check to see
   10: if word wrap works. It is very regular and is used to check to
   11: see if word wrap works. It is very regular and is used to check
   12: to see if word wrap works. It is very regular and is used to
   13: check to see if word wrap works. It is very regular and is used
   14: to check to see if word wrap works.
   15:
   16: Does it work with a one line paragraph?
   17:
   18: $Id: input1.dat,v 1.1 2013-09-24 14:22:42-07 - - $
   19:
   20:
   21: This is another file of test data for test number two. Some lines
   22: are short. Other lines are very long lines, exceeding even the
   23: line length that checksource.perl likes to see and will complain
   24: about.
   25:
   26: Are multiple input blank lines squeezed to a single output blank
   27: line?
   28:
   29: What happens if there is only one word per line.
   30:
   31: $Id: input2.dat,v 1.1 2013-09-24 14:22:42-07 - - $
   32:
   33: This paragraph is indented by a tab. Are tabs deleted at the
   34: front of the line?
   35:
   36: What about spaces? Do they work like pfmt.perl?
   37:
   38: a long word should be on a line by itself
   39: sometimesthereisaverylongwordwhichpokesoutsidethenormalmarginsometimesth
ereisaverylongwordwhichpokesoutsidethenormalmargin
   40: if the word exceeds the margin
   41:
   42: This paragraph has lots of tabs on input. Tabs should be replaced
   43: by spaces on output.
   44:
   45: This paragraph has lots of leading spaces and trailing tabs on
   46: input.
   47:
   48: $Id: input3.dat,v 1.1 2013-09-24 14:22:42-07 - - $
```

```
 1:
 2: // $Id: jarname.java,v 1.1 2015-12-11
 3: 17:06:45-08 - - $ // // // NAME //
 4: jarname - Print out the name of the
 5: current jar file. // // DESCRIPTION //
 6: Makes use of the fact that the
 7: java.class.path, when Java // is run
 8: from a jar, is the name of the jar. //
 9:
10: import static java.lang.System.*;
11:
12: class jarname { public static void main
13: (String[] args) { String jarpath =
14: getProperty ("java.class.path");
15: out.printf ("jarpath = \"%s\"%n",
16: jarpath); int lastslash =
17: jarpath.lastIndexOf ('/'); String
18: jarbase = lastslash < 0 ? jarpath :
19: jarpath.substring (lastslash + 1);
20: out.printf ("jarbase = \"%s\"%n",
21: jarbase); } }
22:
23: //TEST// ./jarname >jartest.out //TEST//
24: mkpspdf jarname.ps jarname.java*
25: jartest*.out
26:
27:
28: // $Id: jcat.java,v 1.1 2015-12-11
29: 17:06:45-08 - - $ // // SYNOPSIS // jcat
30: [filename...] // // DESCRIPTION // The
31: jcat utility functions like cat(1) and
32: copies the contents // of all files to
33: the standard output, with error messages
34: to // the standard error. // // EXIT
35: STATUS // 0 if no errors were detected.
36: // 1 if errors were detected and
37: messages printed. //
38:
39: import java.io.*; import
40: java.util.Scanner; import static
41: java.lang.System.*;
42:
43: class jcat { // Static variables keeping
44: the general status of the program.
45: public static final String JARNAME =
46: get_jarname (); public static final int
47: EXIT_SUCCESS = 0; public static final
48: int EXIT_FAILURE = 1; public static int
49: exit_status = EXIT_SUCCESS;
50:
51: // A basename is the final component of
52: a pathname. // If a java program is run
53: from a jar, the classpath is the //
54: pathname of the jar. static String
55: get_jarname () { String jarpath =
56: getProperty ("java.class.path"); int
57: lastslash = jarpath.lastIndexOf ('/');
58: if (lastslash < 0) return jarpath;
```

```
 59: return jarpath.substring (lastslash +
 60: 1); }
 61:
 62: // Copies a single opened file to
 63: stdout. static void copylines (Scanner
 64: infile) { // Read each line from the
 65: opened file, one after the other. //
 66: Stop the loop at end of file. while
 67: (infile.hasNextLine ()) { String line =
 68: infile.nextLine (); out.printf ("%s%n",
 69: line); } }
 70:
 71: // Open input file and copy contents to
 72: stdout. static void catfile (String
 73: filename) { if (filename.equals ("-")) {
 74: copylines (new Scanner (System.in));
 75: }else { try { Scanner infile = new
 76: Scanner (new File (filename)); copylines
 77: (infile); infile.close (); }catch
 78: (IOException error) { exit_status =
 79: EXIT_FAILURE; err.printf ("%s: %s%n",
 80: JARNAME, error.getMessage ()); } } }
 81:
 82: // Main function scans arguments and
 83: opens/closes files. public static void
 84: main (String[] args) { if (args.length
 85: == 0) { // No files specified on the
 86: command line. catfile ("-"); }else { //
 87: Iterate over each filename given on the
 88: command line. for (int argi = 0; argi <
 89: args.length; ++argi) { catfile
 90: (args[argi]); } } exit (exit_status); }
 91:
 92: }
 93:
 94: //TEST// mkpspdf jcat.ps jcat.java
 95:
 96:
 97: // $Id: parseint.java,v 1.1 2015-12-11
 98: 17:06:45-08 - - $
 99:
100: // // Illustrate try-catch convert args
101: to integers. // Iterate over each
102: element of args and attempt to convert
103: it to // an integer. If it succeeds,
104: print the integer. If not, catch // the
105: error and print an error message. //
106:
107: import static java.lang.System.*;
108:
109: class parseint { public static void main
110: (String[] args) { for (int argi = 0;
111: argi < args.length; ++argi) { String arg
112: = args[argi]; out.printf ("args[%d] =
113: \"%s\": ", argi, arg); try { int value =
114: Integer.parseInt (arg); out.printf ("is
115: int %d%n", value); }catch
116: (NumberFormatException error) {
```

```
117: out.printf ("NumberFormatException:
118: %s%n", error.getMessage()); } } } }
119:
120: //TEST// ./parseint 214748 hello -33 ''
121: 987 >parsetest.out //TEST// mkpspdf
122: parseint.ps parseint.java parsetest.out
123:
```