```
// $Id: wordct.c,v 1.2 2014-04-22 19:38:08-07 - - $

//
// NAME
//    wordct - count lines, words, and characters in files
//
// SYNOPSIS
//    wordct [-lwc] [file...]
//
// DESCRIPTION
//    Print the character (byte), word, and newline counts for
//    each file, and a total line if more than one file is
//    specified.  If no file is specified, or if file is just
//    a -, read stdin.  A word is any white-space delimited
//    sequence of characters.
//
// OPTIONS
//    If no options are specified, print all three counts.
//    If any options are specified, print only those requested.
//    -l  print the line counts
//    -w  print the word counts
//    -c  print the byte counts
//

#include <assert.h>
#include <ctype.h>
#include <errno.h>
#include <libgen.h>
#include <stdarg.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <unistd.h>

char *program_name = NULL;
int exit_status = EXIT_SUCCESS;
const char stdin_name[] = "-";

struct options {
   bool lines;
   bool words;
   bool chars;
   int file_count;
   char **file_names;
};

struct counts {
   size_t lines;
   size_t words;
   size_t chars;
};

```

```c

void error (const char *format, ...) {
   va_list fmt_args;
   fflush (NULL);
   assert (program_name != NULL);
   fprintf (stderr, "%s: ", program_name);
   va_start (fmt_args, format);
   vfprintf (stderr, format, fmt_args);
   va_end (fmt_args);
   fflush (NULL);
   exit_status = EXIT_FAILURE;
}

void scan_options (int argc, char **argv, struct options *opts) {
   bool all_flags = true;
   opts->chars = opts->words = opts->lines = false;
   opterr = false;
   for (;;) {
      int flag = getopt (argc, argv, "cwl");
      if (flag == EOF) break;
      switch (flag) {
         case 'c': opts->chars = true; all_flags = false; break;
         case 'w': opts->words = true; all_flags = false; break;
         case 'l': opts->lines = true; all_flags = false; break;
         default : error ("-%c: invalid option", optopt); break;
      }
   }
   if (all_flags) opts->chars = opts->words = opts->lines = true;
   opts->file_count = argc - optind;
   opts->file_names = &argv[optind];
}

bool is_plain_file (FILE *file, const char *filename) {
   struct stat stat;
   int rc = fstat (fileno (file), &stat);
   if (rc != 0) {
      error ("%s: %s\n", filename, strerror (errno));
      return false;
   }
   if (S_ISREG (stat.st_mode)) return true;
   const char *reason = S_ISDIR (stat.st_mode)
                      ? "is a directory"
                      : "is not a plain file";
   error ("%s: %s\n", filename, reason);
   return false;
}

```

```

void print_count (struct options *opts, struct counts *count,
                  const char *name) {
   if (opts->lines) printf ("%8zd", count->lines);
   if (opts->words) printf ("%8zd", count->words);
   if (opts->chars) printf ("%8zd", count->chars);
   if (name != NULL) printf (" %s", name);
   printf ("\n");
}

void count_file (FILE *file, const char *filename,
                 struct options *opts, struct counts *totals) {
   if (! is_plain_file (file, filename)) return;
   struct counts file_counts = {0, 0, 0};
   bool spaces = true;
   for (;;) {
      int byte = fgetc (file);
      if (byte == EOF) break;
      ++file_counts.chars;
      if (byte == '\n') ++file_counts.lines;
      if (isspace (byte)) {
         spaces = true;
      }else if (spaces) {
         ++file_counts.words;
         spaces = false;
      }
   }
   print_count (opts, &file_counts, filename);
   totals->lines += file_counts.lines;
   totals->words += file_counts.words;
   totals->chars += file_counts.chars;
}

```

```c

int main (int argc, char **argv) {
   program_name = basename (argv[0]);
   struct options opts = {false, false, false, 0, NULL};
   struct counts totals = {0, 0, 0};
   scan_options (argc, argv, &opts);
   if (opts.file_count == 0) {
      count_file (stdin, NULL, &opts, &totals);
   }else {
      for (int filenr = 0; filenr < opts.file_count; ++filenr) {
         char *filename = opts.file_names[filenr];
         if (strcmp (filename, stdin_name) == 0) {
            count_file (stdin, filename, &opts, &totals);
         }else {
            FILE *file = fopen (filename, "r");
            if (file == NULL) {
               error ("%s: %s", filename, strerror (errno));
            }else {
               count_file (file, filename, &opts, &totals);
               fclose (file);
            }
         }
      }
      if (opts.file_count > 1) print_count (&opts, &totals, "total");
   }
   return exit_status;
}

//TEST// alias grind='valgrind --leak-check=full --show-reachable=yes'
//TEST// grind wordct *.c >wordct.out 2>&1
//TEST// mkpspdf wordct.ps wordct.c* wordct.out*

```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting wordct.c
wordct.c: $Id: wordct.c,v 1.2 2014-04-22 19:38:08-07 - - $
gcc -g -O0 -Wall -Wextra -std=gnu99 wordct.c -o wordct -lm
rm -f wordct.o
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished wordct.c
```

```
==21639== Memcheck, a memory error detector
==21639== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==21639== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==21639== Command: wordct catbychar.c catbyline.c getoptex.c undefvar.c wordct.c
==21639==
      64     196    1561 catbychar.c
      67     230    1795 catbyline.c
      78     232    1936 getoptex.c
      24      82     570 undefvar.c
     166     623    4732 wordct.c
     399    1363   10594 total
==21639==
==21639== HEAP SUMMARY:
==21639==     in use at exit: 0 bytes in 0 blocks
==21639==   total heap usage: 5 allocs, 5 frees, 2,840 bytes allocated
==21639==
==21639== All heap blocks were freed -- no leaks are possible
==21639==
==21639== For counts of detected and suppressed errors, rerun with: -v
==21639== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
```