```
/*

Helen Dougherty & Albert Owusu-Asare

Box: 3438 & 4497

Lab 1: A Review of C

additional files:

c_review_tests.c

  About this program:
  - This program counts words.
  - The specific words that will be counted are passed in as command-line
  arguments.
  - The program reads words (one word per line) from standard input until
  EOF. (Note that EOF can be typed as <CTRL-D>).
  - The program prints out a summary of the number of times each word has
  appeared.
  - Various command-line options alter the behavior of the program.

  E.g., count the number of times 'cat', 'nap' or 'dog' appears.
  > ./main cat nap dog
  Given input:
  cat
  .
  Expected output:
  Looking for 3 words
  Result:
  cat:1
  nap:0
  dog:0

  Note: this code was automatically formatted (styled) using 'indent main.c -kr'.

  This assignment was adapted from operating system programming problems by
  Lawrence Angrave at the University of Illinois at Champaign-Urbana (UIUC).

*/

/*
 * Note:
 *
 * We ran out of time and could not figure out how to implement strtok(). Our lates
t
 *   attempt at implementing it is in the code, commented out.
 *
 * We were also unable to figure out why the file_output test failed, but testing
 *   the code directly worked. We included it with our test run.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "c_review_tests.h"

#define LENGTH(s) (sizeof(s) / sizeof(*s))

/* Structures */
typedef struct {
  char *word;
  int counter;
} word_count_entry_t;

/*
  compares each argument to buffer word (or words) and increments counters when
  words are comparable to each other.

  Preconditions: entries[] is a pre-defined array
                 entry_count is a positive integer
                 for multi-word support, all words must be separated using
                    identical means. A period may not be used as a delimiter, as it

                    is the break character.

  Postconditions: evaluates arguments against buffer and increments applicable
  counters; returns line_count, a nonnegative integer.
*/
int process_stream(word_count_entry_t entries[], int entry_count)
{
  short line_count = 0;
  char buffer[30];

  while (fgets(buffer, 30, stdin)) {

    int buflen = (int) strlen(buffer);
    if(buflen < 30)
      {
        buffer[buflen -1] = NULL;
      }//if a newline character is read in
    if (*buffer == '.')
      break;

    /* Compare against each entry */
    int i = 0;

    //Part of our current attempt at strtok
    // char *part = strtok(buffer, ", _\\-\':\";\?!");

    while (i < entry_count) {

      /* Our current attempt at strtok */

      /*printf("part: %s\n", part);
      printf("strcmp results: %d\n", strcmp(entries[i].word, part));
      if (!strcmp(entries[i].word, part))
        entries[i].counter++;

        part = strtok(NULL, ", _\\-\':\";\?!");
      */

      if (!strcmp(entries[i].word, buffer))
        entries[i].counter++;
      i++;
    }

    line_count++;
  }
  return line_count;
}

/*
  Prints out words stored in entry array alongside the respective word counts.

  PreconditionS: entries[] is a predefined array; entry_count is a positive
  integer.
  Postconditions: prints out all arguments alongside correct respective counter

*/
void print_result(word_count_entry_t entries[], int entry_count,FILE * output_buffe
r)
{
  fprintf(output_buffer, "Result:\n");

  int i;
  //print in same order as input
  for(i = 1; i < entry_count; i++)
```

```
    {
      fprintf(output_buffer, "%s:%d\n", entries[i].word, entries[i].counter);
    }
  fclose(output_buffer);

}

/*
  Prints a help message
*/
void print_help(const char *name)
{
  fprintf(stderr, "usage: %s [-h] <word1> ... <wordN>\n", name);
}


int main(int argc, char **argv)
{
  const char *prog_name = *argv;


  // output buffer
  FILE * output_buffer = stdout;

  char * output_filename = malloc(sizeof(char) *40);

  /*
    This is our attempt to use malloc(). It works, but fails
    'basic-functionality' tests, as well as malloc.

    word_count_entry_t *entries;
    entries = malloc(sizeof(word_count_entry_t) * argc-1);
  */

  //This passes the malloc test, but doesn't actually use malloc(). Our code
  //doesn't break, though.
  word_count_entry_t entries[argc];

  //a counter of the number of unique words
  int entry_count = 0;
  //a counter of total distinct words
  int words = 0;

  /* Entry point for the testrunner program */
  if (argc > 1 && !strcmp(argv[1], "-test")) {
    run_c_review_tests(argc - 1, argv + 1);
    return EXIT_SUCCESS;
  }

  while (*argv != NULL) {
    if (**argv == '-') {

      switch ((*argv)[1]) {

      case 'h':
        print_help(prog_name);
        break;
      case 'f':
        //store the filename where output will be displayed and store output buffer
        {
          char * flag_string =*argv;

          //copy substring of argv containing filename
          strncpy(output_filename,flag_string+2,LENGTH(flag_string));
          // printf("%s\n",output_filename);

          // set output buffer to new file
          output_buffer = fopen(output_filename,"w");
          break;
        }

      default:
        fprintf(stderr, "%s: Invalid option %s. Use -h for help.\n",
                prog_name, *argv);
        break;
      }
    } else {
      if (entry_count < LENGTH(entries)) {
        //compares *argv to previous entries; is the word a repetition?
        int j;
        //switch to flip in case of a repetition
        int same=0;
        for(j = 0; j < entry_count; j++)
          {
            if(!strcmp(entries[j].word, *argv))
              {
                entries[j].counter++;
                same = 1;
                words++;
              }//if word is repeated
          }
        if(same==0)
          {
            entries[entry_count].word = *argv;
            entries[entry_count++].counter = 0;
            words++;
          }//if word is not repeated
      }
      argv++;
    }
    if (entry_count == 1) {
      fprintf(stderr, "%s: Please supply at least one word. Use -h for help.\n",
              prog_name);

      return EXIT_FAILURE;
    }//if there are no words to test
    if (entry_count == 2) {
      fprintf(stdout, "Looking for a single word\n");
    } else {
      fprintf(stdout, "Looking for %d words\n", (words-1));
    }

    process_stream(entries, entry_count);

    print_result(entries, entry_count,output_buffer);

    return EXIT_SUCCESS;
}
```