```c
/*
 * ECE220 Spring 2018 (update from Fall 2005)
 *
 * Program name: word_split.c, an English word splitting program
 *
 * Description: This program splits its input into a list of lower-case
 *              words, with one word per line.  Words are defined as
 *              contiguous sequences of alphabetic characters, hyphens,
 *              and apostrophes.  Words must begin with an alphabetic
 *              character.  All other characters are discarded.
 */

#include <stdint.h>       /* Include C's standard integer header file. */
#include <stdio.h>        /* Include C's standard I/O header file.  */

static const int32_t max_word_len = 500;  /* limit on word length */

/* My favorite exit condition definitions. */
enum {
    EXIT_SUCCEED  = 0,
    EXIT_FAIL     = 1,
    EXIT_BAD_ARGS = 2,
    EXIT_PANIC    = 3
};

/*
 * Function: main
 * Description: read a file one character at a time, break input into
 *              lower-case words (alphabetic, hyphens, or apostrophes),
 *              and print words found on separate lines without eliminating
 *              duplicates.  Hyphens and apostrophes are not allowed to
 *              start words.
 * Parameters: argc -- the number of arguments, including the executable name
 *             argv -- an array of strings containing each argument
 *             argc must equal 2, and the second argument is the file name
 *                from which words are read
 * Return Value: EXIT_SUCCEED for success
 *               EXIT_FAIL if file cannot be opened
 *               EXIT_BAD_ARGS if the wrong number of arguments are given
 */

int
main (int argc, char* argv[])
{
    FILE*   in_file;                    /* input stream           */
    char    buf[max_word_len + 1];      /* holds current word     */
    char*   write;                      /* end of current word    */
    int32_t word_len;                   /* length of current word */
    int32_t a_char;                     /* last character read    */

    /* Program must receive exactly two arguments. */
    if (2 != argc) {
        /* Print an error message.  argv[0] is the executable name. */
        fprintf (stderr, "syntax: %s <file name>\n", argv[0]);
        return EXIT_BAD_ARGS;
    }

    /* Open the file for reading. */
    if (NULL == (in_file = fopen (argv[1], "r"))) {
        /* fopen failed: print an error message to stderr. */
        perror ("open file");
        return EXIT_FAIL;
    }

    /* Initialize the word writing variable to point to the start of
       the word buffer. */
    write = buf;
    word_len = 0;

    /* Read characters until we find the end of the input. */
    while (EOF != (a_char = getc (in_file))) {

        /* If necessary, change input character to lower case. */
        if ('A' <= a_char && 'Z' >= a_char)
            a_char = a_char - 'A' + 'a';

        /* Can character be part of a word? */
        if (('a' <= a_char && 'z' >= a_char) ||
            (0 < word_len && ('-' == a_char || '\'' == a_char))) {

            /* Write the character into our word buffer and increment
               the pointer and counter. */
            *write++ = a_char;
            word_len++;

            /* Do we still have room in the buffer?  If so, read
               another character (skip to next loop iteration). */
            if (max_word_len > word_len)
                continue;
        } else {
            /* Invalid character read.  Is there a word that needs
               to be written out?  If not, skip to next character. */
            if (0 == word_len)
                continue;
        }

        /* Write out the current word, then reset the buffer pointer
           and character count. */
        *write = 0;
        puts (buf);
        write = buf;
        word_len = 0;
    }

    /* Any last words? */
    if (0 < word_len) {
        *write = 0;
        puts (buf);
    }

    /* Close the input file, ignoring any errors. */
    (void)fclose (in_file);

    /* Program finished successfully. */
    return EXIT_SUCCEED;
}
```