```c
/*
 * ECE220 Spring 2018 (update from Fall 2005)
 *
 * Program name: line_sort.c, a sorting program
 *
 * Description: This program alphabetically sorts lines from stdin.
 *              Lines are stored using dynamically allocated memory.
 */

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

static const int32_t max_num_lines = 5000; /* limit on number of lines */
static const int32_t max_line_len  = 500;  /* limit on line length */

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


/* function declarations */

/* read lines from stdin into an array; returns number of lines read */
static int32_t read_lines (char* lines[], int32_t max_lines);

/* sort strings in an array alphabetically using insertion sort */
static void sort_lines (char* lines[], int32_t n_lines);

/* print an array of strings in order to stdout */
static void print_lines (char* const lines[], int32_t n_lines);


/*
 * Function: main
 * Description: read stdin one line at a time, copying the lines
 *              into dynamically allocated memory, then sort and
 *              print the lines
 * Parameters: argc -- the number of arguments, including the executable name
 *             argv -- an array of strings containing each argument
 *             argc must equal 1; no other arguments are allowed
 * Return Value: EXIT_SUCCEED for success
 *               EXIT_BAD_ARGS if the wrong number of arguments are given
 */

int
main (int argc, char* argv[])
{
    char*   lines[max_num_lines];       /* array of lines  */
    int32_t num_lines;                  /* number of lines */

    /* Program must receive exactly one argument. */
    if (1 != argc) {
        /* Print an error message.  argv[0] is the executable name. */
        fprintf (stderr, "syntax: %s\n", argv[0]);
        return EXIT_BAD_ARGS;
    }

    /* Read, sort, and print lines from stdin. */
    num_lines = read_lines (lines, max_num_lines);
    sort_lines (lines, num_lines);
    print_lines (lines, num_lines);

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


/*
 * read_lines -- reads lines from stdin into an array
 *    inputs: lines -- an (empty) array of strings
 *            max_lines -- the size of the array
 *    outputs: nothing
 *    returns: number of lines read
 */

static int32_t
read_lines (char* lines[], int32_t max_lines)
{
    char    buf[max_line_len + 1];      /* holds current line */
    int32_t num_lines;                  /* number of lines    */

    /* Initialize the line count. */
    num_lines = 0;

    /* Read lines until we find the end of the input. */
    while (NULL != fgets (buf, max_line_len + 1, stdin)) {

        /* Are more lines available than we can read?  If so, print
           a warning message and stop reading. */
        if (max_lines == num_lines) {
            fprintf (stderr, "WARNING: Cannot sort more than %d lines.\n",
                     max_lines);
            break;
        }

        /* Make duplicate copy of line just read in heap memory, then
           store pointer to new copy in lines array. */
        lines[num_lines++] = strdup (buf);
    }

    /* Return number of lines read to caller. */
    return num_lines;
}
```

```
/*
 * sort_lines -- performs an insertion sort on an array of integers
 *    inputs: lines -- an array of strings
 *            n_lines -- the number of lines in the array
 *    outputs: lines -- returned in sorted order
 *    returns: nothing, but changes array in place
 *
 * NOTE: does nothing if n_lines < 2
 *
 */

static void
sort_lines (char* lines[], int32_t n_lines)
{
    int32_t sorted;     /* outer loop index; number of lines sorted       */
    char*   current;    /* current line being placed into sorted subarray */
    int32_t index;      /* inner loop index for placing current line      */

    /* We start with a subarray of length 1 already sorted, so
       we need iterations to sort each larger subarray from length 2
       up to the full length of the array. */
    for (sorted = 2; n_lines >= sorted; sorted++) {

        /* Keep track of the line being moved into position. */
        current = lines[sorted - 1];

        /* Move other array entries aside to make room for "current." */
        for (index = sorted - 1; 0 < index; index--) {

            /* Check the order of "current" against the line before
               that at index.  If it's still smaller, move the line
               and continue the loop.  Otherwise, we've found the place
               to which we must move "current." */
            if (0 > strcmp (current, lines[index - 1]))
                lines[index] = lines[index - 1];
            else
                break;
        }

        /* Store current in the right place. */
        lines[index] = current;
    }

    /* No return value. */
}
```

```
/*
 * print_lines -- print an array of strings (lines)
 *    inputs: lines -- an array of strings
 *            n_lines -- the number of lines in the array
 *    outputs: nothing
 *    returns: nothing, but prints all lines in order to stdout
 */

static void
print_lines (char* const lines[], int32_t n_lines)
{
    int32_t index; /* loop index for printing */

    /* Print all lines in order. */
    for (index = 0; n_lines > index; index++)
        fputs (lines[index], stdout);

    /* No return value. */
}
```