```c
/****************************************************************************
 * scramble.c
 *
 * David J. Malan <malan@harvard.edu>
 * Nate Hardison <nate@cs.harvard.edu>
 *
 * Implements Scramble with CS50.
 *
 * Usage: scramble [#]
 *
 * where # is an optional grid number.
 ***************************************************************************/

#include <cs50.h>
#include <ctype.h>
#include <libgen.h>
#include <stdio.h>
#include <string.h>
#include <time.h>

// duration of a game in seconds
#define DURATION 30

// grid's dimensions
#define DIMENSION 4

// maximum number of words in any dictionary
#define WORDS 172806

// maximum number of letters in any word
#define LETTERS 29

// default dictionary
// http://www.becomeawordgameexpert.com/wordlists.htm
#define DICTIONARY "words"

// for logging
FILE* logfile;

// grid
char grid[DIMENSION][DIMENSION];

// flags with which we can mark grid's letters while searching for words
bool marks[DIMENSION][DIMENSION];

// defines a word as having an array of letters plus a flag
// indicating whether word has been found on grid
typedef struct
```

```c
{
    bool found;
    char letters[LETTERS + 1];
}
word;

// defines a dictionary as having a size and an array of words
struct
{
    int size;
    word words[WORDS];
}
dictionary;

// prototypes
void clear(void);
bool crawl(string letters, int x, int y);
void draw(void);
bool find(string s);
void initialize(void);
bool load(string s);
bool lookup(string s);
void scramble(void);

// This is Scramble.
int main(int argc, string argv[])
{
    // ensure proper usage
    if (argc > 2)
    {
        printf("Usage: %s [#]\n", basename(argv[0]));
        return 1;
    }

    // seed pseudorandom number generator
    if (argc == 2)
    {
        int seed = atoi(argv[1]);
        if (seed <= 0)
        {
            printf("Invalid grid.\n");
            return 1;
        }
        srand(seed);
    }
    else
        srand(time(NULL));

    // determine path to dictionary
    string directory = dirname(argv[0]);
    char path[strlen(directory) + 1 + strlen(DICTIONARY) + 1];
    sprintf(path, "%s/%s", directory, DICTIONARY);

    // load dictionary
    if (!load(path))
    {
        printf("Could not open dictionary.\n");
        return 1;
    }

    // initialize the grid
    initialize();

    // initialize user's score
    int score = 0;

    // calculate time of game's end
    int end = time(NULL) + DURATION;

    // open log
    logfile = fopen("./log.txt", "w");
    if (logfile == NULL)
    {
        printf("Could not open log.\n");
        return 1;
    }

    // accept words until timer expires
    while (true)
    {
        // clear the screen
        clear();

        // draw the current state of the grid
        draw();

        // logfile board
        for (int row = 0; row < DIMENSION; row++)
        {
            for (int col = 0; col < DIMENSION; col++)
                fprintf(logfile, "%c", grid[row][col]);
            fprintf(logfile, "\n");
        }

        // get current time
        int now = time(NULL);

        // report score
        printf("Score: %d\n", score);
        fprintf(logfile, "%d\n", score);

        // check for game's end
        if (now >= end)
        {
            printf("\033[31m"); // red
            printf("Time:  %d\n\n", 0);
            printf("\033[39m"); // default
            break;
        }

        // report time remaining
        printf("Time:  %d\n\n", end - now);

        // prompt for word
        printf("> ");
        string s = GetString();

        // quit playing if user hits ctrl-d
        if (s == NULL)
            break;

        // capitalize word
        for (int i = 0, n = strlen(s); i < n; i++)
            s[i] = toupper(s[i]);

        // logfile word
        fprintf(logfile, "%s\n", s);

        // check whether to scramble grid
        if (strcmp(s, "SCRAMBLE") == 0)
            scramble();

        // or to look for word on grid and in dictionary
        else
        {
            if (find(s) && lookup(s))
                score += strlen(s);
        }
    }

    // close log
    fclose(logfile);

    return 0;
```

```c
}

/**
 * Clears screen.
 */
void clear()
{
    printf("\033[2J");
    printf("\033[%d;%dH", 0, 0);
}

/**
 * Crawls grid recursively for letters starting at grid[x][y].
 * Returns true iff all letters are found.
 */
bool crawl(string letters, int x, int y)
{
    // if out of letters, then we must've found them all!
    if (strlen(letters) == 0)
        return true;

    // don't fall off the grid!
    if (x < 0 || x >= DIMENSION)
        return false;
    if (y < 0 || y >= DIMENSION)
        return false;

    // been here before!
    if (marks[x][y])
        return false;

    // check grid[x][y] for current letter
    if (grid[x][y] != letters[0])
        return false;

    // mark location
    marks[x][y] = true;

    // look left and right for next letter
    for (int i = -1; i <= 1; i++)
    {
        // look down and up for next letter
        for (int j = -1; j <= 1; j++)
        {
            // check grid[x + i][y + j] for next letter
            if (crawl(&letters[1], x + i, y + j))
                return true;
        }
    }

    // unmark location
    marks[x][y] = false;

    // fail
    return false;
}

/**
 * Prints the grid in its current state.
 */
void draw(void)
{
    printf("\n");
    for (int row = 0; row < DIMENSION; row++)
    {
        printf(" ");
        for (int col = 0; col < DIMENSION; col++)
        {
            printf("%2c", grid[row][col]);
        }
        printf("\n");
    }
    printf("\n");
}

/**
 * Returns true iff word, s, is found in grid.
 */
bool find(string s)
{
    // word must be at least 2 characters in length
    if (strlen(s) < 2)
        return false;

    // search grid for word
    for (int row = 0; row < DIMENSION; row++)
    {
        for (int col = 0; col < DIMENSION; col++)
        {
            // reset marks
            for (int i = 0; i < DIMENSION; i++)
                for (int j = 0; j < DIMENSION; j++)
                    marks[i][j] = false;

            // search for word starting at grid[i][j]
            if (crawl(s, row, col))
```

```
                return true;
        }
    }
    return false;
}

/**
 * Initializes grid with letters.
 */
void initialize(void)
{
    // http://en.wikipedia.org/wiki/Letter_frequency
    float frequencies[] = {
        8.167,  // a
        1.492,  // b
        2.782,  // c
        4.253,  // d
        12.702, // e
        2.228,  // f
        2.015,  // g
        6.094,  // h
        6.966,  // i
        0.153,  // j
        0.747,  // k
        4.025,  // l
        2.406,  // m
        6.749,  // n
        7.507,  // o
        1.929,  // p
        0.095,  // q
        5.987,  // r
        6.327,  // s
        9.056,  // t
        2.758,  // u
        1.037,  // v
        2.365,  // w
        0.150,  // x
        1.974,  // y
        0.074   // z
    };
    int n = sizeof(frequencies) / sizeof(float);

    // iterate over grid
    for (int row = 0; row < DIMENSION; row++)
    {
        for (int col = 0; col < DIMENSION; col++)
        {
            // generate pseudorandom double in [0, 1]
```

```c
            double d = rand() / (double) RAND_MAX;

            // map d onto range of frequencies
            for (int k = 0; k < n; k++)
            {
                d -= frequencies[k] / 100;
                if (d < 0.0 || k == n - 1)
                {
                    grid[row][col] = 'A' + k;
                    break;
                }
            }
        }
    }
}

/**
 * Loads words from dictionary with given filename, s, into a global array.
 */
bool load(string s)
{
    // open dictionary
    FILE* file = fopen(s, "r");
    if (file == NULL)
        return false;

    // initialize dictionary's size
    dictionary.size = 0;

    // load words from dictionary
    char buffer[LETTERS + 2];
    while (fgets(buffer, LETTERS + 2, file))
    {
        // overwrite \n with \0
        buffer[strlen(buffer) - 1] = '\0';

        // capitalize word
        for (int i = 0, n = strlen(buffer); i < n; i++)
            buffer[i] = toupper(buffer[i]);

        // ignore SCRAMBLE
        if (strcmp(buffer, "SCRAMBLE") == 0)
            continue;

        // copy word into dictionary
        dictionary.words[dictionary.size].found = false;
        strncpy(dictionary.words[dictionary.size].letters, buffer, LETTERS + 1);
        dictionary.size++;
    }

    // success!
    return true;
}

/**
 * Looks up word, s, in dictionary.  Iff found (for the first time), flags word
 * as found (so that user can't score with it again) and returns true.
 */
bool lookup(string s)
{
    int low = 0;
    int high = dictionary.size - 1;

    while (low <= high)
    {
        // http://googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html
        int mid = ((unsigned int)low + (unsigned int)high) / 2;

        // see man page for strcmp for details on its return values!
        // make sure to test for >/< 0, not ==/!= 1
        int comparison = strcmp(s, dictionary.words[mid].letters);
        if (comparison == 0)
        {
            // check if already found
            if (dictionary.words[mid].found)
                return false;

            // flag as found
            dictionary.words[mid].found = true;

            // found it!
            return true;
        }
        else if (comparison > 0)
            low = mid + 1;
        else
            high = mid - 1;
    }

    // fail
    return false;
}

/**
 * Scrambles the grid by rotating it 90 degrees clockwise, whereby grid[0][0]
 * rotates to grid[0][DIMENSION - 1]
 *
 * Best to instruct students to draw out all of the cases for a 4x4 grid to
 * figure out the math below. Trying to do the rotation in-place is a mess,
 * since moving one cell requires moving three others (e.g. 0,0 -> 0,3 -> 3,0
 * -> 3,3).
 */
void scramble(void)
{
    // build up a new grid with the rotation
    char rotated_grid[DIMENSION][DIMENSION];
    for (int row = 0; row < DIMENSION; row++)
    {
        for (int col = 0; col < DIMENSION; col++)
        {
            rotated_grid[col][DIMENSION - row - 1] = grid[row][col];
        }
    }

    // copy the rotated grid into the global grid
    for (int row = 0; row < DIMENSION; row++)
    {
        for (int col = 0; col < DIMENSION; col++)
        {
            grid[row][col] = rotated_grid[row][col];
        }
    }
}
```