```c
/*****************************************************************************
 * buggy3.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Should swap two variables' values, but doesn't!
 * Can you find the bug?
 *****************************************************************************/

#include <stdio.h>

// function prototype
void swap(int a, int b);

int main(void)
{
    int x = 1;
    int y = 2;

    printf("x is %d\n", x);
    printf("y is %d\n", y);
    printf("Swapping...\n");
    swap(x, y);
    printf("Swapped!\n");
    printf("x is %d\n", x);
    printf("y is %d\n", y);

    return 0;
}

/**
 * Swap arguments' values.
 */
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

```c
/****************************************************************************
 * compare1.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Tries (and fails) to compare two strings.
 *
 * Demonstrates strings as pointers to arrays.
 ****************************************************************************/

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // get line of text
    printf("Say something: ");
    string s = GetString();

    // get another line of text
    printf("Say something: ");
    string t = GetString();

    // try (and fail) to compare strings
    if (s == t)
        printf("You typed the same thing!\n");
    else
        printf("You typed different things!\n");

    return 0;
}
```

```c
/****************************************************************************
 * scramble.c
 *
 * Problem Set 3
 *
 * 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* log;

// 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
{
```

```
    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 word, int x, int y);
void draw(void);
bool find(string word);
void initialize(void);
bool load(string filename);
bool lookup(string word);
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
```

```c
    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
    log = fopen("log.txt", "a");
    if (log == 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();

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

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

```

```c
        // report score
        printf("Score: %d\n", score);
        fprintf(log, "%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 word = GetString();
        if (word != NULL)
        {
            // log word
            fprintf(log, "%s\n", word);

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

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

    // close log
    fclose(log);

    return 0;
}

/**
 * Clears screen.
 */
void clear()
{
```

```c
    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] == true)
        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) == true)
                return true;
        }
    }

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

    // fail
    return false;
```

```c
}

/**
 * Prints the grid in its current state.
 */
void draw(void)
{
    // TODO
}

/**
 * Returns true iff word is found in grid.
 */
bool find(string word)
{
    // word must be at least 2 characters in length
    if (strlen(word) < 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(word, row, col) == true)
                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]
            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;
                }
            }
        }
    }
}

/**
```

```c
 * Loads words from dictionary with given filename into a global array.
 */
bool load(string filename)
{
    // open dictionary
    FILE* file = fopen(filename, "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 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 word)
{
    // TODO
    return false;
}

/**
```

```c
 * Scrambles the grid by rotating it 90 degrees clockwise, whereby
 * grid[0][0] rotates to grid[0][DIMENSION - 1].
 */
void scramble(void)
{
    // TODO
}
```

```c
/*****************************************************************************
 * sigma1.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Adds the numbers 1 through n.
 *
 * Demonstrates iteration.
 *****************************************************************************/

#include <cs50.h>
#include <stdio.h>

// prototype
int sigma(int);

int main(void)
{
    // ask user for a positive int
    int n;
    do
    {
        printf("Positive integer please: ");
        n = GetInt();
    }
    while (n < 1);

    // compute sum of 1 through n
    int answer = sigma(n);

    // report answer
    printf("%d\n", answer);

    return 0;
}

/**
 * Returns sum of 1 through m; returns 0 if m is not positive.
 */
int sigma(int m)
{
    // avoid risk of infinite loop
    if (m < 1)
        return 0;

    // return sum of 1 through m
    int sum = 0;
```

```
    for (int i = 1; i <= m; i++)
        sum += i;
    return sum;
}
```

```c
/*****************************************************************************
 * sigma2.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Adds the numbers 1 through n.
 *
 * Demonstrates recursion.
 *****************************************************************************/

#include <cs50.h>
#include <stdio.h>

// prototype
int sigma(int);

int main(void)
{
    // ask user for a positive int
    int n;
    do
    {
        printf("Positive integer please: ");
        n = GetInt();
    }
    while (n < 1);

    // compute sum of 1 through n
    int answer = sigma(n);

    // report answer
    printf("%d\n", answer);

    return 0;
}

/**
 * Returns sum of 1 through m; returns 0 if m is not positive.
 */
int sigma(int m)
{
    // base case
    if (m <= 0)
        return 0;

    // recursive case
    else
```

```
        return (m + sigma(m-1));
}
```

```
/*****************************************************************************
 * structs.h
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Defines a student for structs{1,2}.c.
 *****************************************************************************/

#include <cs50.h>

// structure representing a student
typedef struct
{
    int id;
    string name;
    string house;
}
student;
```

```c
/*****************************************************************************
 * structs1.c
 *
 * Computer Science 50
 * David J. Malan
 *
 * Demonstrates use of structs.
 *****************************************************************************/

#include <cs50.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "structs.h"

// class size
#define STUDENTS 3

int main(void)
{
    // declare class
    student class[STUDENTS];

    // populate class with user's input
    for (int i = 0; i < STUDENTS; i++)
    {
        printf("Student's ID: ");
        class[i].id = GetInt();

        printf("Student's name: ");
        class[i].name = GetString();

        printf("Student's house: ");
        class[i].house = GetString();
        printf("\n");
    }

    // now print anyone in Mather
    for (int i = 0; i < STUDENTS; i++)
        if (strcmp(class[i].house, "Mather") == 0)
            printf("%s is in Mather!\n\n", class[i].name);

    // free memory
    for (int i = 0; i < STUDENTS; i++)
    {
        free(class[i].name);
        free(class[i].house);
```

```
    }

    return 0;
}
```

```c
/*****************************************************************************
 * structs.c
 *
 * David J. Malan
 * malan@harvard.edu
 *
 * Demonstrates use of structs.
 *****************************************************************************/

#include <cs50.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "structs.h"

// class size
#define STUDENTS 3

int main(void)
{
    // declare class
    student class[STUDENTS];

    // populate class with user's input
    for (int i = 0; i < STUDENTS; i++)
    {
        printf("Student's ID: ");
        class[i].id = GetInt();

        printf("Student's name: ");
        class[i].name = GetString();

        printf("Student's house: ");
        class[i].house = GetString();
        printf("\n");
    }

    // now print anyone in Mather
    for (int i = 0; i < STUDENTS; i++)
        if (strcmp(class[i].house, "Mather") == 0)
            printf("%s is in Mather!\n\n", class[i].name);

    // let's save these students to disk
    FILE* fp = fopen("database", "w");
    if (fp != NULL)
    {
        for (int i = 0; i < STUDENTS; i++)
```

```
        {
            fprintf(fp, "%d\n", class[i].id);
            fprintf(fp, "%s\n", class[i].name);
            fprintf(fp, "%s\n", class[i].house);
        }
        fclose(fp);
    }

    // free memory
    for (int i = 0; i < STUDENTS; i++)
    {
        free(class[i].name);
        free(class[i].house);
    }

    return 0;
}
```