

```
1.  /*****
2.   * buggy3.c
3.   *
4.   * David J. Malan
5.   * malan@harvard.edu
6.   *
7.   * Should swap two variables' values, but doesn't!
8.   * Can you find the bug?
9.   *****/
10.
11. #include <stdio.h>
12.
13. // function prototype
14. void swap(int a, int b);
15.
16. int main(void)
17. {
18.     int x = 1;
19.     int y = 2;
20.
21.     printf("x is %d\n", x);
22.     printf("y is %d\n", y);
23.     printf("Swapping...\n");
24.     swap(x, y);
25.     printf("Swapped!\n");
26.     printf("x is %d\n", x);
27.     printf("y is %d\n", y);
28.
29.     return 0;
30. }
31.
32. /**
33.  * Swap arguments' values.
34.  */
35. void swap(int a, int b)
36. {
37.     int tmp = a;
38.     a = b;
39.     b = tmp;
40. }
```

```
1.  /*****
2.   * compare1.c
3.   *
4.   * David J. Malan
5.   * malan@harvard.edu
6.   *
7.   * Tries (and fails) to compare two strings.
8.   *
9.   * Demonstrates strings as pointers to arrays.
10.  *****/
11.
12.  #include <cs50.h>
13.  #include <stdio.h>
14.
15.  int main(void)
16.  {
17.      // get line of text
18.      printf("Say something: ");
19.      string s = GetString();
20.
21.      // get another line of text
22.      printf("Say something: ");
23.      string t = GetString();
24.
25.      // try (and fail) to compare strings
26.      if (s == t)
27.          printf("You typed the same thing!\n");
28.      else
29.          printf("You typed different things!\n");
30.
31.      return 0;
32.  }
```

```

1.  /*****
2.   *  scramble.c
3.   *
4.   *  Problem Set 3
5.   *
6.   *  Implements Scramble with CS50.
7.   *
8.   *  Usage: scramble [#]
9.   *
10.  *  where # is an optional grid number.
11.  *****/
12.
13.  #include <cs50.h>
14.  #include <ctype.h>
15.  #include <libgen.h>
16.  #include <stdio.h>
17.  #include <string.h>
18.  #include <time.h>
19.
20.  // duration of a game in seconds
21.  #define DURATION 30
22.
23.  // grid's dimensions
24.  #define DIMENSION 4
25.
26.  // maximum number of words in any dictionary
27.  #define WORDS 172806
28.
29.  // maximum number of letters in any word
30.  #define LETTERS 29
31.
32.  // default dictionary
33.  // http://www.becomeawordgameexpert.com/wordlists.htm
34.  #define DICTIONARY "words"
35.
36.  // for logging
37.  FILE* log;
38.
39.  // grid
40.  char grid[DIMENSION][DIMENSION];
41.
42.  // flags with which we can mark grid's letters while searching for words
43.  bool marks[DIMENSION][DIMENSION];
44.
45.  // defines a word as having an array of letters plus a flag
46.  // indicating whether word has been found on grid
47.  typedef struct
48.  {

```

```
49.     bool found;
50.     char letters[LETTERS + 1];
51. }
52. word;
53.
54. // defines a dictionary as having a size and an array of words
55. struct
56. {
57.     int size;
58.     word words[WORDS];
59. }
60. dictionary;
61.
62. // prototypes
63. void clear(void);
64. bool crawl(string word, int x, int y);
65. void draw(void);
66. bool find(string word);
67. void initialize(void);
68. bool load(string filename);
69. bool lookup(string word);
70. void scramble(void);
71.
72. // This is Scramble.
73. int main(int argc, string argv[])
74. {
75.     // ensure proper usage
76.     if (argc > 2)
77.     {
78.         printf("Usage: %s [#]\n", basename(argv[0]));
79.         return 1;
80.     }
81.
82.     // seed pseudorandom number generator
83.     if (argc == 2)
84.     {
85.         int seed = atoi(argv[1]);
86.         if (seed <= 0)
87.         {
88.             printf("Invalid grid.\n");
89.             return 1;
90.         }
91.         srand(seed);
92.     }
93.     else
94.         srand(time(NULL));
95.
96.     // determine path to dictionary
```

```
97.     string directory = dirname(argv[0]);
98.     char path[strlen(directory) + 1 + strlen(DICTIONARY) + 1];
99.     sprintf(path, "%s/%s", directory, DICTIONARY);
100.
101.     // load dictionary
102.     if (!load(path))
103.     {
104.         printf("Could not open dictionary.\n");
105.         return 1;
106.     }
107.
108.     // initialize the grid
109.     initialize();
110.
111.     // initialize user's score
112.     int score = 0;
113.
114.     // calculate time of game's end
115.     int end = time(NULL) + DURATION;
116.
117.     // open log
118.     log = fopen("log.txt", "a");
119.     if (log == NULL)
120.     {
121.         printf("Could not open log.\n");
122.         return 1;
123.     }
124.
125.     // accept words until timer expires
126.     while (true)
127.     {
128.         // clear the screen
129.         clear();
130.
131.         // draw the current state of the grid
132.         draw();
133.
134.         // log board
135.         for (int row = 0; row < DIMENSION; row++)
136.         {
137.             for (int col = 0; col < DIMENSION; col++)
138.                 fprintf(log, "%c", grid[row][col]);
139.             fprintf(log, "\n");
140.         }
141.
142.         // get current time
143.         int now = time(NULL);
144.
```

```
145.     // report score
146.     printf("Score: %d\n", score);
147.     fprintf(log, "%d\n", score);
148.
149.     // check for game's end
150.     if (now >= end)
151.     {
152.         printf("\033[31m"); // red
153.         printf("Time:  %d\n\n", 0);
154.         printf("\033[39m"); // default
155.         break;
156.     }
157.
158.     // report time remaining
159.     printf("Time:  %d\n\n", end - now);
160.
161.     // prompt for word
162.     printf("> ");
163.     string word = GetString();
164.     if (word != NULL)
165.     {
166.         // log word
167.         fprintf(log, "%s\n", word);
168.
169.         // check whether to scramble grid
170.         if (strcmp(word, "SCRAMBLE") == 0)
171.             scramble();
172.
173.         // or to look for word on grid and in dictionary
174.         else
175.         {
176.             if (find(word) == true && lookup(word) == true)
177.                 score += strlen(word);
178.         }
179.     }
180. }
181.
182. // close log
183. fclose(log);
184.
185. return 0;
186. }
187.
188. /**
189.  * Clears screen.
190.  */
191. void clear()
192. {
```

```
193.     printf("\033[2J");
194.     printf("\033[%d;%dH", 0, 0);
195. }
196.
197. /**
198.  * Crawls grid recursively for letters starting at grid[x][y].
199.  * Returns true iff all letters are found.
200.  */
201. bool crawl(string letters, int x, int y)
202. {
203.     // if out of letters, then we must've found them all!
204.     if (strlen(letters) == 0)
205.         return true;
206.
207.     // don't fall off the grid!
208.     if (x < 0 || x >= DIMENSION)
209.         return false;
210.     if (y < 0 || y >= DIMENSION)
211.         return false;
212.
213.     // been here before!
214.     if (marks[x][y] == true)
215.         return false;
216.
217.     // check grid[x][y] for current letter
218.     if (grid[x][y] != letters[0])
219.         return false;
220.
221.     // mark location
222.     marks[x][y] = true;
223.
224.     // look left and right for next letter
225.     for (int i = -1; i <= 1; i++)
226.     {
227.         // look down and up for next letter
228.         for (int j = -1; j <= 1; j++)
229.         {
230.             // check grid[x + i][y + j] for next letter
231.             if (crawl(&letters[1], x + i, y + j) == true)
232.                 return true;
233.         }
234.     }
235.
236.     // unmark location
237.     marks[x][y] = false;
238.
239.     // fail
240.     return false;
```

```
241. }
242.
243. /**
244.  * Prints the grid in its current state.
245.  */
246. void draw(void)
247. {
248.     // TODO
249. }
250.
251. /**
252.  * Returns true iff word is found in grid.
253.  */
254. bool find(string word)
255. {
256.     // word must be at least 2 characters in length
257.     if (strlen(word) < 2)
258.         return false;
259.
260.     // search grid for word
261.     for (int row = 0; row < DIMENSION; row++)
262.     {
263.         for (int col = 0; col < DIMENSION; col++)
264.         {
265.             // reset marks
266.             for (int i = 0; i < DIMENSION; i++)
267.                 for (int j = 0; j < DIMENSION; j++)
268.                     marks[i][j] = false;
269.
270.             // search for word starting at grid[i][j]
271.             if (crawl(word, row, col) == true)
272.                 return true;
273.         }
274.     }
275.     return false;
276. }
277.
278. /**
279.  * Initializes grid with letters.
280.  */
281. void initialize(void)
282. {
283.     // http://en.wikipedia.org/wiki/Letter_frequency
284.     float frequencies[] = {
285.         8.167, // a
286.         1.492, // b
287.         2.782, // c
288.         4.253, // d
```



```
289.     12.702, // e
290.     2.228, // f
291.     2.015, // g
292.     6.094, // h
293.     6.966, // i
294.     0.153, // j
295.     0.747, // k
296.     4.025, // l
297.     2.406, // m
298.     6.749, // n
299.     7.507, // o
300.     1.929, // p
301.     0.095, // q
302.     5.987, // r
303.     6.327, // s
304.     9.056, // t
305.     2.758, // u
306.     1.037, // v
307.     2.365, // w
308.     0.150, // x
309.     1.974, // y
310.     0.074  // z
311. };
312. int n = sizeof(frequencies) / sizeof(float);
313.
314. // iterate over grid
315. for (int row = 0; row < DIMENSION; row++)
316. {
317.     for (int col = 0; col < DIMENSION; col++)
318.     {
319.         // generate pseudorandom double in [0, 1]
320.         double d = rand() / (double) RAND_MAX;
321.
322.         // map d onto range of frequencies
323.         for (int k = 0; k < n; k++)
324.         {
325.             d -= frequencies[k] / 100;
326.             if (d < 0.0 || k == n - 1)
327.             {
328.                 grid[row][col] = 'A' + k;
329.                 break;
330.             }
331.         }
332.     }
333. }
334. }
335.
336. /**
```

```
337.  * Loads words from dictionary with given filename into a global array.
338.  */
339.  bool load(string filename)
340.  {
341.      // open dictionary
342.      FILE* file = fopen(filename, "r");
343.      if (file == NULL)
344.          return false;
345.
346.      // initialize dictionary's size
347.      dictionary.size = 0;
348.
349.      // load words from dictionary
350.      char buffer[LETTERS + 2];
351.      while (fgets(buffer, LETTERS + 2, file))
352.      {
353.          // overwrite \n with \0
354.          buffer[strlen(buffer) - 1] = '\0';
355.
356.          // capitalize word
357.          for (int i = 0, n = strlen(buffer); i < n; i++)
358.              buffer[i] = toupper(buffer[i]);
359.
360.          // ignore SCRAMBLE
361.          if (strcmp(buffer, "SCRAMBLE") == 0)
362.              continue;
363.
364.          // copy word into dictionary
365.          dictionary.words[dictionary.size].found = false;
366.          strncpy(dictionary.words[dictionary.size].letters, buffer, LETTERS + 1);
367.          dictionary.size++;
368.      }
369.
370.      // success!
371.      return true;
372.  }
373.
374.  /**
375.   * Looks up word in dictionary.  If found (for the first time), flags word
376.   * as found (so that user can't score with it again) and returns true.
377.   */
378.  bool lookup(string word)
379.  {
380.      // TODO
381.      return false;
382.  }
383.
384.  /**
```

```
385.  * Scrambles the grid by rotating it 90 degrees clockwise, whereby
386.  * grid[0][0] rotates to grid[0][DIMENSION - 1].
387.  */
388. void scramble(void)
389. {
390.     // TODO
391. }
```

```
1.  /*****
2.   * sigma1.c
3.   *
4.   * David J. Malan
5.   * malan@harvard.edu
6.   *
7.   * Adds the numbers 1 through n.
8.   *
9.   * Demonstrates iteration.
10.  *****/
11.
12. #include <cs50.h>
13. #include <stdio.h>
14.
15. // prototype
16. int sigma(int);
17.
18. int main(void)
19. {
20.     // ask user for a positive int
21.     int n;
22.     do
23.     {
24.         printf("Positive integer please: ");
25.         n = GetInt();
26.     }
27.     while (n < 1);
28.
29.     // compute sum of 1 through n
30.     int answer = sigma(n);
31.
32.     // report answer
33.     printf("%d\n", answer);
34.
35.     return 0;
36. }
37.
38. /**
39.  * Returns sum of 1 through m; returns 0 if m is not positive.
40.  */
41. int sigma(int m)
42. {
43.     // avoid risk of infinite loop
44.     if (m < 1)
45.         return 0;
46.
47.     // return sum of 1 through m
48.     int sum = 0;
```

```
49.     for (int i = 1; i <= m; i++)
50.         sum += i;
51.     return sum;
52. }
```

```
1.  /*****
2.   * sigma2.c
3.   *
4.   * David J. Malan
5.   * malan@harvard.edu
6.   *
7.   * Adds the numbers 1 through n.
8.   *
9.   * Demonstrates recursion.
10.  *****/
11.
12. #include <cs50.h>
13. #include <stdio.h>
14.
15. // prototype
16. int sigma(int);
17.
18. int main(void)
19. {
20.     // ask user for a positive int
21.     int n;
22.     do
23.     {
24.         printf("Positive integer please: ");
25.         n = GetInt();
26.     }
27.     while (n < 1);
28.
29.     // compute sum of 1 through n
30.     int answer = sigma(n);
31.
32.     // report answer
33.     printf("%d\n", answer);
34.
35.     return 0;
36. }
37.
38. /**
39.  * Returns sum of 1 through m; returns 0 if m is not positive.
40.  */
41. int sigma(int m)
42. {
43.     // base case
44.     if (m <= 0)
45.         return 0;
46.
47.     // recursive case
48.     else
```

```
49.     return (m + sigma(m-1));  
50. }
```

```
1.  /* *****
2.   * structs.h
3.   *
4.   * David J. Malan
5.   * malan@harvard.edu
6.   *
7.   * Defines a student for structs{1,2}.c.
8.  *****/
9.
10. #include <cs50.h>
11.
12. // structure representing a student
13. typedef struct
14. {
15.     int id;
16.     string name;
17.     string house;
18. }
19. student;
```



```
1.  /*****
2.   * structs1.c
3.   *
4.   * Computer Science 50
5.   * David J. Malan
6.   *
7.   * Demonstrates use of structs.
8.   *****/
9.
10. #include <cs50.h>
11. #include <stdio.h>
12. #include <stdlib.h>
13. #include <string.h>
14.
15. #include "structs.h"
16.
17. // class size
18. #define STUDENTS 3
19.
20. int main(void)
21. {
22.     // declare class
23.     student class[STUDENTS];
24.
25.     // populate class with user's input
26.     for (int i = 0; i < STUDENTS; i++)
27.     {
28.         printf("Student's ID: ");
29.         class[i].id = GetInt();
30.
31.         printf("Student's name: ");
32.         class[i].name = GetString();
33.
34.         printf("Student's house: ");
35.         class[i].house = GetString();
36.         printf("\n");
37.     }
38.
39.     // now print anyone in Mather
40.     for (int i = 0; i < STUDENTS; i++)
41.         if (strcmp(class[i].house, "Mather") == 0)
42.             printf("%s is in Mather!\n\n", class[i].name);
43.
44.     // free memory
45.     for (int i = 0; i < STUDENTS; i++)
46.     {
47.         free(class[i].name);
48.         free(class[i].house);
49.     }
```

```
49.     }  
50.  
51.     return 0;  
52. }
```

```
1.  /*****
2.   * structs.c
3.   *
4.   * David J. Malan
5.   * malan@harvard.edu
6.   *
7.   * Demonstrates use of structs.
8.   *****/
9.
10. #include <cs50.h>
11. #include <stdio.h>
12. #include <stdlib.h>
13. #include <string.h>
14.
15. #include "structs.h"
16.
17. // class size
18. #define STUDENTS 3
19.
20. int main(void)
21. {
22.     // declare class
23.     student class[STUDENTS];
24.
25.     // populate class with user's input
26.     for (int i = 0; i < STUDENTS; i++)
27.     {
28.         printf("Student's ID: ");
29.         class[i].id = GetInt();
30.
31.         printf("Student's name: ");
32.         class[i].name = GetString();
33.
34.         printf("Student's house: ");
35.         class[i].house = GetString();
36.         printf("\n");
37.     }
38.
39.     // now print anyone in Mather
40.     for (int i = 0; i < STUDENTS; i++)
41.         if (strcmp(class[i].house, "Mather") == 0)
42.             printf("%s is in Mather!\n\n", class[i].name);
43.
44.     // let's save these students to disk
45.     FILE* fp = fopen("database", "w");
46.     if (fp != NULL)
47.     {
48.         for (int i = 0; i < STUDENTS; i++)
```

```
49.     {
50.         fprintf(fp, "%d\n", class[i].id);
51.         fprintf(fp, "%s\n", class[i].name);
52.         fprintf(fp, "%s\n", class[i].house);
53.     }
54.     fclose(fp);
55. }
56.
57. // free memory
58. for (int i = 0; i < STUDENTS; i++)
59. {
60.     free(class[i].name);
61.     free(class[i].house);
62. }
63.
64. return 0;
65. }
```