

```
1.  /**
2.   * fifteen.c
3.   *
4.   * CS50 AP
5.   * Name: Gonzalo de la Torre Amaya
6.   *
7.   * Implements Game of Fifteen (generalized to d x d).
8.   *
9.   * Usage: fifteen d
10.  *
11.  * whereby the board's dimensions are to be d x d,
12.  * where d must be in [DIM_MIN,DIM_MAX]
13.  *
14.  * Note that usleep is obsolete, but it offers more granularity than
15.  * sleep and is simpler to use than nanosleep; `man usleep` for more.
16.  */
17.
18. // necessary for usleep
19. #define _XOPEN_SOURCE 500
20.
21. // libraries to include
22. #include <cs50.h>
23. #include <stdio.h>
24. #include <stdlib.h>
25. #include <unistd.h>
26.
27. // constants
28. #define DIM_MIN 3
29. #define DIM_MAX 9
30. #define BLANK 2424
31.
32. // globally declared board
33. int board[DIM_MAX][DIM_MAX];
34.
35. // globally declared board dimension
36. int d;
37.
38. // prototypes
39. void clear(void);
40. void greet(void);
41. void init(void);
42. void draw(void);
43. bool move(int tile);
44. bool won(void);
45.
46. int main(int argc, string argv[])
47. {
48.     // TODO 00: Incorrect usage
```

```
49.     if (argc != 2)
50.     {
51.         printf("Usage: fifteen d\n");
52.         return 1;
53.     }
54.
55.     // TODO 01: Be sure that the user puts a dimension 3x3 through 9x9
56.     d = atoi(argv[1]);
57.     if (d < DIM_MIN || d > DIM_MAX)
58.     {
59.         printf("Board must be between %i x %i and %i x %i, inclusive.\n",
60.             DIM_MIN, DIM_MIN, DIM_MAX, DIM_MAX);
61.         return 2;
62.     }
63.
64.     // open log file to record moves
65.     FILE* file = fopen("log.txt", "w");
66.     if (file == NULL)
67.     {
68.         return 3;
69.     }
70.
71.     // TODO 02: For the function "greet"
72.     greet();
73.
74.     // TODO 03: For the function "init"
75.     init();
76.
77.     // accept moves until game is won
78.     while (true)
79.     {
80.         // TODO 04: For the function "clear"
81.         clear();
82.
83.         // TODO 05: For the function "draw"
84.         draw();
85.
86.         // log the current state of the board (for testing)
87.         for (int i = 0; i < d; i++)
88.         {
89.             for (int j = 0; j < d; j++)
90.             {
91.                 fprintf(file, "%i", board[i][j]);
92.                 if (j < d - 1)
93.                 {
94.                     fprintf(file, "|");
95.                 }
96.             }
```

```
97.         fprintf(file, "\n");
98.     }
99.     fflush(file);
100.
101.     // TODO 06: Congratulate if the user win !
102.     if (won())
103.     {
104.         printf("ftw!\n");
105.         break;
106.     }
107.
108.     // TODO 07: Ask the user for the tile with a GetInt
109.     printf("Tile to move: ");
110.     int tile = GetInt();
111.
112.     // quit if user inputs 0 (for testing)
113.     if (tile == 0)
114.     {
115.         break;
116.     }
117.
118.     // log move (for testing)
119.     fprintf(file, "%i\n", tile);
120.     fflush(file);
121.
122.     // TODO 08: If the user does a "illegal move": scold !
123.     if (!move(tile))
124.     {
125.         printf("\nIllegal move.\n");
126.         usleep(500000);
127.     }
128.
129.     // TODO 09: Delay the program 500000 microseconds
130.     usleep(500000);
131. }
132.
133. // close log
134. fclose(file);
135.
136. // TODO 10: To end the game we need to return 0;
137. return 0;
138. }
139.
140. /**
141.  * Clears screen using ANSI escape sequences.
142.  */
143. void clear(void)
144. {
```

```
145.     printf("\033[2J");
146.     printf("\033[%d;%dH", 0, 0);
147. }
148.
149. /**
150.  * Greets player.
151.  */
152. void greet(void)
153. {
154.     clear();
155.     printf("WELCOME TO GAME OF FIFTEEN\n");
156.     usleep(2000000);
157. }
158.
159. /**
160.  * Initializes the game's board with tiles numbered 1 through d*d - 1
161.  * (i.e., fills 2D array with values but does not actually print them).
162.  */
163. void init(void)
164. {
165.     // Declare the Last number
166.     int LastNum = d * d - 1;
167.
168.     for (int i = 0; i < d; i++)
169.     {
170.
171.         for(int j = 0; j < d; j++)
172.         {
173.             board[i][j] = LastNum;
174.             LastNum--;
175.         }
176.
177.     }
178.
179.     // Check if the nummber is odd, it must be changed the position of 1 and 2
180.     if ((d * d - 1) % 2 != 0)
181.     {
182.         board[d - 1][d - 2] = 2;
183.         board[d - 1][d - 3] = 1;
184.     }
185. }
186.
187. /**
188.  * Prints the board in its current state.
189.  */
190. void draw(void)
191. {
192.     // TODO
```

```
193.     // create board with dimensions given by user
194.
195.     // For the rows
196.     for (int i = 0; i < d; i++)
197.     {
198.         // To order the numbers in the rows correctly
199.         for (int j = 0; j < d; j++)
200.         {
201.             if (board[i][j] == 0)
202.             {
203.                 printf("_ ");
204.             }
205.             else if (board[i][j] < 10)
206.             {
207.                 printf("%d ", board[i][j]);
208.             }
209.             else
210.             {
211.                 printf("%d ", board[i][j]);
212.             }
213.         }
214.         printf("\n");
215.     }
216. }
217.
218. /**
219.  * If tile borders empty space, moves tile and returns true, else
220.  * returns false.
221.  */
222. bool move(int tile)
223. {
224.     // TODO
225.     for (int i = 0; i < d; i++)
226.     {
227.         for (int j = 0; j < d; j++)
228.         {
229.             if (board[i][j] == tile)
230.             {
231.                 if ((i + 1) < d && board[i + 1][j] == 0)
232.                 {
233.                     board[i + 1][j] = tile;
234.                     board[i][j] = 0;
235.                     return true;
236.                 }
237.                 else if ((i - 1) >= 0 && board[i - 1][j] == 0)
238.                 {
239.                     board[i - 1][j] = tile;
240.                     board[i][j] = 0;
```

```
241.         return true;
242.     }
243.     else if ((j + 1) < d && board[i][j + 1] == 0)
244.     {
245.         board[i][j + 1] = tile;
246.         board[i][j] = 0;
247.         return true;
248.     }
249.     else if ((j - 1) >= 0 && board[i][j - 1] == 0)
250.     {
251.         board[i][j - 1] = tile;
252.         board[i][j] = 0;
253.         return true;
254.     }
255. }
256. }
257. }
258. return false;
259. }
260.
261. /**
262.  * Returns true if game is won (i.e., board is in winning configuration),
263.  * else false.
264.  */
265. bool won(void)
266. {
267.     int n = 1;
268.     for (int i = 0; i < d; i++)
269.     {
270.         for (int j = 0; j < d; j++)
271.         {
272.             if (board[i][j] == n)
273.             {
274.                 n++;
275.                 if (n == d * d && board[d - 1][d - 1] == 0)
276.                 {
277.                     return true;
278.                 }
279.             }
280.         }
281.     }
282.     return false;
283. }
284. }
```