

```
1  #include <cs50.h>
2  #include <stdio.h>
3
4  int main(void)
5  {
6      int height;
7      do
8      {
9          height = get_int("Height: ");
10     }
11     while (height < 1 || height > 8);
12
13     for (int i = 0; i < height; i++)
14     {
15         for (int j = 0; j < height - i - 1; j++)
16         {
17             printf(" ");
18         }
19         for (int k = 0; k <= i; k++)
20         {
21             printf("#");
22         }
23         printf(" ");
24         for (int k = 0; k <= i; k++)
25         {
26             printf("#");
27         }
28         printf("\n");
29     }
30 }
```

```
1  #include <cs50.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int main(int argc, string argv[])
6  {
7      if (argc != 4)
8      {
9          printf("Usage: ./calc <num> <op> <num>\n");
10         return 1;
11     }
12
13     float num1 = (float) strtod(argv[1], NULL);
14     float num2 = (float) strtod(argv[3], NULL);
15     char op = argv[2][0];
16
17     int quot = (int) (num1 / num2);
18     float rem = num1 - (num2 * quot);
19
20     switch (op)
21     {
22         case '+':
23             printf("%f\n", num1 + num2);
24             break;
25         case '-':
26             printf("%f\n", num1 - num2);
27             break;
28         case 'x':
29             printf("%f\n", num1 * num2);
30             break;
31         case '/':
32             printf("%f\n", num1 / num2);
33             break;
34         case '%':
35             printf("%f\n", rem);
36             break;
37         default:
38             printf("Operator must be +, -, x, /, or %%\n");
39             return 1;
40     }
41 }
```

```
1  #include <cs50.h>
2  #include <ctype.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6
7  int main(int argc, string argv[])
8  {
9      if (argc != 2)
10     {
11         printf("Usage: ./caesar <key>\n");
12         return 1;
13     }
14
15     int key = (int) strtol(argv[1], NULL, 10);
16
17     string plaintext = get_string("plaintext: ");
18     printf("ciphertext: ");
19
20     for (int i = 0; i < strlen(plaintext); i++)
21     {
22         if (isalpha(plaintext[i]))
23         {
24             if (isupper(plaintext[i]))
25             {
26                 printf("%c", ((plaintext[i] - 'A' + key) % 26) + 'A');
27             }
28             else
29             {
30                 printf("%c", ((plaintext[i] - 'a' + key) % 26) + 'a');
31             }
32         }
33         else
34         {
35             printf("%c", plaintext[i]);
36         }
37     }
38
39     printf("\n");
40 }
```

```
1  // Implements Game of Fifteen (generalized to d x d)
2
3  #define _XOPEN_SOURCE 500
4
5  #include <cs50.h>
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <unistd.h>
9
10 // Constants
11 #define DIM_MIN 3
12 #define DIM_MAX 9
13 #define COLOR "\033[32m"
14
15 // Board
16 int board[DIM_MAX][DIM_MAX];
17
18 // Dimensions
19 int d;
20
21 // Saved locations of the blank tile
22 int blank_row;
23 int blank_col;
24
25 // Prototypes
26 void clear(void);
27 void greet(void);
28 void init(void);
29 void draw(void);
30 bool move(int tile);
31 bool won(void);
32 void swap(int *a, int *b);
33 void print_grid_row(int d);
34 void print_tile(int tile);
35
36 int main(int argc, string argv[])
37 {
38     // Ensure proper usage
39     if (argc != 2)
40     {
41         printf("Usage: fifteen d\n");
42         return 1;
43     }
44
45     // Ensure valid dimensions
```

```
46     d = atoi(argv[1]);
47     if (d < DIM_MIN || d > DIM_MAX)
48     {
49         printf("Board must be between %i x %i and %i x %i, inclusive.\n",
50             DIM_MIN, DIM_MIN, DIM_MAX, DIM_MAX);
51         return 2;
52     }
53
54     // Open log
55     FILE *file = fopen("log.txt", "w");
56     if (file == NULL)
57     {
58         return 3;
59     }
60
61     // Greet user with instructions
62     greet();
63
64     // Initialize the board
65     init();
66
67     // Accept moves until game is won
68     while (true)
69     {
70         // Clear the screen
71         clear();
72
73         // Draw the current state of the board
74         draw();
75
76         // Log the current state of the board (for testing)
77         for (int i = 0; i < d; i++)
78         {
79             for (int j = 0; j < d; j++)
80             {
81                 fprintf(file, "%i", board[i][j]);
82                 if (j < d - 1)
83                 {
84                     fprintf(file, "|");
85                 }
86             }
87             fprintf(file, "\n");
88         }
89         fflush(file);
90     }
```

```
91         // Check for win
92         if (won())
93         {
94             printf("ftw!\n");
95             break;
96         }
97
98         // Prompt for move
99         int tile = get_int("Tile to move: ");
100
101         // Quit if user inputs 0 (for testing)
102         if (tile == 0)
103         {
104             break;
105         }
106
107         // Log move (for testing)
108         fprintf(file, "%i\n", tile);
109         fflush(file);
110
111         // Move if possible, else report illegality
112         if (!move(tile))
113         {
114             printf("\nIllegal move.\n");
115             usleep(500000);
116         }
117
118         // Sleep thread for animation's sake
119         usleep(50000);
120     }
121
122     // Close log
123     fclose(file);
124
125     // Success
126     return 0;
127 }
128
129 // Clears screen using ANSI escape sequences
130 void clear(void)
131 {
132     printf("\033[2J");
133     printf("\033[%d;%dH", 0, 0);
134 }
135
```

```
136 // Greets player
137 void greet(void)
138 {
139     clear();
140     printf("WELCOME TO GAME OF FIFTEEN\n");
141     usleep(2000000);
142 }
143
144 // Initializes the game's board with tiles numbered 1 through d*d - 1
145 // (i.e., fills 2D array with values but does not actually print them)
146 void init(void)
147 {
148     int tile = d * d - 1;
149
150     for (int i = 0; i < d; i++)
151     {
152         for (int j = 0; j < d; j++)
153         {
154             board[i][j] = tile;
155             tile--;
156         }
157     }
158
159     if (d % 2 == 0)
160     {
161         int temp = board[d-1][d-2];
162         board[d-1][d-2] = board[d-1][d-3];
163         board[d-1][d-3] = temp;
164     }
165 }
166
167 // Prints the board in its current state
168 void draw(void)
169 {
170     for (int i = 0; i < d; i++)
171     {
172         for (int j = 0; j < d; j++)
173         {
174             if (board[i][j] == 0)
175             {
176                 printf("_\t");
177             }
178             else
179             {
180                 printf("%d\t", board[i][j]);
```

```
181     }
182     }
183     printf("\n");
184 }
185 }
186
187 // If tile borders empty space, moves tile and returns true, else returns false
188 bool move(int tile)
189 {
190     for (int i = 0; i < d; i++)
191     {
192         for (int j = 0; j < d; j++)
193         {
194             if (board[i][j] == tile)
195             {
196                 // check up
197                 if (i-1 >= 0 && board[i-1][j] == 0)
198                 {
199                     board[i-1][j] = tile;
200                     board[i][j] = 0;
201                     return true;
202                 }
203                 // check down
204                 else if (i+1 < d && board[i+1][j] == 0)
205                 {
206                     board[i+1][j] = tile;
207                     board[i][j] = 0;
208                     return true;
209                 }
210                 // check left
211                 else if (j-1 >= 0 && board[i][j-1] == 0)
212                 {
213                     board[i][j-1] = tile;
214                     board[i][j] = 0;
215                     return true;
216                 }
217                 // check right
218                 else if (j+1 < d && board[i][j+1] == 0)
219                 {
220                     board[i][j+1] = tile;
221                     board[i][j] = 0;
222                     return true;
223                 }
224             }
225         }
226     }
227 }
```



```
226     }
227     return false;
228 }
229
230 // Returns true if game is won (i.e., board is in winning configuration), else false
231 bool won(void)
232 {
233     int correct = 1;
234     for (int i = 0; i < d; i++)
235     {
236         for (int j = 0; j < d; j++)
237         {
238             if (board[i][j] != correct && correct < d * d)
239             {
240                 return false;
241             }
242             correct++;
243         }
244     }
245     return true;
246 }
```