```c
/**
 * find.c
 *
 * Computer Science 50
 * Problem Set 3
 *
 * Prompts user for as many as MAX values until EOF is reached,
 * then proceeds to search that "haystack" of values for given needle.
 *
 * Usage: ./find needle
 *
 * where needle is the value to find in a haystack of values
 */

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

#include "helpers.h"

// maximum amount of hay
const int MAX = 65536;

int main(int argc, string argv[])
{
    // ensure proper usage
    if (argc != 2)
    {
        printf("Usage: ./find needle\n");
        return -1;
    }

    // remember needle
    int needle = atoi(argv[1]);

    // fill haystack
    int size;
    int haystack[MAX];
    for (size = 0; size < MAX; size++)
    {
        // wait for hay until EOF
        printf("\nhaystack[%i] = ", size);
        int straw = GetInt();
        if (straw == INT_MAX)
        {
            break;
        }

```

```
49.         // add hay to stack
50.         haystack[size] = straw;
51.     }
52.
53.     // sort the haystack
54.     sort(haystack, size);
55.
56.     // try to find needle in haystack
57.     if (search(needle, haystack, size))
58.     {
59.         printf("\nFound needle in haystack!\n\n");
60.         return 0;
61.     }
62.     else
63.     {
64.         printf("\nDidn't find needle in haystack.\n\n");
65.         return 1;
66.     }
67. }
```

```
#
# Makefile
#
# Computer Science 50
# Problem Set 3
#

all: find generate

find: find.c helpers.c helpers.h
    clang -ggdb3 -O0 -std=c11 -Wall -Werror -o find find.c helpers.c -lcs50 -lm

generate: generate.c
    clang -ggdb3 -O0 -std=c11 -Wall -Werror -o generate generate.c

clean:
    rm -f *.o a.out core find generate
```

```c
/**
 * helpers.h
 *
 * Computer Science 50
 * Problem Set 3
 *
 * Helper functions for Problem Set 3.
 */

#include <cs50.h>

/**
 * Returns true if value is in array of n values, else false.
 */
bool search(int value, int values[], int n);

/**
 * Sorts array of n values.
 */
void sort(int values[], int n);
```

```c
/**
 * helpers.c
 *
 * Computer Science 50
 * Problem Set 3
 *
 * Helper functions for Problem Set 3.
 */

#include <cs50.h>

#include "helpers.h"

/**
 * Returns true if value is in array of n values, else false.
 */
bool search(int value, int values[], int n)
{
    int left, right,mid;
    left = 0;
    right = n-1;
    while(left <= right){
         mid = (left+right)/2;
        if(values[mid] == value){
            return true;
        }else if (values[mid] > value){
            right = mid - 1;        //search in the left half
        }else{
            left = mid + 1;        //search in the right half
        }
    }
    return false;
}

/**
 * Sorts array of n values.
 */
void sort(int values[], int n)
{
    int swap = 0;
    for(int i=0 ; i<n ;i++){
        for(int j=i+1 ; j < n; j++){
            if(values[i] > values[j]){
                swap = values[i];
                values[i] = values[j];
                values[j] = swap;
            }
        }
```

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

```c
/**
 * generate.c
 *
 * Computer Science 50
 * Problem Set 3
 *
 * Generates pseudorandom numbers in [0,LIMIT), one per line.
 *
 * Usage: generate n [s]
 *
 * where n is number of pseudorandom numbers to print
 * and s is an optional seed
 */

#define _XOPEN_SOURCE

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

// constant
#define LIMIT 65536

int main(int argc, string argv[])
{
    // Return 1 and exit if one or two command line arguments are not passed.
    if (argc != 2 && argc != 3)
    {
        printf("Usage: generate n [s]\n");
        return 1;
    }

    // Convert the first command line argument agrv[1] to integer.
    int n = atoi(argv[1]);

    // If two command line arguments are passed, convert it to long integer and pass it to srand48 as an argument.Else pass NULL to srand48()
    if (argc == 3)
    {
        srand48((long int) atoi(argv[2]));
    }
    else
    {
        srand48((long int) time(NULL));
    }

    // Execute the loop for the number of times as the first command line argument.Call drand48() to generate a pseudorandom int
    for (int i = 0; i < n; i++)
```

```
49.     {
50.         printf("%i\n", (int) (drand48() * LIMIT));
51.     }
52.
53.     // success
54.     return 0;
55. }
```

```
#
# Makefile
#
# Computer Science 50
# Problem Set 3
#

fifteen: fifteen.c
    clang -ggdb3 -O0 -std=c11 -Wall -Werror -o fifteen fifteen.c -lcs50 -lm

clean:
    rm -f *.o a.out core fifteen log.txt
```

1. 15|14|13|12
2. 11|10|9|8
3. 7|6|5|4
4. 3|1|2|0
5.

```c
/**
 * fifteen.c
 *
 * Computer Science 50
 * Problem Set 3
 *
 * Implements Game of Fifteen (generalized to d x d).
 *
 * Usage: fifteen d
 *
 * whereby the board's dimensions are to be d x d,
 * where d must be in [DIM_MIN,DIM_MAX]
 *
 * Note that usleep is obsolete, but it offers more granularity than
 * sleep and is simpler to use than nanosleep; `man usleep` for more.
 */

#define _XOPEN_SOURCE 500

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

// constants
#define DIM_MIN 3
#define DIM_MAX 9

// board
int board[DIM_MAX][DIM_MAX];

// dimensions
int d;

int blankRow;
int blankCol;

// prototypes
void clear(void);
void greet(void);
void init(void);
void draw(void);
bool move(int tile);
bool won(void);
bool adjacent(int row,int col,int tile);
int main(int argc, string argv[])
{
    // ensure proper usage
```

```c
    if (argc != 2)
    {
        printf("Usage: fifteen d\n");
        return 1;
    }

    // ensure valid dimensions
    d = atoi(argv[1]);
    blankRow = d-1;
    blankCol = d-1;
    if (d < DIM_MIN || d > DIM_MAX)
    {
        printf("Board must be between %i x %i and %i x %i, inclusive.\n",
            DIM_MIN, DIM_MIN, DIM_MAX, DIM_MAX);
        return 2;
    }

    // open log
    FILE* file = fopen("log.txt", "w");
    if (file == NULL)
    {
        return 3;
    }

    // greet user with instructions
    greet();

    // initialize the board
    init();

    // accept moves until game is won
    while (true)
    {
        // clear the screen
        clear();

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

        // log the current state of the board (for testing)
        for (int i = 0; i < d; i++)
        {
            for (int j = 0; j < d; j++)
            {
                fprintf(file, "%i", board[i][j]);
                if (j < d - 1)
                {
                    fprintf(file, "|");
```

```
 97.                 }
 98.             }
 99.             fprintf(file, "\n");
100.         }
101.         fflush(file);
102. 
103.         // check for win
104.         if (won())
105.         {
106.             printf("ftw!\n");
107.             break;
108.         }
109. 
110.         // prompt for move
111.         printf("Tile to move: ");
112.         int tile = GetInt();
113. 
114.         // quit if user inputs 0 (for testing)
115.         if (tile == 0)
116.         {
117.             break;
118.         }
119. 
120.         // log move (for testing)
121.         fprintf(file, "%i\n", tile);
122.         fflush(file);
123. 
124.         // move if possible, else report illegality
125.         if (!move(tile))
126.         {
127.             printf("\nIllegal move.\n");
128.             usleep(500000);
129.         }
130. 
131.         // sleep thread for animation's sake
132.         usleep(500000);
133.     }
134. 
135.     // close log
136.     fclose(file);
137. 
138.     // success
139.     return 0;
140. }
141. 
142. /**
143.  * Clears screen using ANSI escape sequences.
144.  */
```

```c
void clear(void)
{
    printf("\033[2J");
    printf("\033[%d;%dH", 0, 0);
}

/**
 * Greets player.
 */
void greet(void)
{
    clear();
    printf("WELCOME TO GAME OF FIFTEEN\n");
    usleep(2000000);
}

/**
 * Initializes the game's board with tiles numbered 1 through d*d - 1
 * (i.e., fills 2D array with values but does not actually print them).
 */
void init(void)
{
    int value = d*d - 1;
    bool isEven = false;
    if(value % 2 == 0){
        isEven = true;
    }
    for(int row=0; row<d; row++){
        for(int col=0; col<d; col++){
            board[row][col] = value;
            value--;
        }
    }
    if(!isEven){
        board[d-1][d-2] = 2;
        board[d-1][d-3] = 1;
    }
    board[d-1][d-1]  = 0 ;
}

/**
 * Prints the board in its current state.
 */
void draw(void)
{
    for(int row=0; row<d; row++){
        for(int col=0; col<d; col++){
            printf("%2d ",board[row][col]);
```

```
193.         }
194.         printf("\n");
195.     }
196. 
197. }
198. 
199. /**
200.  * If tile borders empty space, moves tile and returns true, else
201.  * returns false.
202.  */
203. bool move(int tile)
204. {
205.     bool found = false;
206.     int row;
207.     int col;
208.     for(row = 0;row < d && !found; row++){
209.         for(col = 0;col < d; col++){
210.             if(board[row][col] == tile){
211.                 found = true;
212.                 row--;
213.                 break;
214.             }
215.         }
216.     }
217.     if(adjacent(row,col,tile)){
218.         return true;
219.     }
220.     return false;
221. }
222. 
223. /**
224.  * Returns true if game is won (i.e., board is in winning configuration),
225.  * else false.
226.  */
227. bool won(void)
228. {
229.     int temp[d*d];
230.     int index = 0;
231.     for(int row=0;row<d;row++){
232.         for(int col=0;col<d;col++){
233.             temp[index++] = board[row][col] ;
234. 
235.         }
236.     }
237.     if(temp[d*d-1] != 0){
238.         return false;
239.     }
240.     for(int i=0;i< d*d-2;i++){
```

```
241.         if(temp[i] > temp[i+1]){
242.             return false;
243.         }
244.     }
245.     draw();
246.     return true;
247. }
248.
249. bool adjacent(int row,int col,int tile)
250. {
251.     if(col == blankCol-1 && row == blankRow){
252.         board[row][blankCol] = tile;
253.         board[row][col] = 0;
254.         blankCol--;
255.         return true;
256.     }else if(col == blankCol+1 && row == blankRow){
257.         board[row][blankCol] = tile;
258.         board[row][col] = 0;
259.         blankCol++;
260.         return true;
261.     }else if(row == blankRow-1 && col == blankCol){
262.         board[blankRow][col] = tile;
263.         board[row][col] = 0;
264.         blankRow--;
265.         return true;
266.     }else if(col == blankCol && row == blankRow+1){
267.         board[blankRow][col] = tile;
268.         board[row][col] = 0;
269.         blankRow++;
270.         return true;
271.     }
272.     return false;
273. }
274.
275.
```

```
 1. questions.txt
 2.
 3. Computer Science 50
 4. Problem Set 3
 5.
 6. 0. 3*3,5*5,6*6,7*7,8*8,9*9
 7. 1. 2 dimensional integer array
 8. 2. greet()
 9. 3. init(), draw(), move(), won()
10.
```