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

int main(void)
{
    int height;
    do
    {
        height = get_int("Height: ");
    }
    while (height < 1 || height > 8);

    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < height - i - 1; j++)
        {
            printf(" ");
        }
        for (int k = 0; k <= i; k++)
        {
            printf("#");
        }
        printf("  ");
        for (int k = 0; k <= i; k++)
        {
            printf("#");
        }
        printf("\n");
    }
}
```

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

int main(int argc, string argv[])
{
    if (argc != 4)
    {
        printf("Usage: ./calc <num> <op> <num>\n");
        return 1;
    }

    float num1 = (float) strtod(argv[1], NULL);
    float num2 = (float) strtod(argv[3], NULL);
    char op = argv[2][0];

    int quot = (int) (num1 / num2);
    float rem = num1 - (num2 * quot);

    switch (op)
    {
        case '+':
            printf("%f\n", num1 + num2);
            break;
        case '-':
            printf("%f\n", num1 - num2);
            break;
        case 'x':
            printf("%f\n", num1 * num2);
            break;
        case '/':
            printf("%f\n", num1 / num2);
            break;
        case '%':
            printf("%f\n", rem);
            break;
        default:
            printf("Operator must be +, -, x, /, or %%\n");
            return 1;
    }
}
```

```c
#include <cs50.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, string argv[])
{
    if (argc != 2)
    {
        printf("Usage: ./caesar <key>\n");
        return 1;
    }

    int key = (int) strtol(argv[1], NULL, 10);

    string plaintext = get_string("plaintext: ");
    printf("ciphertext: ");

    for (int i = 0; i < strlen(plaintext); i++)
    {
        if (isalpha(plaintext[i]))
        {
            if (isupper(plaintext[i]))
            {
                printf("%c", ((plaintext[i] - 'A' + key) % 26) + 'A');
            }
            else
            {
                printf("%c", ((plaintext[i] - 'a' + key) % 26) + 'a');
            }
        }
        else
        {
            printf("%c", plaintext[i]);
        }
    }

    printf("\n");
}
```

```c
// Implements Game of Fifteen (generalized to d x d)

#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
#define COLOR "\033[32m"

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

// Dimensions
int d;

// Saved locations of the blank tile
int blank_row;
int blank_col;

// Prototypes
void clear(void);
void greet(void);
void init(void);
void draw(void);
bool move(int tile);
bool won(void);
void swap(int *a, int *b);
void print_grid_row(int d);
void print_tile(int tile);

int main(int argc, string argv[])
{
    // Ensure proper usage
    if (argc != 2)
    {
        printf("Usage: fifteen d\n");
        return 1;
    }

    // Ensure valid dimensions
```

```c
    d = atoi(argv[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, "|");
                }
            }
            fprintf(file, "\n");
        }
        fflush(file);

```

```c
        // Check for win
        if (won())
        {
            printf("ftw!\n");
            break;
        }

        // Prompt for move
        int tile = get_int("Tile to move: ");

        // Quit if user inputs 0 (for testing)
        if (tile == 0)
        {
            break;
        }

        // Log move (for testing)
        fprintf(file, "%i\n", tile);
        fflush(file);

        // Move if possible, else report illegality
        if (!move(tile))
        {
            printf("\nIllegal move.\n");
            usleep(500000);
        }

        // Sleep thread for animation's sake
        usleep(50000);
    }

    // Close log
    fclose(file);

    // Success
    return 0;
}

// Clears screen using ANSI escape sequences
void clear(void)
{
    printf("\033[2J");
    printf("\033[%d;%dH", 0, 0);
}
```

```c
// 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 tile = d * d - 1;

    for (int i = 0; i < d; i++)
    {
        for (int j = 0; j < d; j++)
        {
            board[i][j] = tile;
            tile--;
        }
    }

    if (d % 2 == 0)
    {
        int temp = board[d-1][d-2];
        board[d-1][d-2] = board[d-1][d-3];
        board[d-1][d-3] = temp;
    }
}

// Prints the board in its current state
void draw(void)
{
    for (int i = 0; i < d; i++)
    {
        for (int j = 0; j < d; j++)
        {
            if (board[i][j] == 0)
            {
                printf("_\t");
            }
            else
            {
                printf("%d\t", board[i][j]);
```

```c
            }
        }
        printf("\n");
    }
}

// If tile borders empty space, moves tile and returns true, else returns false
bool move(int tile)
{
    for (int i = 0; i < d; i++)
    {
        for (int j = 0; j < d; j++)
        {
            if (board[i][j] == tile)
            {
                // check up
                if (i-1 >= 0 && board[i-1][j] == 0)
                {
                    board[i-1][j] = tile;
                    board[i][j] = 0;
                    return true;
                }
                // check down
                else if (i+1 < d && board[i+1][j] == 0)
                {
                    board[i+1][j] = tile;
                    board[i][j] = 0;
                    return true;
                }
                // check left
                else if (j-1 >= 0 && board[i][j-1] == 0)
                {
                    board[i][j-1] = tile;
                    board[i][j] = 0;
                    return true;
                }
                // check right
                else if (j+1 < d && board[i][j+1] == 0)
                {
                    board[i][j+1] = tile;
                    board[i][j] = 0;
                    return true;
                }
            }
        }
```

```c
    }
    return false;
}

// Returns true if game is won (i.e., board is in winning configuration), else false
bool won(void)
{
    int correct = 1;
    for (int i = 0; i < d; i++)
    {
        for (int j = 0; j < d; j++)
        {
            if (board[i][j] != correct && correct < d * d)
            {
                return false;
            }
            correct++;
        }
    }
    return true;
}
```