```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <time.h>

const int EMPTY = -1;

typedef struct SimulationState {
    int number_of_balls;
    int number_of_compartments;
    int number_of_rows;
} SimulationState;

bool ball_goes_right() {
    return rand() % 2 == 0;
}

void set_random_seed() {
    srand(time(0));
}

int safely_read_integer() {
    int integer;
    char* input = NULL;
    size_t input_length = 0;
    ssize_t getline_return = EOF;
    ssize_t sscanf_return;
    char character_after_number;

    while(getline_return == EOF || sscanf_return != 2 || character_after_number != '\n')
{
        if(input != NULL) {
            free(input);
            input = NULL;
            clearerr(stdin);
            printf("Input error. Try again.\n");
        }
        printf(">> ");
        getline_return = getline(&input, &input_length, stdin);
        sscanf_return = sscanf(input, "%d%c", &integer, &character_after_number);
    }

    return integer;
}

void init_array_with(int* array, size_t size, int value) {
    for(size_t i = 0; i < size; ++i) {
        array[i] = value;
    }
}

void count_and_clear_last_row(int** board, int* histogram, SimulationState state) {
    int* last_row = board[state.number_of_rows - 1];
    for(int i = 0; i < state.number_of_compartments; ++i) {
        if(last_row[i] != EMPTY) {
            ++histogram[i];
            last_row[i] = EMPTY;
            return;
        }
    }
}

/*
Starts with the last row and works itself up.
The last row is already assumed to be empty.
*/
void let_balls_fall_1_row(int** board, SimulationState state) {
    for(int i = state.number_of_rows - 1; i > 0; --i) {
        int* row_above = board[i - 1];
```

```
        int size_of_row_above = i + 1;

        // Get the index and ball number of the ball in the row above
        int ball_number;
        int index_of_ball_above = -1;
        for(int j = 0; j < size_of_row_above; ++j) {
            if(row_above[j] != EMPTY) {
                ball_number = row_above[j];
                index_of_ball_above = j;
                break;
            }
        }
        if(index_of_ball_above == -1) {
            //No ball in the above row, continue
            continue;
        }

        //Determine the new index of the ball in the current row
        int new_index = index_of_ball_above;
        if(ball_goes_right()) {
            new_index = index_of_ball_above + 1;
        }

        //Assign the ball to the new index and remove it from the old
        int* current_row = board[i];
        current_row[new_index] = ball_number;
        row_above[index_of_ball_above] = EMPTY;
    }
}

void insert_ball_at_top(int** board, SimulationState state) {
    if(state.number_of_balls <= 0) {
        return;
    }
    int index = ball_goes_right()? 1 : 0;
    board[0][index] = state.number_of_balls;
}

/*
The histogram will be displayed in a quadratic grid.
So it needs to be normalized to the number of compartments.
*/
void display_histogram(int* histogram, SimulationState state) {
    int max = EMPTY;
    for(int i = 0; i < state.number_of_compartments; ++i) {
        if(histogram[i] > max) {
            max = histogram[i];
        }
    }

    int number_of_balls_per_X = (int)round(max / state.number_of_compartments);
    if(number_of_balls_per_X < 1) {
        number_of_balls_per_X = 1;
    }
    for(int i = state.number_of_compartments; i > -1; --i) {
        for(int j = 0; j < state.number_of_compartments; ++j) {
            if(histogram[j] >= (i + 1) * number_of_balls_per_X) {
                printf("X ");
            }
            else {
                if(i == 0) {
                    printf("- ");
                }
                else {
                    printf("  ");
                }
            }
        }
        printf("\n");
    }
```

```
    printf("\nX = %d balls. Underlying histogram:\n", number_of_balls_per_X);
    for(int i = 0; i < state.number_of_compartments; ++i) {
        printf("%d ", histogram[i]);
    }
}

int** create_board(SimulationState state) {
    int** board = (int**)malloc(state.number_of_rows * sizeof(int*));
    for(size_t i = 0; i < state.number_of_rows; ++i) {
        size_t size = i + 2;
        board[i] = (int*)malloc(size * sizeof(int));
        init_array_with(board[i], size, EMPTY);
    }

    return board;
}

/*Simulation concept:
    Let's say there are 2 balls and 3 compartments. The board
    will have 3 - 1 = 2 rows.
    Every ball will have a unique number starting with 1.
    Every row of the board corresponds to an n+2 array, where n
    is the row number (starting with zero). It will contain a ball
    number at the ball's current position and -1 everywhere else.
    Every iteration, a ball is put in the first row, if any are left.
    Every following ball will go down 1 row.
    After two iterations, the arrays may look like this:
    -1 2
    -1 -1 1
*/
int* run_simulation(SimulationState state) {
    set_random_seed();
    int** board = create_board(state);

    int* histogram = (int*)malloc(state.number_of_compartments * sizeof(int));
    init_array_with(histogram, state.number_of_compartments, 0);

    int number_of_iterations = state.number_of_balls + state.number_of_rows;
    for(int i = 0; i < number_of_iterations; ++i) {
        insert_ball_at_top(board, state);
        let_balls_fall_1_row(board, state);
        count_and_clear_last_row(board, histogram, state);
        --state.number_of_balls;
    }

    return histogram;
}

SimulationState read_simulation_state() {
    SimulationState state = {.number_of_balls = -1, .number_of_compartments = -1};

    printf("A Galton Board simulation.\n"
           "First, type in the number of balls.\n");

    while(state.number_of_balls <= 0) {
        printf("The number of balls needs to be positive.\n");
        state.number_of_balls = safely_read_integer();
    }

    printf("Now, type in the number of compartments.\n");
    printf("This should be lower or equal to the number of columns of your terminal to di
splay the histogram correctly.\n");
    while(state.number_of_compartments <= 1) {
        printf("The number of compartments has to be greater than 1.\n");
        state.number_of_compartments = safely_read_integer();
    }
    state.number_of_rows = state.number_of_compartments - 1;

    printf("\nRunning simulation with: \n"
```

```
            "Number of balls: %d\n"
            "Number of compartments: %d\n\n",
            state.number_of_balls, state.number_of_compartments);

    return state;
}

int main() {
    SimulationState state = read_simulation_state();

    int* histogram = run_simulation(state);
    display_histogram(histogram, state);

    return 0;
}
```