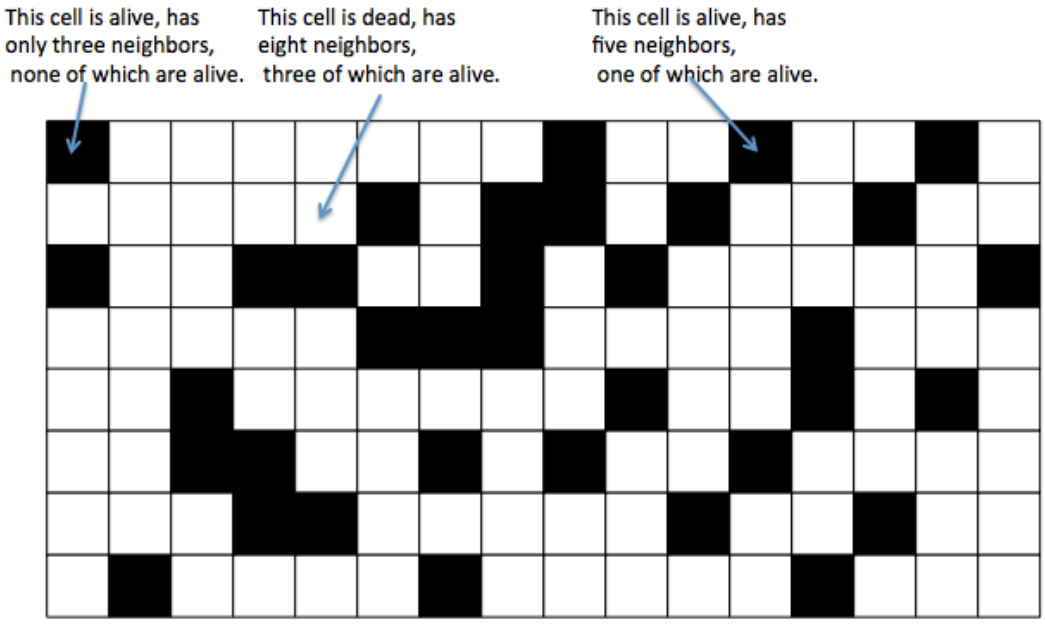
Cells

Cells is a simulation of a simple world that exists on a two-dimensional grid.

Each element in the grid is called a “cell”. Cells are either alive or dead.

Cells that do not border an edge have 8 neighbors. Cells on a single edge have 5 neighbors. Cells that are in the corner have 3 neighbors.

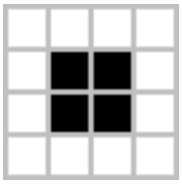


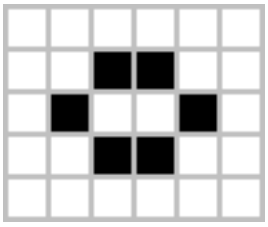
Each new generation is created with the following rules:

- Live cells with fewer than two live neighbors die of loneliness.
- Live cells with two or three live neighbors live on to the next generation.
- Live cells with more than three live neighbors die from overcrowding.
- Dead cells with exactly three live neighbors are reborn as live cells.

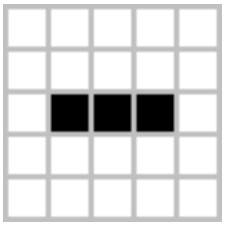
Some Example Patterns:

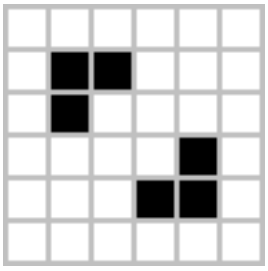
The following two patterns should not change between generations:



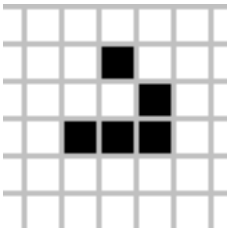


The following two patterns should repeat with a period of 2.





The following pattern should move across the grid, provided they collide or interact with stuff in the way.



The Assignment

Given an input file, write a program that generates a sequence of new world-states over time. The width, height, number of generations, and initial state are specified in the input file. After each new state is generated, print an ASCII representation of world-state to the terminal screen. This assignment should be completed in C without any C++ constructs (it should compile with a C compiler).

The Input File

The input file for the grid pictured above has the following format:

```
16
8
12
1 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0
0 0 0 0 0 1 0 1 1 0 1 0 0 1 0 0
1 0 0 1 1 0 0 1 0 1 0 0 0 0 0 1
0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0
0 0 1 1 0 0 1 0 1 0 0 1 0 0 0 0
0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 0
0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0
```

- The first line is the width of the grid, width > 0.
- The second line is the height of the grid, height > 0.
- The third line is the number of generations to run the sim (the state will change 12 times, so there is 13 states if you include the initial state).
- Finally, the grid. Each alive cell is represented by a 1, and dead by a 0.
- Each cell in the same row is separated by a single space. After the final cell in a row, there is no space, only a newline.

Provided Code

I am providing some helper code for this assignment, which you are not required to use. However, if you decide not to use my code, your program should still work with any input file that follows the above format. Output can have any representation as long as it's recognizable as a cell grid that changes states.

char *get_input_file_name(int argc, char **argv)
Takes argc and argv from main() and returns a pointer to the first command line argument. If there was no command line argument, the default return value is "input.txt"

void get_dimensions(char *input_filename, unsigned int *width, unsigned int *height)
gets the dimensions from the input file and puts them in the locations pointed to by 'width' and 'height'.

void get_generations(char *input_filename, unsigned int *generations);
Puts the number of generations from the input file into the location pointed to by 'generations'.

int **input_to_2d_array(char *input_filename, unsigned int width, unsigned int height);
Dynamically allocates a 2d array and fills it from the input file. Elements are accessed by <array_name>[x][y] where x < width, y < height (zero-indexed).

int **allocate_2d_array(unsigned int width, unsigned int height)
Dynamically allocates a 2d integer array filled with zeroes. Elements are accessed by <array_name>[x][y] where x < width, y < height (zero-indexed).

void free_2d_array(int **array, unsigned int width, unsigned int height)
Deallocates arrays that were allocated from 'input_to_2d_array' and 'allocate_2d_array'.

void print_2d_array(int **array, unsigned int width, unsigned int height)
Prints a 2d array to the console.

void sleep_250_ms(void)
Sleeps the process for 250ms. Use this function between states to slow down your simulation.

Feel free to modify/improve my helper functions; there are some glaring inefficiencies that I left intentionally because I thought it would make them easier to use.

Helpful Functions to Find on Google

from stdio.h:
`fscanf`
`fopen`
`fclose`
`printf`
`fprintf`

from stdlib.h
`system("clear")`
`malloc`
`free`

Requirements

- Cells follow the rules specified above.
- Print every state. The rate of printing should be slow enough to be observed.
- Accepts the input format specified above.
- Prints the specified number of generations.
- Cells should behave correctly for any input file. Write your own tests to be sure. For example, what if the grid is not square or only has one row?
- Use dynamic memory. Do not use a large static buffer such as **int arr[1000][1000]** to hold your 2d array.
- No memory leaks. For every call to malloc, there should be a call to free.

Bonus

- Ignore some or all of my starter code and write from scratch.
- Write the last frame to a file called “output.txt”
- Instead of ascii, create a graphical interface to display your cells. As long as the simulation is in C, the interface can be written with whatever suits you.
- Simulate your cells with Trick. I only recommend this if you have little to learn from doing it in pure C.