Implement each function.
Goes into universe.c

Universe *uv_create(int rows, int cols, bool toroidal)
Allocate memory for the universe using calloc()
Memory can be allocated starting at “rows” like an array
In that array of row pointers each row will contain the pointer to the column

void uv_delete(Universe *u)
Want to free up memory space
Loop through each row we allocated and free each one

Also needs to free up the entire grid that was allocated
And free the universe itself “u”

int uv_rows(Universe *u)
Return the number of rows defined from the universe struct

int uv_cols(Universe *u)
Return the number of columns defined the universe struct

void uv_live_cell(Universe *u, int r, int c)
Make sure the cell you are trying to access is valid (within grid range)
Access the cell in the grid, like rolls from asgn1 [r][c]
Make that cell TRUE to mean it's alive

void uv_dead_cell(Universe *u, int r, int c)
Same thing as live_cell except make cell = false for dead

bool uv_get_cell(Universe *u, int r, int c)
Getting the value of the cell (T,F) at specified row and column

bool uv_populate(Universe *u, FILE *infile)
Check for null in file or universe
Make a loop to scan in the two numbers from the FILE / use fscanf
Populate the cell at the scanned in coordinates (make the cell alive)
Keep scanning in numbers until end of file/End the loop at EOF

Return true at end of function
If an error occurs return false like out of bounds

int uv_census(Universe *u, int r, int c)

```
    Counts number of live neighbors
    Make a counter and loop through each neighbor, if alive then increment counter
    Do not count starting cell into census

    If the universe if toroidal then also count the neighbors that wrap around
    Add dimensions of grid to both column and row then mod it by the dimention
    Add row dimension to r and % row dimension (same for column) for wrap around
```

void uv_print(Universe *u, FILE *outfile)

```
    Loop through each cell starting at [0][0] / always prints flattened universe
    Print o for alive and . for dead cell
```

Playing the game of life

Goes into life.c

Use getopt() to go through command-line options
Like in asgn2 when using the switch statement

Next use fscanf() to read from the file to get the dimensions of the universe to create

Create two universes using the specified dimensions, if -t then mark toroidal (true)

Populate the first universe, A, using the populate function
Using the input of the file

Setup ncurses screen

```
Make a loop to go through the specified number of generations to go through:
    If -s != true, clear the screen display universe A,refresh, then sleep

    Nested loop:
        Take the census of each cell in universe A
        Based off of the rules, set the specific cell you are in in A apply the change in B
        Get out of loop when going through each cell in grid

    Swap the universes by swapping the pointers

Close the screen with endwin()
```

Make the last output of A into uv_print()

For next generation:

- Have 2 grids one A one B
- If a live cell has 2 or 3 as census then in the next generation it will be alive (true)
- If a dead cell has 3 neighbors then it becomes alive in next generation
- Else all other cells die
- Mark it dead with uv_dead_cell

  when A is done transferring to generation B, B points back to A for the next generation

  Swap the pointer of the A and B grid to go to next generation

Universe.c

- Defining the universe struct
- Implementing the functions after

Life.c

- Using the functions in universe.c to play game
- Contains main function
- If there is an error in a function return false?

ncurses

Go through all rows and all columns in nested for loop