A Lot in Life

Angela Shen

February 2023

1 Introduction

Deliverables:

universe.c

life.c

The following describes the basic pseudocode for the implementation of universe.c and life.c.

2 Universe.c

Universe.c contains the implementation of the universe structure using the following functions:

Universe *uv_create(uint32_t rows, uint32_t cols, bool toroidal): Creates the universe. Based on pseudocode in assignment 4. Uses parameter rows to calloc an array of rows to **grid, then uses a for loop and parameter cols to calloc the columns to create a 2 dimensional matrix. Also assigns .

void uv_delete(Universe *u): Iterate through grid and use free() to free the space and prevent memory leaks. Free grid and then free universe (must be in this order).

uint32_t uv_rows(Universe *u): Accessor function that returns the value of rows.

uint32_t uv_cols(Universe *u): Accessor function that returns the value of cols.

void uv_live_cell(Universe *u, uint32_t r, uint32_t c): Manipulator function that changes the boolean at row r, column c to be true.

void uv_dead_cell(Universe *u, uint32_t r, uint32_t c): Manipulator function that changes the boolean at row r, column c to be true.

bool uv_get_cell(Universe *u, uint32_t r, uint32_t c): Returns the boolean value of the cell at row r, column c. Also returns false if row and column are out of bounds.

bool uv_populate(Universe *u, FILE *infile): Uses a while loop with fscanf() to read the given pairs from infile. If there is an odd number of inputs, a number is negative, or a number is larger than the size of grid (rows or columns), then return false. Otherwise, return true.

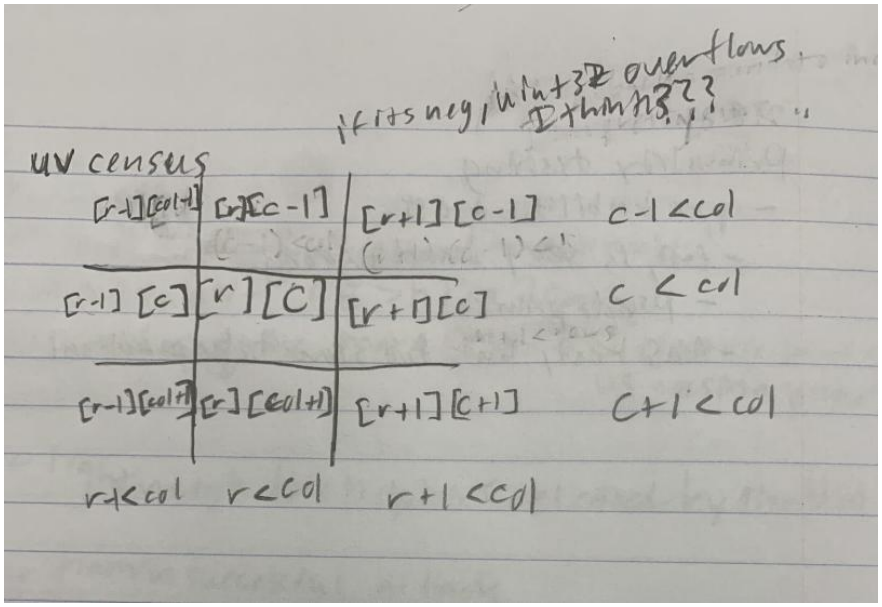
void uv_print(Universe *u, FILE *outfile): Outputs the final result of universe to the output file using fprintf in a while loop. It loops through the rows and columns, checking each cell in the grid to see if it is alive or dead, printing "o" if it is alive and "." if not. It also adds a newline after every row.

uint32_t uv_census(Universe *u, uint32_t r, uint32_t c): Counts number of live neighbors.

Admittedly my implementation of census is not the smartest. I tried to work with other ways, but I made them too convoluted in my mind which made for unclear code.

Anyways, each possible adjacent neighbor of grid[r][c] is checked individually (each diagonal and the squares above, below, to the right, and to the left of it). It initially checks if the dimensions of the neighbor are a valid square on the grid, if it's non-toroidal, or if the grid is toroidal (meaning the neighbor will exist, but may loop around to another end of the grid). If it is toroidal and the dimensions of the neighbor would be considered invalid on a non-toroidal, then it loops around to another spot on the grid and increments the number of neighbors if that cell is alive. If the dimensions are valid on a non-toroidal, then it checks if that neighbor is alive and increments the number of neighbors.

Figure 1: An illustration of the possible adjacent neighbors of grid[r][c], trying to figure out what needs to be checked for each neighbor.



3 Life.c

Using a while loop and a switch, the main function parses the above command line options:

-t: Sets the toroidal boolean equal to True.

-s: Sets the boolean for s equal to True, which silences ncurses (nothing will be displayed by ncurses).

-n generations: Specify the number of generations that the universe goes through. The default number of generations is 100.

-i input: Has an argument input, which is the file to input the population of the universe (stdin by default). Uses fopen to open the file and strcopy to copy the optarg string.

-o output: Has an argument output, which is the file to output the final state of the universe (stdout by default). Uses fopen to open the file and strcopy to copy the optarg string.

-h: Displays the help message.

Then creates a populated universe a based on the input file (may be stdin). Goes through multiple generations of the universe (displayed using ncurses if not silenced), taking a census of the entire grid each time to alter the second universe b. Universes b and a are swapped, continuing until the maximum number of generations is reached. Prints the final output to the output file (may be stdout).