

Assignment 4 DESIGN

Description of Program

In this assignment, we will be recreating the Game of Life, created by John Horton Conway in C. The game of life follows a specific set of rules, described in detail later, and is played using a 2D grid of cells that represent a universe. Life, is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves. Each cell is either dead or alive based on the set of rules, and the grid if stated can be toroidal.

The Rules

1. Any live cell with two or three live neighbors survives.
2. Any dead cell with exactly three live neighbors becomes a live cell.
3. All other cells die, either due to loneliness or overcrowding.

The Deliverables

(files that need to be included in the directory “asgn4”)

1. universe.c implements the Universe ADT.
2. universe.h specifies the interface to the Universe ADT. This file is provided and may not be modified.
3. life.c contains main() and may contain any other functions necessary to complete your implementation of the Game of Life.
 - -t : Specify that the Game of Life is to be played on a toroidal universe.
 - -s : Silence ncurses. Enabling this option means that nothing should 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 : Specify the input file to read in order to populate the universe. By default the input should be stdin.

- -o output : Specify the output file to print the final state of the universe to. By default the output should be stdout.

4. Makefile:

- CC = clang must be specified.
- CFLAGS = -Wall -Wextra -Werror -Wpedantic must be specified.
- make must build the life executable, as should make all and make life.
- make clean must remove all files that are compiler generated.
- make format should format all your source code, including the header files.

5. README.md: This must use proper Markdown syntax. It must describe how to use your program and Makefile. It should also list and explain any command-line options that your program accepts. Any false positives reported by scan-build should be documented and explained here as well. Note down any known bugs or errors in this file as well for the graders.

6. DESIGN.pdf: What I am writing right now.

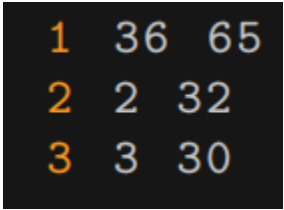
7. WRITEUP.pdf: This document must be a proper PDF. This writeup document must include everything you learned from this assignment. Make sure to mention everything in detail while being as precise as possible. How well you explain all the lessons you have learned in this assignment will be really important here.

Pseudocode / Structure

*Code will be spread out across multiple files

<p><i>Step 1. Creating the universe with Universe.c</i></p>	<p>The header:</p> <ul style="list-style-type: none"> - stdio.h (the standard library for C) - Need to use stdlib.h to use calloc() and malloc() to dynamically allocate memory like in asgn3 - universe.h - cannot modify, declares the new type and universe.c defines its concrete implementation - An instance of a universe contains rows, cols, and a 2-D boolean grid. False in the grid if dead, true in the grid if alive (cells)
---	--

- Universe *uv_create(uint32_t rows, uint32_t cols, bool toroidal)
 - This is where we will create our Universe
 - We need to dynamically allocate memory who, based on columns and rows.
- void uv_delete(Universe *u) Frees any memory allocated for a universe by the constructor function.
- uint32_t uv_rows(Universe *u) an accessor function. Opaque data type = user-defined data structure. Allows us to hold a pointer to a structure, but not modify or view its contents directly. We want to return the number of rows inside universe.c
- uint32_t uv_cols(Universe *u) - same as rows but with columns
- void uv_live_cell(Universe *u, uint32_t r, uint32_t c)
 - The need for manipulator functions follows the rationale behind the need for accessor functions: we need some way to alter fields of a data type. This function simply marks the cell at row r and column c as live. If the specified row and column lie outside the bounds of the universe, *nothing changes*. Since we are using bool, we assume that true means live and false means dead.
- void uv_dead_cell(Universe *u, uint32_t r, uint32_t c)
 - Opposite of live cell
- bool uv_get_cell(Universe *u, uint32_t r, uint32_t c)

	<ul style="list-style-type: none"> - This function returns the value of the cell at row <i>r</i> and column <i>c</i>. If the row and column are out-of bounds, false is returned. Again, true means the cell is live. - Bool uv_populate(Universe *u, FILE *infile) <ul style="list-style-type: none"> - Populates our universe - 36 65 (number of rows, number of columns in the universe) - 2 32 (Row and column of a live cell) <ul style="list-style-type: none"> - Use a loop that calls fscanf() - If a pair lies outside the universe m return false. Print an error message. Return true if successful - uint32_t uv_census(Universe *u, uint32_t r, uint32_t c) <ul style="list-style-type: none"> - Creates a toroidal universe. - You should calculate the row and column for each neighbor and apply modular arithmetic if the universe is toroidal. - It never falls off the earth/universe, will go around to the other side. - void uv_print(Universe *u, FILE *outfile) <ul style="list-style-type: none"> - Prints out the universe to outfile. A live cell 'o' and a dead cell '.' Use either fputc() or fprintf() to print to the specified outfile. Since you cannot print a torus, you will always print out the flattened universe.
<p><i>Step 2: Creating the Makefile</i></p>	<p>Since no Makefile is provided, we need to make it ourselves to be able to compile and run our files.</p> <ol style="list-style-type: none"> 1. Make sure to use CFLAG and CLANG

<p><i>Step 3: Controlling the curser</i></p>	<ul style="list-style-type: none"> - To be able to use the curser implement this code provided for us using ncurses.h <pre> Short ncurses example. 1 #include <ncurses.h> 2 #include <unistd.h> // For usleep(). 3 4 #define ROW 0 5 #define DELAY 50000 6 7 int main(void) { 8 initscr(); // Initialize the screen. 9 curs_set(FALSE); // Hide the cursor. 10 for (int col = 0; col < 40; col += 1) { 11 clear(); // Clear the window. 12 mvprintw(ROW, col, "o"); // Displays "o". 13 refresh(); // Refresh the window. 14 usleep(DELAY); // Sleep for 50000 microseconds. 15 } 16 endwin(); // Close the screen. 17 return 0; 18 } </pre> <p>Make sure to close the screen using endwin()</p>
<p><i>Step 4: Creating command line options in life.c main()</i></p>	<p>As listed in the deliverables, these options need to be implemented in the test case or place where we run the game, life.c() like in previous assignments, using the main() and getopt().</p> <p>We can create universes and test our universe.c functions here.</p>

Specifics not mentioned already

1. Create two universes using the dimensions that were obtained using fscanff(). Mark the universes toroidal if the -t option was specified. We will refer to these universes as universe A and universe B.
2. Populate universe A using uv_populate() with the remainder of the input.
3. For each generation up to the set number of generations:
 - a. If ncurses isn't silenced by the -s option, clear the screen, display universe A, refresh the screen, then sleep for 50000 microseconds.
 - b. Perform one generation. This means taking a census of each cell in universe A and either setting or clearing the corresponding cell in universe B, based on the rules.
 - c. Swap the universes. Think of universe A as the current state of the universe and universe B as the next state of the universe. To update the universe then, we

simply have to swap A and B. Hint: swapping pointers is much like swapping integers.

4. Output universe A to the specified file using `uv_print()`. This is what you will be graded on. We will know if you properly evolved your universe for the set number of generations by comparing your output to that of the supplied program.