Assignment 4 DESIGN.pdf

Victor Nguyen

February 3, 2022

1 Description of Program:

This program will simulate the well known, Game of Life. The program will take in some file to read and under specific rules will display the generations of the file. The user may also specify other things such as:

- t: Specify whether or not we use a toroidal grid or not.
- s: Silence ncurses (more on this later).
- n generations: Specify the number of generations to iterate over.
- i input: Input file to read from to start the universe.
- o output: Output file to write to after iterating over all the generations.

The program will terminate when we iterate through all the generations.

2 Files to be included in directory "asgn4":

1. life.c
    - Source file that contains main() and is responsible for printing out the game of life.
2. universe.c
    - File responsible for the implementations of the rules and creation of the universe.
3. universe.h
    - Header file that includes the functions of universe.c, useful for linking to other files.
    - This file was given.
4. test.c
    - Source file that I used to test random things.
5. Makefile
    - File that formats the program into clang-format.

- File also compiles life.c and test.c.
- Cleans files that were generated during the compilation process.
- File also contains scan-build and valgrind to check for any errors/memory leaks.

6. README.md
    - A text file that's in Markdown format that describes how you would build/run the program.
7. DESIGN.pdf
    - Describes the design and thought process of the main program.
8. EXTRA_TESTS.pdf
    - A document where I talked about testing irregular/malformed inputs. I also wrote this document for others to see, which I posted in the class discord in asgn4.

3 Structure for universe.c:

Create a struct universe to hold all the values pertaining to the universe. The rest of universe.c will contain other functions to perform operations to this struct.

3.1 Creating the Universe

Starting with uv_create, this function is responsible for creating a universe onto the heap. My first idea is to create the universe onto the heap. Afterwards, set the other parameters inside the struct. Lastly return the universe's address on the heap.

3.2 Deleting the Universe

Since we created and stored something onto the heap, we need to clear it from the heap. C doesn't have a built in memory garbage collection, so we need to make one ourselves. The function uv_delete will free the heap space used from the universe.

3.3 Getting the Number of Rows and Columns

Next, I need to create functions that can tell us the numbers of rows and columns of a given universe, so uv_rows and uv_cols will return the rows/columns given the universe.

3.4 Setting Cells Alive/Dead and Getting the State of the Cell

I also need a way to make a particular cell alive or dead. So in uv_live_cell, given a row and column, set that position to be true. For uv_dead_cell we should do something similar and set the position to false. I also need to make a function called uv_get_cell that gets the current state of a cell.

3.5 Populating the Universe

The populate function requires a universe and also a pointer to a file. What I'll do is parse through the file and run the values into our uv_live_cell function to make cell alive. If at any point that I can't populate one of the pairs of data, I'll return false to signify that the population process failed to work.

3.6 Counting Neighbors

Next, I need to make a function that checks the surrounding area for live cells. Keeping in mind that the universe can be toroidal or not we first need to check that. So if the universe is non toroidal, I'll need to iterate and check the surrounding cells and increase the count for the number of live cells in the vicinity. Otherwise, for a toroidal space, if the cell we are looking at are on the border of the grid, I'll make sure to account for the edge neighbors.

3.7 Printing the Universe

Lastly, we need a function to print all the data of the universe into some file. I'll need to iterate over the rows and columns of the grid and print the respective state of the cell into the file.

3.8 Note about the pseudocode:

- This source file is a necessary implementation for the game of life.
- Struct - A struct universe should contain 4 key details:
  - Rows - The number of rows for a given universe.
  - Columns - The number of columns for a given universe.
  - 2D array (grid) - The universe itself, which should contain booleans.
  - Toroidalness - Whether the universe wraps around itself.
- The grid should also be stored onto the heap. This also means we will need to free the heap space used by the grid.
- When I set the cell to live or dead, we expect some value for the row and column. I should also check if the number is in the bounds of the grid. Otherwise, I'll probably get errors down the line, when I try to populate the grid.
- If there is an attempt to get the current state of a cell that lies outside the grid, just return false. This is because I'm assuming that anything outside of a specified universe is nonexistent, or dead.
- In my populate function, I should assume the file that is being read is formatted correctly. The first line will determine the size of the grid so I should of used this line already. The rest of the file should be positive integers in the form of having the row value being first then the column.

4 Pseudocode / Structure of life.c:

```
I want to implement a usage page that will display if the user inputs nothing
or when it's an invalid input. To do this, I'll create a function called usage
that takes in no arguments.


Get the user input for the file to read and store it for later usage. We also
should check if the user specified an output file to write too.

See if the user specified the number of generations and store it for later
use.

Check if the user specified whether or not to silence ncurses or not, and
store this for later use. Same should be applied for toroidal.

Check to see if the user specified an input file, if so:
    Open the file specified:
        Check to see if opening the file was unsuccessful, if so:
            Print an error saying we couldn't open the file.
            Return EXIT_FAILURE.
        Scan the file and store the first line.
Otherwise, scan the standard input and store the first line.

Create universe a and b using the row and columns that we just got.

Check again if the user specified an input file, if so:
    Populate universe a with the user specified input file.
    Check to see if we failed to populate all points on to the universe.
        If so, print out a message that tells us we failed to.
        Return EXIT_FAILURE.
    Close the file.
Otherwise, populate universe a with the standard input.
    Check to see if we failed to populate all points on to the universe.
        If so, print out a message that tells us we failed to.
        Return EXIT_FAILURE.

Initialize the screen.
Iterate through the generations.
    Check to see if the silence option is false.
        Clear the screen.
        Iterate through the rows.
            Iterate through the columns.
                Check if the current cell at the specified row and column is alive.
                    If so, print out an o to signify it's alive.
```

```
                    Refresh the screen.
        Delay the screen.

    Iterate through the rows.
        Iterate through the columns.
            Write the necessary conditions and checks for the game of life.

    Swap the universe a and universe b.
Terminate the window.

Check if the user specified an output file, if so:
    Open the file for writing.
        Use uv_print to print to this file.
    Close the file.
Otherwise, print to standard output.

Delete the universes that we created.
Return 0.
```

4.1 Notes about Pseudocode / Structure:

- In the game of life, there are a few rules to be mindful of.
    1. Any live cell with two or three live neighbors will live on.
    2. Any dead cell with three live neighbors becomes alive.
    3. All other cells die due to loneliness/overcrowding.
- The program should use getopt to get the command line arguments from the user.
- ncurses is a library used to display the current state of the universe. Some functions that I'll use will be:
    - initscr - Initialize the screen.
    - curs_set - Show or hide the cursor.
    - clear - clears the window.
    - mvprintw - used to display something specific at a specific row and column.
    - refresh - Refreshes the screen so that you can see what should be displayed.
    - usleep - Adds a delay to the code so that we can visualize the generations.
    - endwin - Closes the screen.
- The silence option should be off by default.
- The number of generations should be defaulted to 100.
- The input and output should have default inputs of stdin and stdout.
- The toroidal option should be defaulted to false.

- You should use fscanf to get the initial line and store those values. These are the x and y values that will determine the grid size.
- Printing out a message to the user when populate failed to do it's job will tell the user that there is an invalid input inside their input file.
- I built a separate function called swap that swaps the universes. This is so that we can perform however many generations using three universes.
  - The swap function should take in two pointers to the universes and the boolean statement for if the universe is toroidal or not.
  - Create a temporary universe and clean the temporary universe. This is to ensure that we cleaned the heap that this universe uses before we point it at universe a. Otherwise, we'd lose the pointer to the original temporary universe.
  - Perform the swap.

5 Error Handling:

Some problems and solutions I found while coding.

1. If the user ever inputs anything incorrectly, or nothing I will display the help page.
2. I had a hard time figuring out why I kept getting segmentation faults in my swap function. As I've aforementioned before, I just needed to clear the temporary universe before using it.
3. I did some testing and realized a few things, here are some of them.
   - I noticed that if you inputted a negative number for one of the rows or columns when during the process of creating the universe, you'll get a segfault. To fix this, I made sure to return an error message that tells the user they have a malformed input.
   - Another segfault error can occur when you specify ./life to read from a file that doesn't exists. To fix this, I printed out an error message to the user that the program failed to open the file.
   - One error that I couldn't fix is if the user didn't input any files at all for reading. Resulting in the program running forever. I thought I should let you know this.

6 Credit:

- I've tried to follow the information provided in asgn4 to implement my universe.c and life.c.
- I attended Eugene's section on January 28th. He helped me understand the basics of the assignment. He gave me ideas on how to implement the universe functions. He also helped me understand how to open files, using fscanf, and normalizing toroidal space.
- I've used collatz.c as a reference on how to use this very cool line of code for if statements. You'll see me use it a couple of times in universe.c and life.c.
- Audrey's example design pdf helped me structure my own design pdf. This can be found on this quarters cse13s discord server in the asgn-1-faq-n-tips channel.

- I used my asgn3 sorting.c file as reference and inspiration to implementing my switch cases.
- I've used the asgn4.pdf command line options as my synopsis help page for the ./life executable.
- The C Programming Language book may have influenced some of the code I've written.
- When calculating the neighbors for the census function, I've done a similar assignment in the past, specifically from my CSE 30 class from UCSC. The assignment was called the Holy Queens problem. I'm not going to go over every detail of the assignment, but I remember using an idea that helped get the neighbors for the surrounding cell. How it works is we keep a library of all the different directions we can "traverse". Then iterate over all the directions. If you want the file to the assignment, I can send it if it's asked for.
- Ben's tutorial on how to compile and link multiple files together was really helpful. I also followed his scan-build guide on the hilalmorrar.com website.
- I've used my own Makefile from asgn3 as reference to build the makefile for this assignment.
- I've used the provided example of ncurses from the asgn 4 pdf to make my own ncurses.
- Thanks to the discord user isn#1107 and jlortiz#0850 for helping me test and debug errors that the user could input.
- I took inspiration from the discord user Elmer#1515 on implementing fscanf.