Game of Life

Hint: The Answer is 42

Lucais Sanderson

11 February 2023

1 Intro

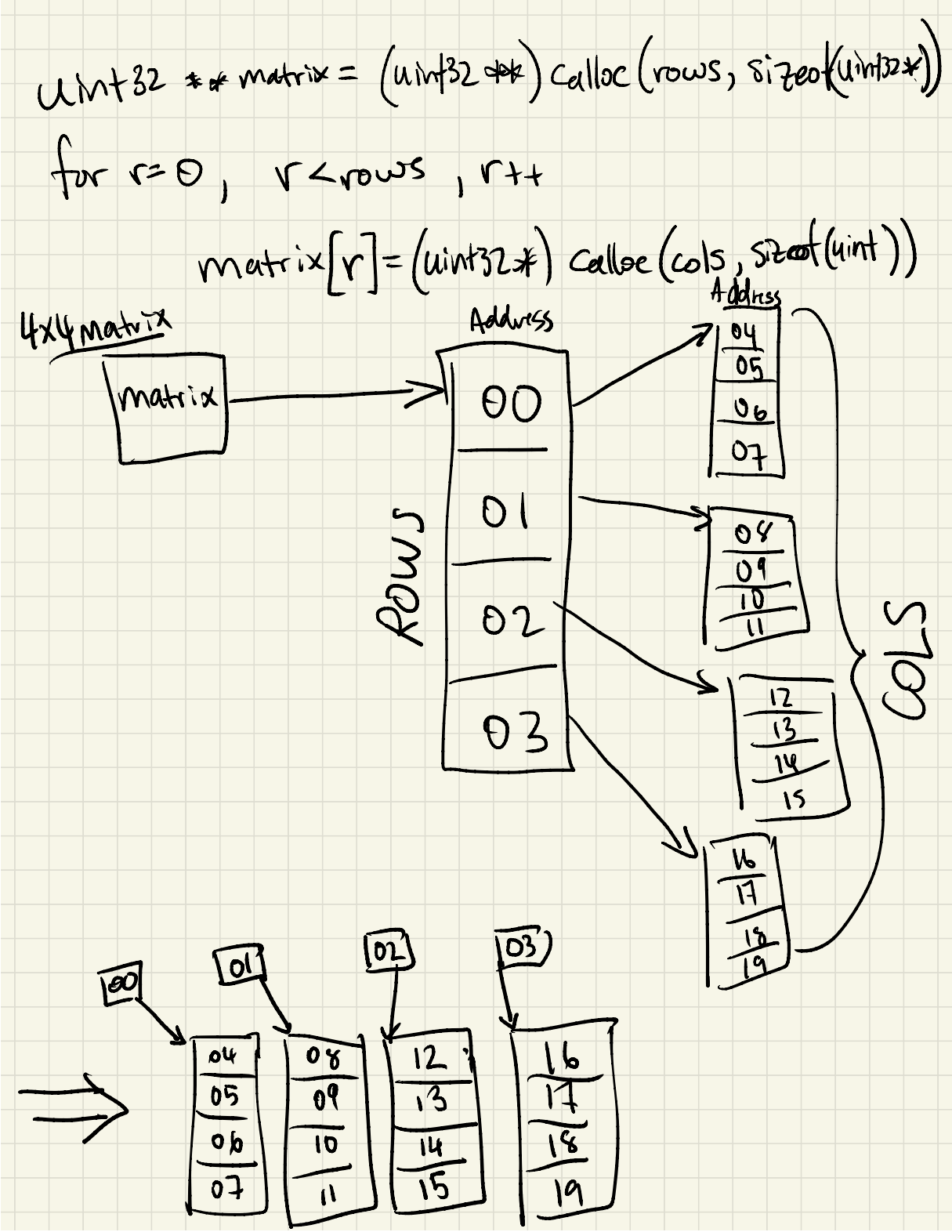
The Game of Life, created by John Horton Conway, is a game where, in theory, there is an infinite grid that starts with each cell in the grid either dead or alive. After each "generation", a cell continues living (assuming it's already alive) if it has 2 or 3 live neighbors or is resurrected (if it's dead) when it has exactly 3 neighbors. This process continues until the desired number of generations is reached.

Since implementing the game in `C`, I've learned quite a bit about dynamically allocating arrays, matrices, and user-created types (`Universe` in this case). Additionally, I learned about using `fscanf` and `fprintf` to with different file streams, using either an actual file *or* `stdin`/`stdout`. Also, I learned how to implement the opacity of an ADT as well as utilizing the `ncurses` library.

2 Implementing the Universe

After taking CSE12, I had some understanding of pointers and some inclination of its importance in languages such as `C`; however, I didn't *really* see the connection between pointers and arrays (and ultimately matrices) until this assignment.

Understanding basically how pointers worked in a data stack and reading how they intertwine with arrays from *C: The Programming Language* , I knew that an array is ultimately a pointer to the first "index" of a stack of data. Extending this, the spec document provides an example of a 2D matrix/grid which was declared initially as a double pointer, `**grid`. I broke down the definition in the spec that dynamically allocated the matrix to grasp what was happening.



Above was my line of thinking for what a 2D grid looks like at the pointer/address level. As I eluded to before, `matrix` is a pointer to an array of pointers. I expressed only the addresses

of each "block". The example I used was a 4x4 grid which consists of 4 rows and 4 columns. Thus, it's intuitive (from previous classes) that we first need to allocate the row pointers. Afterward, we have each "row" point to a new array of, in this case, unsigned integers. When we want to access values to either retrieve or change them, we need to first dereference the row, which then gives the address of that row's columns. Dereferencing again yields the column which we can then use to retrieve or change the value of.

So those are the details behind the `grid` in our `Universe` type. Moving on, the `Universe` type also contains the length of the grid's rows and columns, and whether the universe is *toroidal* or not. Although I had some experience with `typedef`s in `C` in preparation for this class, I definitely have a better understanding of how it works and when to apply it.

Harping back on dynamically allocated data, I also developed my understanding of `malloc` and `calloc`. It was already pretty clear to me how to use these two and how they basically work from the last assignment, but I learned that they're useful for allocating space for user-defined types. This is where `sizeof` came in handy. `sizeof` can also be used for user-defined types, such as `Universe`, as expressed in the spec.

I also learned the usefulness of `valgrind` which led me to see I had leaks despite `free()`ing all my Universe components. Though not a huge portion of my findings in this assignment, I saw that I need to also free the memory allocated for the file pointer with `fclose()`.

My final point for `Universe` is the implementation to make the specifics opaque to all other functions/files that call it.

3 Using Universe to Make Life

The bulk of the actual implementation of Game of Life is in `life.c`. This included taking either a file or input from `stdin`. This problem was new to me because I understand how to use `scanf` and `printf` however, the `fprintf` versions didn't make sense to me before. After reading the `man` pages (exceedingly helpful lol), I understand it now. The `fscanf` function takes a file stream as the first argument (a file pointer). This could be a pointer to some file accessed with `fopen()` *or* `stdin`. This was very helpful when handling option arguments to decide whether to take `stdin` or an input file. Likewise for `fprintf`.

As part of the spec, we need to implement the `ncurses` library to show each generation and the evolution by displaying the current state every 50,000 μs. Implementing it was very easy but I did learn the basics from the spec.

4 Wrap-Up

In essence, I take away from this project a stronger understanding of pointers and arrays, their use in forming matrices, and implementing opaque data structures.

5 Appendix

- toroidal

  Toroidal, in this context, means our 2D "Universe" behaves as though it was a torus. This is an important distinction compared to a "normal" Universe because the surface of a torus is

continuous, so when cut and laid out flat the edges aren't where the grid technically ends. If the Universe were still a fully-formed torus, the edge would just continue around to the other side. Thus, when describing a toroidal grid, going off any edge means wrapping around to the opposite side.