Systems Programming Coursework: Conway's Game of Life

Konrad Dabrowski

Deadline: 2p.m. Friday 28th February

Conway's Game of Life is a cellular automaton invented by John Conway in 1970. The game world consists of an infinite two-dimensional grid of cells, each of which is either "dead" or "alive" in each "generation." The game comes with a "rule" that explains how to calculate the state of a cell in the next generation. To find ("evolve") the next generation, each cell looks at the eight cells next to it (vertically, horizontally or diagonally), checks which of these are dead or alive in the current generation and then uses this rule to decide whether the cell should be alive in the next generation. There are various rules one can use to evolve the next generation. Conway's rule is as follows:

1. If a cell is alive in the current generation and exactly two or three of its neighbours are alive, then it will be alive in the next generation.
2. If a cell is dead in the current generation and exactly three of its neighbours are alive, then it will be alive in the next generation.
3. Otherwise, the cell will be dead in the next generation.

In this coursework, you will implement the Game of Life in C. Since you cannot use infinite memory, you will implement it on a finite rectangular grid (the "universe"). One approach to do this is to make cells on the border outside the grid stay permanently dead. Another approach is to give the grid the topology of a torus: the grid wraps around, so if you go one row higher than the top row, you reach the bottom row, and if you go one column to the left of the leftmost column, you reach the rightmost column etc. You will implement both of these.

To do this, you will construct a dynamically-linked library `libgol.so` which implements the Game of Life, and a program `gameoflife` that calls the library for all its key functionality. Your code should on Linux. **In particular, you should compile & test it on `mira`** (see Practical 1 for instructions on how to access mira.) You may not use any external libraries and only use the header file `gol.h` (provided) and those in the C standard.

The file format for *valid* input and output files: each line represents one row of the grid and each character represents one cell on that row: '`*`' represents a live cell and '`.`' represents a dead cell. For example:

```
.........
.........
.........
..***....
.........
.........
```

In the functions below, the rows are numbered $0, 1, 2, \ldots$ from top to bottom and the columns are numbered $0, 1, 2, \ldots$ from left to right. In any valid file, there will be at most 512 columns; the number of rows is not limited.

A: Dynamically-linked library (50%)

You have been provided with the file `gol.h`, whose contents are as follows:

```
struct universe {
/*Put some appropriate things here*/
};

/*Do not modify the next seven lines*/
void read_in_file(FILE *infile, struct universe *u);
void write_out_file(FILE *outfile, struct universe *u);
int is_alive(struct universe *u, int column, int row);
int will_be_alive(struct universe *u, int column, int row);
int will_be_alive_torus(struct universe *u,  int column, int row);
void evolve(struct universe *u, int (*rule)(struct universe *u, int column, int row));
void print_statistics(struct universe *u);
/*You can modify after this line again*/
```

In `gol.h`, devise an appropriate structure `struct universe` to hold the universe of cells and any data you need to keep track of; do not add additional functions to `gol.h`. [4 marks]

Create a file `gol.c` that implements the functions above:

1. `read_in_file()` reads in the file from the file pointer `infile` and stores the universe in the structure pointed to by `u`. You **must** use dynamic memory allocation to set aside memory for storing the universe. (Hint: how will you work out how many cells are in each row? You may want to `fscanf()` with `%c`. Recall that `fscanf()` returns `EOF` when the file ends. Press `Ctrl+D` to send `EOF` if typing input in.) [10 marks]
2. `write_out_file()` writes the content of the universe pointed to by `u` into the file from the file pointer `outfile`. [5 marks]
3. `is_alive()` returns `1` if the cell in that column and row is alive and `0` otherwise. [3 marks]
4. `will_be_alive()` returns `1` if the cell in that column and row will be alive in the next generation and `0` otherwise, assuming that cells outside the universe are always dead. [5 marks]
5. `will_be_alive_torus()` returns `1` if the cell in that column and row will be alive in the next generation and `0` otherwise, assuming a torus topology. [5 marks]
6. `evolve()` changes the universe from the current generation to the next generation. It uses the function pointed to by the function pointer `rule` to decide whether a cell will be alive or dead, e.g. `will_be_alive` or `will_be_alive_torus`. Note: a user of your library can provide their own rule. [4 marks]
7. `print_statistics()` should calculate the percentage of cells that are alive in the current generation and the average percentage that have been alive in all of the generations so far (including the original generation). Percentages should be given to three decimal places, and printed to the screen in the following format: [4 marks]

   ```
   32.000% of cells currently alive
   10.667% of cells alive on average
   ```

Your library should support having multiple universe structures in memory at a time. It must not use any global variables. Consider appropriate error checking and make sure that your library can be used robustly. In case of errors, print a message to `stderr` and exit the current program with a non-zero error code. [10 marks]

B: command-line program (30%)

Provide a command-line program `gameoflife` to use your library, with source code `gameoflife.c`. Make sure your program uses the functions in `gol.h` rather than manipulating universe structures directly. Support the following command-line switches:

- `-i input_filename` to specify that the initial generation of the universe should be read from a file. If this option is not specified, you should let the user type in the input. [4 marks]
- `-o output_filename` to specify a file into which the final generation should be output. If this option is not specified, you should output the final generation on the screen. [4 marks]
- `-g number_of_generations` to specify the number of new generations for which the game should be run. (Set to 5 if this option is not given.) [4 marks]
- `-s` to print statistics after the final generation has been output. [4 marks]
- `-t` to use the torus topology for the rule. If this is not specified, use the rule for cells outside the universe being permanently dead. [4 marks]

Command-line options can be given in any order. You must use a switch statement to help you parse them. As with the library, you should consider appropriate error checking and make sure your program can be used robustly. [10 marks]

A basic `gameoflife.c` file is provided, which ignores all command-line options. This can be used to help test your library.

C: Build system (20%)

Provide a `Makefile` to compile your library and your command-line program. In your Makefile, use the `-Wall -Wextra -pedantic -std=c11` options each time you call `gcc`. Your code should compile without any warnings or errors. Have rules for creating an object file from each `.c` file, for creating your dynamically-linked library `libgol.so` and for creating your command-line program `gameoflife`. Using the command `make` should compile your program to a file called `gameoflife` and your library `libgol.so`, both in the current folder. Running `make clean` should delete all files produced during compilation. [20 marks]

To Submit

Submit a zip file on DUO containing your files `gol.c`, `gol.h`, `gameoflife.c` and `Makefile` and no other files. You can check some of the basic functionality of your program by running the provided `test_script.sh` file.

Please note that submitted code will be checked for plagiarism.