Assignment 4 Design

Nhan Nguyen

January 2022

1 Introduction

1.1 Description

The purpose of this program is to simulate the Conway's Game of Life, note that simulation will be displayed using **ncurses** and the result after some number of generations will also be printed

1.2 List of Files

- **universe.h:** The interface to universe.c
- **universe.c:** The implementation of the Universe that stores the components for Conway's Game of Life
- **life.c:** This file contains the main() function as well as the necessary functions to simulate Conway's Game of Life on the **ncurses** screen
- **Makefile:** This file formats program into clang-format and also compiles it. In addition, this makefile will also link life.c and universe.c
- **README.md:** A text file in Markdown format that describes how to build the program, as well as describing the problems encountered while making the program
- **DESIGN.pdf:** This PDF file describes the idea behind the program with visualization and pseudocode

2 universe.c

2.1 Universe Struct

This universe.c file contains all the necessary components to make a Universe used for Conway's Game of Life, The **Universe** is a **struct** object that stores the following variables:

- **rows:** Number of Rows of the Universe

- **cols:** Number of Columns of the Universe
- ****grid:** Pointer that points to the 2D grid that stores all the cells, note that each cell in this grid will be a boolean that specify if the cell at that grid is dead or alive
- **toroidal:** A boolean that notifies if this grid is in toroidal form

Pseudocode:

```
Universe Structure:
    Unsigned 32 bit integer rows
    Unsigned 32 bit integer cols
    bool **grid
    bool toroidal
```

2.2 `uv_create()`

This is a Constructor that returns a pointer to a universe with the specified rows, columns, toroidal form or not. In this function, it's important to create a pointer to a Universe with the specified variables above, we can access and assign values to the variables of the Universe we just create by using the symbol "$->$". Note that when you create a new Universe object, it's important to allocate memory space that's equal to the size of Universe object to avoid segmentation error, we can allocate memory by using calloc() and malloc()
Pseudocode:

```
uv_create(rows, cols, toroidal):
    Universe *sub;
    dynamically allocate memory equal to size of Universe
    sub− >rows = rows
    sub− >cols = cols
    sub− >toroidal = toroidal
    sub− >grid = dynamically allocate memory the number of rows · size of bool
    for i from 0 to rows:
        sub− >grid[r] = dynamically allocate memory the number of cols · size of bool
    return sub
```

2.3 `uv_delete()`

This function frees any memory allocated from the function `uv_create()` of any Universe specified to prevent memory leaks. For this function, simply use the function **free()** will do the trick. Note that we need to free the following: the elements in the grid of the universe u, the grid itself, and then the universe u, since we have allocated memory for each of these components
Pseudocode:

```
uv_delete(pointer to universe u):
    for i from 0 to rows of universe u:
        free(u− >grid[i])
    free(u− >grid)
    free(u)
```

2.4 `uv_rows()` and `uv_cols()`

These 2 function will be the accessor methods to get the rows and columns of any universe specified in the paramenter. As mentioned in the `uv_create()` section, simply use the symbol ”− >” to access the variables of the Universe

Pseudocode:

```
uv_rows(pointer to universe u)
    return u− >rows
uv_cols(pointer to universe u)
    return u− >cols
```

2.5 `uv_live_cell()` and `uv_dead_cell()`

These 2 functions will make any cell in the grid either dead or alive. One thing to note is that these 2 functions will do nothing if the coordinates specified are out of the grid's bounds, to do this we will need to check if the rows/cols coordinates are either < 0 (which doesn't happen because we're using unsigned integers) or $\geq$ **number of rows of the Universe** or $\geq$ **number of cols of the Universe**. To make the cell at the coordinate dead or alive, simply assigned the boolean value false or true respectively to the coordinate.

Pseudocode:

```
uv_live_cell(pointer to universe u, r, c)
    if r $\geq$ number of rows of the Universe u: return
    if c $\geq$ number of rows of the Universe u: return
    u− >grid[r][c] = true
uv_dead_cell(pointer to universe u, r, c)
    if r $\geq$ number of rows of the Universe u: return
    if c $\geq$ number of rows of the Universe u: return
    u− >grid[r][c] = false
```

2.6 `uv_get_cell()`

This function returns the state of the cell at the coordinate and universe specified in the parameter, this function will automatically return false if the coordinate is out of the grid's boundaries, as stated in previous sections we can access the variables and even elements of the grid using the symbol ”− >”

Pseudocode:

```
uv_get_cell(pointer to universe u, r, c)
    if r < 0 or r ≥ number of rows of the Universe u: return false
    if c < 0 or c ≥ number of rows of the Universe u: return false
    return u− >grid[r][c]
```

2.7 `uv_populate()`

This function will configure all the cells specified in the input to be alive and returns true if all the cells are configure correctly and false otherwise. Since the first line of the input (rows and columns of the grid) is already read in life.c this function only reads the 2 lines onward. We will be reading the variables using the fscanf() function and read until we reach the end of the file (EOF) since there's no line that specifies how many inputs there are. As for checking if the cells are configured correctly simply use `uv_get_cell()` after you assign the alive cell using `uv_live_cell()`, `uv_get_cell()` will always return true, and only return false if the coordinate is out of bounds, then we get a false from `uv_get_cell()` we should return false immediately
Pseudocode:

```
uv_populate(pointer to universe u, file for input)
    num1 = 0
    num2 = 0
    while (read lines of the file and assign to num1 and num2) != EOF:
        if doesn't read properly: return false
        uv_live_cell(u, r, c)
        if uv_get_cell(u, r, c) == false: return false
    return true
```

2.8 `uv_census()`

This functions counts the number of neighbors in all 8 directions of a cell, if the universe is toroidal, we have to check the opposite side of the grid if the direction we're going will result in an out of bounds.
If the Universe is not toroidal, we simply use if statements to check 2 things: if the direction we're going will result in an out of bounds error, and is the cell in the direction we're going alive. If the conditions are met we increment a **a counting variable that starts from 0**, do this for 8 directions
If the Universe is toroidal, we will not go out of bounds because we will be circling back to opposite side of the grid, to simulate this we could do if statements, for my code I used the following situations to develop my code:

- if either **r + 1** or **c + 1** exceeds the rows or columns respectively: In this case simply take **r + 1** or **c + 1** and modulo with rows or columns respectively to circle back to the left or upward side of the grid

- if either **r - 1** or **c - 1** goes below 0, we need dedicated if statements to make the function. If the r or c is 0, we need to use the value rows - 1 or cols - 1 respectively to circle to the right/downward side of the grid

Pseudocode:

```
uv_census(pointer to universe u, r, c)
    cnt = 0
    if u−>toroidal == false:
        if r < rows - 1 and uv_get_cell(u, r + 1, c) == true: cnt++
        if r > 0 and uv_get_cell(u, r - 1, c) == true: cnt++
        if c < cols - 1 and uv_get_cell(u, r, c + 1) == true: cnt++
        if c > 0 and uv_get_cell(u, r, c - 1) == true: cnt++
        if r > 0 and c > 0 and uv_get_cell(u, r - 1, c - 1) == true: cnt++
        if r < rows - 1 and c < cols - 1 and uv_get_cell(u, r + 1, c + 1) ==
true: cnt++
        if r > 0 and c < cols - 1 and uv_get_cell(u, r - 1, c + 1) == true:
cnt++
        if r < rows - 1 and c > 0 and uv_get_cell(u, r + 1, c - 1) == true:
cnt++
    else:
        cnt += uv_get_cell(u, (r + 1) mod rows, c)
        cnt += uv_get_cell(u, r, (c + 1) mod cols)
        cnt += uv_get_cell(u, (r + 1) mod rows, (c + 1) mod cols)
        if r == 0:
            cnt += uv_get_cell(u, rows - 1, (c + 1) mod cols)
            cnt += uv_get_cell(u, rows - 1, c)
        else if r > 0:
            cnt += uv_get_cell(u, r - 1, (c + 1) mod cols)
            cnt += uv_get_cell(u, r - 1, c)
        if c == 0:
            cnt += uv_get_cell(u, (r + 1) mod rows, cols - 1)
            cnt += uv_get_cell(u, r, cols - 1)
        else if c > 0:
            cnt += uv_get_cell(u, (r + 1) mod rows, c - 1)
            cnt += uv_get_cell(u, r, c - 1)
        if r == 0 and c == 0:
            cnt += uv_get_cell(u, rows - 1, cols - 1)
        if r == 0 and c > 0:
            cnt += uv_get_cell(u, rows - 1, c - 1)
        if r > 0 and c == 0:
            cnt += uv_get_cell(u, r - 1, cols - 1)
        else if r > 0 and c > 0
            cnt += uv_get_cell(u, r - 1, c - 1)
    return cnt
```

2.9 uv_print()

This function prints out the grid from the universe, if the cell is alive the function prints "o" and "." if the cell is dead. This function is fairly straightforward since the function mainly use for loops, when print out the result, use fprintf() since you need to print the results on a specific file and `uv_get_cell()` to see which cell is dead or alive

Pseudocode:

```
uv_print(pointer to universe u, file for output)
    for i from 0 to rows:
        for j from 0 to cols:
            if uv_get_cell(u, i, j):
                print "o " to file
            else:
                print ". " to file
        print "\n" to file
```

3 life.c

This file contains the main() function and will utilized the functions implemented in universe.c to simulate the entire Conway's game of life

3.1 Command

The commands this program accepts are:

- **-t:** Specify that the game will be played on a toroidal universe
- **-s:** Specify that the game will be played without the screen
- **-n generations:** Specify the number of generations the game will be running (Default: 100)
- **-i input:** Specify the file to read in for the input (Default: stdin)
- **-o output:** Specify the file to print for the output (Default: stdout)

Pseudocode:

```
toroidal = false
display = true
gens = 100
input = stdin
output = stdout

get command
if command is t:
    toroidal = true
```

```
if command is s:
    display = false
if command is n:
    read **number**
    gens = **number**
if command is i:
    read **string**
    open file with name **string**
if command is o:
    read **string**
    open file with name **string**
```

3.2 Display and Processes

For the display we will be using ncurses to simulate the Game of Life, this involves printing the current iteration of the Universe, update the iteration to a new state then print that new state, this cycle will repeat a number of times specified earlier in the **n** command. For this entire operation we will be using almost all of the functions we implemented in universe.c, ncurses functions to implement a screen, some if statements to simulate the game
Note 1: Keep in mind that I'll not be discussing the rules of Conway's game of life because it's already explained in details in the doc
Note 2: The Pseudocode below will have the variables implemented in the previous section since these 2 portions are 1 complete program

Pseudocode:

```
rows = 0
cols = 0
read the first line of the input file and assign values to rows and cols
if didn't read prpoerly:
    display error message
    exit the program
pointer to universe a = uv_create(rows, cols, toroidal)
pointer to universe b = uv_create(rows, cols, toroidal)
open = uv_populate(a, input)
if open == false:
    display error message
    exit the program
for i from 0 to gens:
    if display == true:
        open display
        disable cursor
        clear the screen
        for ii from 0 to uv_rows(a)
            for jj from 0 to uv_cols(a)
```

```
                if uv_get_cell(a, ii, jj) == true:
                    print ”o” to display
                else:
                    print ”.” to display
                print ” ” to display
            print ”\n” to display
        refresh the screen
        sleep for 50000 microseconds

    for ii from 0 to uv_rows(a)
        for jj from 0 to uv_cols(a)
            if uv_get_cell(a, ii, jj) == true and uv_census(a, ii, jj) == 2 or 3:
                uv_live_cell(a, ii, jj)
            if uv_get_cell(a, ii, jj) == false and uv_census(a, ii, jj) == 3:
                uv_live_cell(a, ii, jj)
            else:
                uv_dead_cell(a, ii, jj)
    swap pointer to universe a with pointer to universe b
if display == true:
    close the display
uv_print(a, output file)
uv_delete(a)
uv_delete(b)
close input file
close output file
```