Design Document - Assignment 4

Hsiang Yun Lu

February 13, 2023

1 Purpose of Programs

The `life.c` program implements the Game of Life evolution based on input files or STDIN and shows the animation of the evolution of the universe. The `universe.c` program includes the structure of `Universe` and the accessing and manipulating functions that will be used in `life.c`.

2 Files To Be Included in Directory *asgn3*

- `universe.c`
  - This is the implementation of the `Universe` ADT, including type construction and associated functions.
- `universe.h`
  - This file specifies the interface to the `Universe` ADT, including declaration of new `type` and `universe`.
- `life.c`
  - This program contains `main()` and can read in inputs to create the universe. It's the main implementation of the Game of Life.
- `Makefile`
  - This file compiles the sorting programs and builds the `life` executable
- `README.md`
  - This file describes how to use all the programs and `Makefile`, including explanations on command-line options
- `DESIGN.pdf`

  - This file describes the design and design process for all the programs
  - Include pseudocode

- `WRITEUP.pdf`
  - Things learned in this assignment in detail

3 Design Process of `universe.c`

The `universe.c` includes the detailed of `Universe` struct and all the helper, accessor, constructor, destructor and manipulator functions for `Universe`. All of the "uv-" prefixed functions were implemented according to the descriptions in the assignment PDF. I added a `in_bound` function to check whether the given row/column pairs are out-of-bounds and a `swap` function to help swap the pointers of two `Universe`. The other helper function is `one_generation`, which performs one generation of the cell evolution according to the given rules. All these functions are used in `life.c`.

4 Design/Structure of `universe.c`

DEFINE Universe instance ← [rows, cols, **grid, toroidal]

function int in_bound(uint32_t rows, uint32_t cols, uint32_t r, uint32_t c)
    IF (r+1) <= rows and (c+1) <= cols
        RETURN 1
    ELSE
        RETURN 0
    END IF
END function

function void swap(Universe *x, Universe *y)
    tmp ← *x
    *x ← *y
    *y ← tmp
END function

function int32_t mod(int32_t a, int32_t b)
    n ← a % b
    RETURN (n + b) if n < 0 else RETURN n
END function

```
function Universe *uv_create(uint32_t rows, uint32_t cols, bool toroidal)
    SET Universe *u
    u ← calloc(1, sizeof(Universe))
    SET **grid ← calloc(rows, sizeof(bool pointer))
    FOR r = 0 to rows
        grid[r] ← calloc(cols, sizeof(bool))
    END FOR
    u->rows ← rows
    u->cols ← cols
    u->grid ← grid
    u->toroidal ← toroidal
    RETURN universe
END function


function void uv_delete(Universe *u)
    FOR i = 0 to u->rows
        free(u->grid[i])
    free(u->grid)
    free(u)
END function


function uint32_t uv_rows(Universe *u)
    RETURN u->rows
END function


function uint32_t uv_cols(Universe *u)
    RETURN u->cols
END function


function void uv_live_cell(Universe *u, uint32_t r, uint32_t c)
    IF (r+1) <= u->rows and (c+1) <= u->cols
        u->grid[r][c] ← TRUE
    END IF
END function


function void uv_dead_cell(Universe *u, uint32_t r, uint32_t c)
```

```
    IF (r+1) <= u->rows and (c+1) <= u->cols
        u->grid[r][c] ← FALSE
    END IF
END function


function bool uv_get_cell(Universe *u, uint32_t r, uint32_t c)
    IF (r+1) <= u->rows and (c+1) <= u->cols
        RETURN u->grid[r][c]
    ELSE
        RETURN FALSE
    END IF
END function


function bool uv_populate(Universe *u, FILE *infile)
    SET r ← 0
    SET c ← 0
    WHILE TRUE
        SET input ← fscanf(infile, format read in 2 uint32_t, r, c)
        IF input == 2 and in_bound(u->rows, u->cols, r, c) == 1
            SET u->grid[r][c] ← TRUE
        ELSE IF in_bound(u->rows, u->cols, r, c) == 0
            RETURN FALSE
        ELSE IF read till the end of file
            RETURN TRUE
            break
        ELSE IF input not equal 2 and input not equal end of file
            RETURN FALSE
        END IF
    END WHILE
END function


function uint32_t uv_census(Universe *u, uint32_t r, uint32_t c)
    IF u->toroidal == false
        SET live ← 0
    r_int ← (int32_t) r
    c_int ← (int32_t) c
        FOR i = r_int-1 to r_int+1
            FOR j = c_int-1 to c_int+1
                IF (i+1) <= u->rows and (j+1) <= u->cols and i >= 0 and j >= 0 and u->grid[i][j] ==
```

```
TRUE
                    live ← live + 1
                END IF
            END FOR
        END FOR
        IF u->grid[r][c] == TRUE
            RETURN live - 1
        ELSE
            RETURN live
        END IF
    ELSE
        SET live ← 0
    r_int ← (int32_t) r
    c_int ← (int32_t) c
        FOR i = r_int-1 to r_int+1
            FOR j = c_int-1 to c_int+1
                IF (i+1) > u->rows or i < 0 or (j + 1) > u->cols or j < 0
                    IF u->grid[mod(i, u->rows)][mod(j, u->cols)] == TRUE
                        live ← live + 1
                    END IF
                ELSE
                    IF u->grid[i][j] == TRUE
                        live ← live + 1
                    END IF
                END IF
            END FOR
        END FOR
        IF u->grid[r][c] == TRUE
            RETURN live - 1
        ELSE
            RETURN live
        END IF
    END IF
END function


function void uv_print(Universe *u, FILE *outfile)
    FOR i = 0 to u->rows
        FOR j = 0 to u->cols
            IF (j + 1) == u->cols
                IF u->grid[i][j] == TRUE
                    PRINT “o” to outfile
```

```
            ELSE
                PRINT "." to outfile
            END IF
        ELSE
            IF u->grid[i][j] == TRUE
                PRINT "o" to outfile
            ELSE
                PRINT "." to outfile
            END IF
        END IF
    END FOR
END FOR
END function


function void one_generation(Universe *u_A, Universe *u_B, uint32_t r, uint32_t c)
    FOR i = 0 to r
        FOR j = 0 to c
            IF uv_get_cell(u_A, i, j) == TRUE and uv_census(u_A, i, j) == 2
                uv_live_cell(u_B, i, j)
            ELSE IF uv_get_cell(u_A, i, j) == TRUE and uv_census(u_A, i, j) == 3
                uv_live_cell(u_B, i, j)
            ELSE IF uv_get_cell(u_A, i, j) == FALSE and uv_census(u_A, i, j) == 3
                uv_live_cell(u_B, i, j)
            ELSE
                uv_dead_cell(u_B, i, j)
            END IF
        END FOR
    END FOR
END function
```

5 Design Process of `life.c`

This program first reads the command-line options and the input files or STDIN, creates and populates two `Universe`, performs some generation of cell evolution, displays the evolution animation on screen, and finally outputs the final state of the `Universe` to a file or STDOUT. I first declared the helper functions I wrote in `universe.c` in order to use them in `life.c`. Then I wrote a function for showing helping information before `main()`.

In `main()`, I declared variables for storing command-line arguments and input/output file

names. After reading the command-line arguments, the first line of the input was read and the numbers were stored for creating `Universe`. The input data were used to create and populate two `Universe` using functions in `universe.c`. The `FILE` pointer for input stream should be closed after the two `Universe` were successfully populated. Based on the command-line arguments, for each generation display `Universe A`, perform one generation and then swap the pointers of two `Universe` if "-s" was not specified. After displaying animation, print out the final state of the `Universe`. If "-s" was specified, print out the final state of the `Universe` without animation. Finally, I deleted both `Universe` after printing successfully.

6 Design/Structure of `life.c`

DEFINE command-line options "tsn:i:o:h"

function usage(program)
    print(program synopsis and usage)
END function

function main(arguments)
    SET opt ← 0
    SET torus ← FALSE
    SET silence ← FALSE
    SET num ← 100
    SET inp ← FALSE
    SET out ← FALSE
    SET char *input
    SET char *output
    SET FILE *finptr
    SET FILE *foutptr
    SET r ← 0
    SET c ← 0

    WHILE opt ← here is at least an argument
        SWITCH(opt)
            case argument "t" SET torus ← TRUE
                break
            case argument "s" SET silence ← TRUE
                break
            case argument "n" SET num ← number converted from read-in string
                break
            case argument "i"
                inp ← TRUE

```
            input ← optarg
            break
        case argument “o”
            out ← TRUE
            output ← optarg
            break
        case argument “h” do usage(argv[0])
            EXIT
        default do usage(argv[0])
            return EXIT_FAILURE
    END SWITCH
END WHILE

IF inp == FALSE
    fscanf(stdin, format read in 2 uint32_t, &r, &c)
ELSE
    IF (finptr ← fopen(input, “r”)) does not exist
        PRINT ERROR
        EXIT
    ELSE
        fscanf(finptr, format read in 2 uint32_t, &r, &c)
    END IF
END IF

SET Universe *uni_A ← uv_create(r, c, torus)
SET Universe *uni_A ← uv_create(r, c, torus)

IF inp == FALSE
    populated = uv_populate(uni_A, stdin)
    IF populated == FALSE
        PRINT ERROR
        EXIT
    END IF
ELSE
    populated = uv_populate(uni_A, finptr)
    IF populated == FALSE
        PRINT ERROR
        EXIT
    END IF
END IF

CLOSE finptr
```

```
SET r_A ← uv_rows(uni_A)
SET c_A ← uv_cols(uni_A)

IF silence == FALSE
    initscr()
    curs_set(FALSE)
    FOR k = 0 to num-1
        clear()
        FOR i = 0 to r_A-1
            FOR j = 0 to c_A-1
                IF uv_get_cell(uni_A, i, j) == TRUE
                    PRINT “o” at location (i, j) on screen
                ELSE
                    PRINT “.” at location (i, j) on screen
                END IF
            END FOR
        END FOR
        refresh()
        usleep(50000)
        one_generation(uni_A, uni_B, r_A, c_A)
        swap(uni_A, uni_B)
    END FOR
    endwin()
ELSE
    FOR k = 0 to num-1
        one_generation(uni_A, uni_B, r_A, c_A)
        swap(uni_A, uni_B)
    END FOR
END IF

IF out == FALSE
    uv_print(uni_A, stdout)
ELSE
    foutptr ← fopen(output, “w”))
    uv_print(uni_A, foutptr)
END IF

uv_delete(uni    _A)
uv_delete(uni    _B)

RETURN 0
```

End function

7 Credit

- Dev's section on Feb. 7th.
- The *readfile_example.c* in resources
- Modular arithmetic concepts from the discussion