Assignment 4

The Game of Life

DESIGN.pdf

Reuben T. Chavez

February 2, 2022

Files in asgn4

Brief Description of the following files are as followed:

- **life.c**: Program that contains main() and when called implements the Game of Life
- **universe.c**: Implements the Universe ADT
- **universe.h**: Specifies the interface to the Universe ADT
- **Makefile**: File that stores commands to complie life.c be an executable programs , format the all the .c and .h files in clang format, and removes any unnecessary binary files before the program, through the command make clean
- **README.md**: Marks Down File that briefly describes how the program can be run and built on the terminal, which command-line options the program accepts, and error handling on the program
- **DESIGN.pdf**: A .pdf file that covers the purpose the program, has a lay out and structure of the program, with clear description/explanation on how each part works

Pseudeocode

```
#Libaries
import universe
import stdlib
import bool
import stdint

# Psuedocode for life.c

## Usage ##
function usage is
    input: executable
    output: void

    print(
    "Synopsis of Life\n"

    "Usage of Life\n"

    "Options for Life\n"
    )

function main is
    input : argument count argc and argument vector argv
    output: zero to exit program

    intialize opt to 0
    initialize toroidal to false
    initialize silence to false
    initilaize generations to 100
    initialize input to stdin
    initilize output to stdout

    while getting commands from command line do
        switch command:
            case toroidal:
                set toroidal to true
            case silence :
                set to true
            case generations :
                set generations equal to user's input
            case input:
```

```
      set input to open and read file from user's input
    case output:
      set output to open and write file from user's input
    default help:
      prints usage and ends program

Intialize row and cols as unsigned integers
Initialize Universe A and Universe B

Scan first line of input file to get rwo and column size for universe

Create Universe A and B with the rows and columns given, alogn with toroidal

if the input file is able to populate Universe A:

  for number of generations do;
    if silence is false:
      Initialize Screen
      Hide the Cursor
      Clear the window
      for range of Universe A rows do;
        for the range of Universe A columns do;
          if specified cordinate i true then
            Print in coordinate

    Refresh window
    Sleep for 50000 microseconds

    for number of rows in Universe A do;
      for number of columns in Universe A do;
        if cell in grid is alive and has 2 or 3 neighbors
          cell stays alive
        else if cell in grid is dead and has 3 exact neighbors
          cell is alive
        else
          cell is dead from overcrowding or loneliness
  Swap Universe A and Universe B

else:
  print error that the input file had a coordinate that is out of bounds
```

```
    Delete Universe A and B
    Close both the input and output files

    return 0
```

The process for life.c is as follows. First, the initial libraries are used as a reference to use the universe ADT file, assigning unsigned 32-bit integers throughout the program as return types and parameters, and using the bool variable true and false for more straightforward implementation of the true and false statement.

The usage function is only a simple void function. The function displays the synopsis of life.c, the various parameter that the program takes as input and a brief description of said parameter. Whenever the user calls the -H flag on the command line Terminal, it is only called.

The main function works as follows.

First, initialize various variables. These variables work for the purpose of having the default conditions specified in the assigment. They include the following: opt is equal to zero(to check command-line options), toroidal is equal to false(to initialize Universe variables as planes), silence equal to false(to display changes within the universe over time), generations equal to 100(number of iteration to change Universe type), input(input file to be used as a reference for all Universe variables), and output(where the final form of the Universe will be printed onto).

Next is to get the user's input, which is accomplished similarly to how the other assignments use a while loop, getop, and a switch case. The command lines when executing the file are read using getopt. Depending on which one is called, the initialized variables from before are changed. In the case of generations, input, and output, they are adjusted to the user's specifications and use optarg to get the exact argument.

After getting the user's input, the Universe objects are initialized. After scanning the first line of the input file, the Universe objects are set to equal uvcreate, from universe.h, with the specified rows, columns, and toroidal boolean.

An if statement is used to implement the function uvpopulate, where it checks if the coordinates in the rest of the diagram are within the bounds of the graph in the Universe object. If true, then the Universe object has been parsed with all the coordinates from the given input file. Else it will print an error message that there was a coordinate in the input file ut of bounds and end the program.

Back to the case in where it successfully populates the Universe object, for a said number of generations either given or with the default 100, it will either display the current universe, or it won't, depending on if -s is called on the command line. Then it will change the second Universe, Universe B, by checking every coordinate in Universe A and depending on the number of neighbors the specified cell has it will apply the rule of Conway's Game of Life.

After completing the number of generations, the final outcome of the graph will be printed onto the specified output file, both Universes will be deleted with undelete, and the input and output files will be closed with fclose() to prevent memory leaks as the program ends.

```
#Pseudocode for universe.c

#Libaries
import universe
import bool
import inttypes
import stdlib


Initialize an enum called direction containing all 8 directions

Initailze stuct called Universe:
    with varible to store number of rows
    with varible to store number of columns
    with bool double pointer create grid
    with bool toroidal

function uvcreate
    input: total num of rows and cols to create and bool torodial
    output: universe pointer

    set a universer pointer to malloc to allocate memory in heap
        for Universe u
    set Universe u's toroidal to the torodial input
    set Universe u's rows to row input
    set Universe u's columns to cols input
    set the grid to calloc to allocate memory for the grid

    for the size of rows do
        set grid[rows] to calloc
        set grid row to false

    return u
```

```
function uvdelete
    input: Universe pointers
    output: void
    for the size of universe u's row do
        free the allocated memory in the grid[current row]
    free allocated memory og grid
    free the inputed universe u

    return

function uvrows:
    input: Universe pointers
    output: unsiged 32 integer

    return Universe u's total rows

function uvcols:
    input: Universe pointers
    output: unsiged 32 integer

    return Universe u's total rows

function uvlivecell
    input: Universe pointer and two unsigned 32 intergers for both
        the row and column
    output: void

    set specified point in grid to true

function uvlivecell
    input: Universe pointer and two unsigned 32 intergers for both
        the row and column
    output: void

    set specified point in grid to true
    return

function uvdeadcell
    input: Universe pointer and two unsigned 32 intergers for both
        the row and column
    output: void

    set specified point in grid to false
    return
```

```
function getcell
    input: univerese pointer and two unsigned 32 intergers for both
        the row and column
    output: boolean

    if eithier row or column is out of bounds
        return false

    return boolean from grid specfied row and column

function uvpopulate
    input: Universe pointer and FILE infile
    output: boolean

    initalize two unsigned 32 intergers for both the row and column
        of the grid

    scan the infile and set the two values to the initialized row
        and columns

    create universe u that hs rows and colums size from scan

    while file is not end of file:
       initailze two diffrent unsigned 32 bit inters for the rest of
           the scan
       scan rest of file
       if coordinate is out of bounds
        return false
      turn specfied cell using uvlivecell
    return true

function uvcensus
    input: Universe pointer and two unsigned 32 intergers for both
        the row and column
    output: unsiged 32 bit interger

    initailize niegbors to zero

    for items in the enum DIRECTIONS do
        initalize bool varibles named plane bounds and toroidal
            bounds
        inintilize unigned 32 bits tr and tc

        switch(direction)
```

```
case top:
    set tr to modluar of rows
    set tc to c + 1 modular of colums
    set plane bounds to top of specfied cell
    set tordial bounds to if Universe is tordial and call
        uvget cell with tr and tc
    if eithier plane bounds or tordial bounds is true
        increment neighbor
case bottom:
    set tr to modluar of rows
    set tc to c - 1 modular of colums
    set plane bounds to bottom of specfied cell
    set tordial bounds to if Universe is tordial and call
        uvget cell with tr and tc
    if eithier plane bounds or tordial bounds is true
        increment neighbor
case left:
    set tr r - 1 to modluar of rows
    set tc to modular of colums
    set plane bounds to left of specfied cell
    set tordial bounds to if Universe is tordial and cale
        uvget cell with tr and tc
    if eithier plane bounds or tordial bounds is true
        increment neighbor
case right:
    set tr r + 1 to modluar of rows
    set tc to modular of colums
    set plane bounds to right of specfied cell
    set tordial bounds to if Universe is tordial and cale
        uvget cell with tr and tc
    if eithier plane bounds or tordial bounds is true
        increment neighbor
case topleft:
    set tr to r -1 modluar of rows
    set tc to c + 1 modular of colums
    set plane bounds to topleft of specfied cell
    set tordial bounds to if Universe is tordial and cale
        uvget cell with tr and tc
    if eithier plane bounds or tordial bounds is true
        increment neighbor
case topright:
    set tr to r + 1 modluar of rows
    set tc to c + 1 modular of colums
    set plane bounds to topright of specfied cell
    set tordial bounds to if Universe is tordial and cale
```

```
                  uvget cell with tr and tc
              if eithier plane bounds or tordial bounds is true
                  increment neighbor
           case bottomleft:
              set tr to r - 1 modluar of rows
              set tc to c - 1 modular of colums
              set plane bounds to bottomleft of specfied cell
              set tordial bounds to if Universe is tordial and cale
                  uvget cell with tr and tc
              if eithier plane bounds or tordial bounds is true
                  increment neighbor
           case bottomright:
              set tr to r + 1 modluar of rows
              set tc to c - 1 modular of colums
              set plane bounds to bottomright of specfied cell
              set tordial bounds to if Universe is tordial and cale
                  uvget cell with tr and tc
              if eithier plane bounds or tordial bounds is true
                  increment neighbor
      return neighbors

function uvprint
    input: Universe pointer and FILE outfile
    output: void

    for size of row do;
       for size of colums do ;
          if  call uvgetcell is true
              print onto outfile O
          else
              print onto outfile .
```