Viveka Agrawal
Professor Veenstra
CSE13S, Winter 2023

CSE13S ASGN4 DESIGN DOC

Description of the program:
This program simulates the Game of Life. Here are the 3 main rules of the game:

1. Any *live* cell with two or three live neighbors *survives.*
2. Any *dead* cell with exactly three live neighbors *becomes a live cell.*
3. All other cells die, either due to loneliness or overcrowding.

The functions will be coded in universe.c and life.c includes main.

In this assignment, these 3 files must be created and submitted on git. They include:

universe.c: implements the Universe ADT

universe.h: specifies the interface to the Universe ADT. This file is provided and may not be modified

life.c: contains main() and may contain any other functions necessary to complete your implementation of the Game of Life

There is one other header file I have created that I will also create and submit to git:

life.h: specifies the interface to the Universe Struct

A makefile, readme document, and writeup must also be completed for this assignment.

universe.c

Include universe.h, inttypes.h, stdbool.h, stdio.h, and stdlib.h

Create the struct Universe code which is provided by Professor Long. It declares 4 variables, 2 of them type uint32_t named rows and cols, and 2 of them type boolean. One has a pointer to a pointer named grid and the other is just a boolean named toroidal.

```
Universe Struct
    rows
    cols
    Pointer to a pointer grid
    toroidal
```

Create a constructor function that creates a Universe called uv create which is a pointer and takes in 3 parameters. The first 2 parameters are the rows and columns (both type uint32_t) which are used as the dimensions of the boolean grid. The 3rd parameter is a boolean variable named toroidal. If toroidal is true, then make the universe toroidal. The return type of the function is Universe * so it should return a pointer to a universe. Use calloc to dynamically allocate memory.

```
Pointer function of type Universe uv_create with parameters rows, cols, and
toroidal
    Dynamically allocate memory for a Universe
    Store the Universe Struct rows as rows
    Store the Universe Struct cols as cols
    Create a pointer to a pointer boolean matrix and dynamically allocate
        memory for the matrix
    Create a for loop to loop through the number of rows
        Dynamically allocate memory at the matrix index
    Store the Universe Struct grid as matrix
    Store the Universe Struct toroidal as toroidal
    Return the universe
```

Create a destructor function that destroys a Universe called uv destroy which is void and takes a pointer to a universe as a parameter. This function frees any memory allocated for a Universe by the constructor function.

```
Void function uv_delete with a Universe parameter
    Create a for loop to loop through the number of rows
        Free the dynamically allocated memory in the grid
     Free the dynamically allocated memory in the grid
     Free the universe
    return
```

Create a uint32_t function that returns the number of rows in the specified Universe called uv rows and takes a pointer to a universe as a parameter.

```
uint32_t function uv_rows with a Universe parameter
    Return the Universe Struct rows
```

Create a uint32_t function that returns the number of columns in the specified Universe called uv cols and takes a pointer to a universe as a parameter.

```
uint32_t function uv_cols with a Universe parameter
    Return the Universe Struct columns
```

Create a void function named uv live cell with parameters of a pointer to a universe, and 2 uint32_t variables called r and c representing rows and columns. This function marks the cell at row r and column c as live. If the row and column are out of bounds, nothing happens and true indicates live while false indicated dead.

```
Void function uv_live_cell with a Universe parameter, r, and c
    If r is less than the Universe Struct rows and c is less than the Universe
        Struct columns
        Set that row and column index on the grid as true
```

Create a void function named uv dead cell with parameters of a pointer to a universe, and 2 uint32_t variables called r and c representing rows and columns. This function marks the cell at row r and column c as dead. If the row and column are out of bounds, nothing happens and true indicates live while false indicated dead.

```
Void function uv_dead_cell with a Universe parameter, r, and c
    If r is less than the Universe Struct rows and c is less than the Universe
        Struct columns
        Set that row and column index on the grid as false
```

Create a boolean function named uv get cell with parameters of a pointer to a universe, and 2 uint32_t variables called r and c representing rows and columns. This function returns the value of the cell at the specified row and column. If the row and column are out of bounds, then false is returned.

```
Boolean function uv_get_cell with a Universe parameter, r, and c
    If r is less than the Universe Struct rows and c is less than the Universe
        Struct columns
        Return the row and column index from the grid
    Else
        Return false
```

Create a boolean function named uv populate with parameters of a pointer to a universe, and a pointer to an infile. This function populates the Universe with row-column pairs which are read from the infile. The first line of the infile is the number of rows followed by the number of columns of the Universe. Every other line is the row and column pair of a live cell. If pair is out of bounds, return false that the universe failed to be populated and true if it did. If the Universe is failed to be populated, the game should fail with an error message.

```
Boolean function uv_populate with a Universe parameter and infile
    Declare uint32_t variables r and c
    // populating the universe with dead cells
    Create a for loop to loop through the number of rows (i)
        Create a for loop to loop through the number of columns (j)
            Call uv_dead_cell with a Universe parameter, i, and j

    // populating the live cells in the universe
    While loop to check whether the infile has not reached the end of the file
        from the number of rows (r) and number of columns (c)
        If r is less than the Universe Struct rows and c is less than the
            Universe Struct columns
            Call uv_live_cell with a Universe parameter, r, and c
        Else
            Return false
    Return true
```

Create a uint32_t function named uv census with parameters of a pointer to a universe, and 2 uint32_t variables called r and c representing rows and columns. This function returns the number of live neighbors adjacent to cell at row r and column c. If the Universe is non-toroidal, only consider valid neighbors for the count. If the universe is toroidal, then all neighbors all valid (wrap-around to the other side of the universe).

```
uint32_t function uv_census with a Universe parameter, r, and c
    Initialize uint32_t variable numlivecells to track number of live cells to 0
    Initialize uint32_t variable rtemp to r
    Initialize uint32_t variable ctemp to c

    // check for the neighbors on the right
    Increment ctemp
    If the universe is toroidal
        If the current column index is at the right edge of the universe, then
        wrap around to column index 0 in the toroidal universe using modular
        arithmetic
    If the cell at the specified row and index is true
        Increment the number of live cells
```

```
// check for neighbors on the left
Reset rtemp and ctemp to r and c respectively
If the column is at index 0
    If the universe is toroidal
        Set ctemp equal to the last column value (wrap around)
        If the specified cell is a live cell
            Increment the number of live cells
Else if the column is greater than index 0 (not at the left edge of the
universe)
    Decrement the column index ctemp
    If the specified cell is a live cell
            Increment the number of live cells

// check for top neighbors
Reset rtemp and ctemp to r and c respectively
Create a uint32_t variable named indextopneighbors and initialize it to 2
If the row index is at 0
    If the universe is toroidal
        Set rtemp equal to the last row value (wrap around)
        If the column is at index 0
             Set ctemp equal to the last column value (wrap around)
        Else
            Decrement ctemp
        For a uint32_t variable i in the range of 3
            If the current column index is at the top edge of the
            universe, then wrap around to column index 0 in
            the toroidal universe using modular arithmetic
            If the specified cell is a live cell
                Increment the number of live cells
            Increment ctemp
Else if the row index is greater than 0
    Decrement rtemp
    If the column is at index 0
        If the universe is toroidal
            Set ctemp equal to the last column value (wrap around)
        Else
            Set indextopneighbors equal to 1
    Else
        Decrement ctemp
        In a for loop in the range of 0 less than or equal to
            indextopneighbors
            If the universe is toroidal
                If the current column index is at the right edge of
                the universe, then wrap around to column index
                0 in the toroidal universe using modular
                arithmetic
        If the specified cell is a live cell
            Increment the number of live cells
        Increment ctemp

// check for bottom neighbors
Reset rtemp and ctemp to r and c respectively
Create a uint32_t variable named indextopneighbors and initialize it to 2
Increment rtemp
If the universe is toroidal
    If the current row index is at the bottom edge of the universe, then
    wrap around to row index 0 in the toroidal universe using modular
    arithmetic
If the column index is 0 (left edge of the universe)
    If the universe is toroidal
        Set ctemp equal to the last column value (wrap around)
    Else
        Set indextopneighbors equal to 1
Else
    Decrement ctemp
In a for loop in the range of 0 less than or equal to indextopneighbors
    If the universe is toroidal
        If the current column index is at the right edge of the universe,
        then wrap around to column index 0 in the toroidal universe
        using modular arithmetic
        If the specified cell is a live cell
            Increment the number of live cells
        Increment ctemp

Return the number of live cells
```

Create a void function named uv print with parameters of a pointer to a universe, and a pointer to an outfile. This function prints out the Universe to an outfile. A live cell is denoted by a lower case ‘o’ and a dead cell is denoted by a period ‘.’.

```
Void function uv_print with a Universe parameter and pointer to a file named
outfile
    Create a for loop to loop through the number of rows (i)
        Create a for loop to loop through the number of columns (j)
            If the cell at the specified index is live
                Print the cell as an ‘o’ in the outfile
            Else
                Print the cell as a ‘.’ in the oufile
        Print the outfile
```

<u>life.c</u>

Include life.h, inttypes.h, stdbool.h, stdio.h, stdlib.h, stdint.h, unistd.h, and ncurses.h

Define DELAY to be 50000
Define command line options for t, s, i, o, n, and h

Create a void function called uv populate empty universe which takes a pointer to a universe as a parameter. This function populates a universe with dead cells.

```
Create 2 uint32_t variables called rows and columns and set them equal to the row
    and column in the Universe Struct
Create a for loop to loop through the rows
    Create a for loop to loop through the columns
        Initialize the Universe with dead cells
```

Create a void function called generate which takes 2 pointers to a Universe as parameters (named u and v). This function implements the 3 rules of the Game of Life.

```
Create a uint32_t variable named numlivecells and initialize it to 0
Create a for loop to loop through the rows
    Create a for loop to loop through the columns
    // implement rule 1
    If the value of the cell in universe u at a specified row and column is true
        (indicating a live cell)
        Set numlivecells equal to the number of live cells by calling uv census
            function with universe u and the row and the column
        If there are 2 or 3 live cells
            Set the cell as a live cell
        Else
            Set the cell as a dead cell
    // implement rule 2
    If the value of the cell in universe u at a specified row and column is true
        (indicating a dead cell)
        Set numlivecells equal to the number of live cells by calling uv census
            function with universe u and the row and the column
         If there are 3 live cells
            Turn the dead cell into a live cell
        Else
            Keep the dead cell a dead cell
    // Since I initialized universe v with all dead cells, rule 3 is already
        implemented
```

Create a void function called print help with a void parameter which prints out a help message when the -h option is present on the command line.

```
In main,
    initialize an integer opt variable to 0
    2 uint32_t variables called rows and columns to 0
```

```
another uint32_t variable named generations to 100 (default number of
    generations)
2 boolean variables named toroidal and silent to false
1 pointer to a file named infile initialized to standard input
1 pointer to a file named outfile initialized to standard output

Create a while loop to parse the command line
    Create a switch statement for the different command line options
        Case t specifies the Game of Life to be played on a toroidal
            universe
        Case s silences ncurses
        Case n specifies the number of generations that the universe
            goes through (default is 100 times)
        Case i specifies the input file to be read to populate the universe
            (default standard input).
        Case o specifies the output file to print the final state of the
            universe to (default standard output)
        Case h prints the help message by calling the print help function
        Case ? is a case to check whether no infile or outfile are
            specified on the command line by the user in the -i and
            -o cases and prints the help message and closes the
            outfile
        The default is to break out of the switch statement

Create 2 universes A and B
Scan the infile
initialize universes A and B to the uv create function

If A is successfully populated
    Initialize universe B to have dead cells
    Create 2 universe variables named currgen and nextgen and set them
        equal to A and B respectively
    Traverse through number of generations specified by the user or the
        default number of generations
        Set up ncurses to display the state of the universe
        Implement the 3 rules of the game by calling the generate
            function with parameters currgen and nextgen
        If the current generation (currgen) is A
            Set the current generation to B
            Set the next generation (nextgen) equal to A
        Else
            Set currgen to B
            Set nextgen to A
        Populate universe nextgen with empty cells
    Print currgen to an outfile

Else
    Print the error message and close the open files (input and output)
    Call the destructor function to free allocated memory for universe A
        and B
    Return 0 to end the game

Close the open files (input and output)
Call the destructor function to free allocated memory for universe A and B
Return 0 to exit
```