Nicolas Corfmat
CSE13S, Winter 2023
ncorfmat@ucsc.edu

Assignment 4 DESIGN.pdf

1 Description of Program:

This program simulates "The Game of Life," an evolutionary game that requires no further input beyond an initial state. Essentially, a 2-D grid representing a *universe* consists of either dead or alive cells, whose states are updated as time, or *generations*, progresses. The rules of the game are straightforward. If a live cell has two or three live adjacent cells, it survives. A dead cell becomes live if it has exactly three life neighbors. Lastly, any cell that does not meet this criteria dies, and the cycle continues. The game ends once it has performed its last generation.

2 Files to be included in directory "asgn4":

1. universe.c:
    - Implements the Universe abstract data type.
2. universe.h:
    - Header file containing the interface to the Universe abstract data type.
3. life.c:
    - Contains the main() and other necessary functions needed to run the Game of Life.
4. Makefile:
    - Supports the compilation of the all .c programs in the "asgn4" directory
5. README.md:
    - Proper explanation of how to run the program and *Makefile*. It also contains a list of available command-line options and explains their functions, as well as any false positives reported by scan-build.
6. DESIGN.pdf:
    - Explains the thought process behind the program with sufficient detail. Comes in PDF format.
7. WRITEUP.pdf:
    - PDF file created using LaTeX that includes graphs displaying the differences between approximate and actual values and an analysis as to what they mean.

3 Pseudocode / Structure:

universe.c:

Create the abstract data type “Universe”:
    Define a 32-bit unsigned integer variable for rows and columns
    Define a grid pointer
    Define a boolean variable “toroidal”

Create the Universe constructor that takes three parameters (rows, cols, and toroidal):
    Allocate memory for a Universe pointer
    Store the values of the parameters (rows, cols, and toroidal) into the created Universe

    Allocate memory for the grid
    For every row in the grid:
        Allocate memory for each row

    Return the Universe

Create a function that takes a Universe:
    For every row in grid:
        Deallocate memory of that row (free space)
    De-allocate the grid’s memory
    Deallocate the Universe’s memory

    Set the Universe to NULL

Create a function that takes a Universe:
    Return the number of rows in that Universe

Create a function that takes a Universe:
    Return the number of columns in that Universe

Create a function that takes a Universe, row, and column:
    If the row and column are within the grid of the Universe:
        Set the grid cell at (row, column) as live (true)

Create a function that takes a Universe, row, and column:
    If the row and column are within the grid of the Universe:
        Set the grid cell at (row, column) as dead (false)
Create a function that takes a Universe, row, and column:
    If the row and column are within the grid of the Universe:
        Return the state of the cell at (row, column)

    Return false

Create a function that takes a Universe and file:
    If the file contains nothing or doesn't exist:
        Return false indicating that the Universe failed to populate
    For every row-column pair in the file:
        If the row or column is outside the Universe grid:
            Return false
        Else:
            Make the cell at (row, column) alive
    Return true to indicate that the Universe successfully populated

Create a function that takes a Universe, row, and column:
    Create a variable *counter* and initialize it to 0

    For every row-column pair coordinate adjacent to the passed row and column:
        If the row and column are NOT equivalent to the cell whose neighbors we are checking:
            If the Universe is toroidal:
                Wrap the row
                Wrap the column
            Else:
                Shift the row
                Shift the column
            If the current row and column values are within the Universe boundaries:
                If the cell at the updated row and column is alive:
                    Increment *count* by 1

    Return count

Create a function that takes a Universe and file:
    For every row in the Universe:
        For every column in the Universe:
            If the grid cell at (row, column) is alive:
                Append “o” to the file
            Else:
                Append “.” to the file

    Print a new line

life.c:

Create the function *perform_generation()* which takes pointers to Universe A and B and number of rows and columns as parameters:
    Set the number of live cells to 0

    For every row:
        For every column:
            Record the number of live cells adjacent to (row, col) of Universe A

            If the cell at (row, col) of Universe A is alive:
                If the number of live adjacent cells is either 2 or 3:
                    Make the cell at (row, col) of Universe B alive
                Else:
                    Make the cell at (row, col) of Universe B dead
            Else:
                If the number of live adjacent cells is 3:
                    Make the cell at (row, col) of Universe B alive
                Else:
                    Make the cell at (row, col) of Universe B dead

Create the function *swap_universes()* that takes two Universe pointers as parameters:
    Create a temporary Universe and assign it the value of Universe A

    Copy Universe B to Universe A
    Copy the temporary Universe to Universe B

Create the function *swap_universes()* that takes a Universe pointer as parameter:

- For every row in the Universe:
  - For every column in the Universe:
    - If the cell at (row, col) is alive:
      - Display “o” on the screen
    - Else:
      - Display “ “ on the screen

Main function:

- Create getopt variable and set it to 0
- Initialize default generations to 100
- Initialize *toroidal* to false
- Set default input file to *stdin*
- Set default output file to *stdout*

- While retrieving arguments from command line:
  - If user types -t:
    - Make the Universe toroidal
  - If user types -s:
    - Silence *ncurses*
  - If user types -n:
    - Accepts an argument for changing the number of generations
  - If user types -i:
    - Takes the user-specified input file
  - If user types -o:
    - Takes the user-specified output file
  - Default case:
    - Prints out program usage

- Record the number of rows and columns specified in the first line of the input file

- Create Universe A and B with the values defined for rows, columns, and toroidal

- If the Universe failed to populate:
  - Print an error message

- Initialize the screen
- Hide the cursor

For every number starting at 0 and ending at number of generations:

If *ncurses* isn't silenced:

Clear the screen
Display Universe A
Refresh the screen
Sleep for 50000 microseconds

Perform a generation (calls to *perform_generation()* function);
Swap Universe A and B (calls to *swap_universes()* functions)

Close the screen

Output Universe A to the specific file

Delete (deallocate) Universe A and B

Close the input and output files

Exit main