Assignment 3- The Game of Life

This program simulates the Game of Life: a zero-player game played on a 2-D grid representing a universe which uses an initial input to progress through generations of time. The grid, which can either be flat or toroidal, contains cells that can be dead or alive— depending on the state of their neighbors.

Basic Organization:

- In life.c
  - defines command-line getopt() options
    - -t: specifies universe to be toroidal
    - -s: silences ncurses library
    - -n: specifies # of generations universe goes through (default # = 100)
    - -i: specifies input file to be read to populate Universe (default = stdin)
    - -o: specifies output file to print Universe final state (default = stdout)
  - contains main()
    - calls functions from universe.c
    - uses return values from universe.c to print output when specific getopt() options are called

- In universe.c
  - contains functions to be called in life.c
    - Universe *uv_create (int rows, int cols, bool toroidal)
      - constructor function -> uses input row + col values to create Universe as well as information specifying toroidal/flat
      - memory is allocated using calloc()
      - returns pointer to Universe
    - void uv_delete (Universe *u)
      - destructor function -> frees any memory allocated for Universe by constructor function
    - int uv_rows (Universe *u)
      - returns # of rows in Universe
    - int uv_cols (Universe *u)
      - returns # of columns in Universe
    - void uv_live_cell (Universe *u, int r, int c)
      - marks cell at location r,c as alive (if cell out of bounds = nothing done) [alive = TRUE; dead = FALSE]
    - void uv_dead_cell (Universe *u, int r, int c)

        - marks cell at location r,c as dead (if cell is out bounds = nothing done)
    - bool uv_get_cell (Universe *u, int r, int c)
        - returns value/status of cell at location r,c (FALSE returned if out of bounds; TRUE = alive cell)
    - bool uv_populate (Universe *u, FILE *infile)
        - populates the universe using row-col pairs from infile (returns FALSE if pair is out of bounds)
    - int uv_census (Universe *u, int r, int c)
        - counts the neighbors of a specified cell and checks their status (dead/alive)
    - void uv_print (Universe *u, FILE *outfile)
        - prints status of cells in universe (flattened) to outfile (live cell = 'o'; dead cell = '.')

Pseudocode:

1) universe.c:
struct universe defined
helper function(inbounds)
    checks that a specified cell is within the bounds of the universe
function (create-universe)
    scan input
    dynamically allocate memory
        *first: allocate pointers to rows
        *second: allocate rows
function(delete-universe)
    clears memory allocated to universe- for use at the end:
    free row
    free grid
    free universe (think about water in bucket analogy)
function(rows)
    returns #rows in universe
function(cols)
    returns #cols in universe
function(uv_live_cell)
    if toroidal is true
        if bounds are exceeded, mod first to normalize to bounds
        then assign true for given row, col
    if toroidal is false

```
            if bounds are exceeded
                return false
            else
                assign true for given row, col
function(uv_dead_cell)
    same as true except assign as false
function(uv_get_cell)
    if toroidal is true
        if bounds are exceeded, mod first to normalize to bounds
        then return value for given row, col
    if toroidal is false
        if bounds are exceeded
            return false
        else
            return value for given row, col
function(uv_populate)
    reads remainder of input file
    for each specified row, col read
        sets grid(row,col) = true
function(uv_census)
    initialize count of alive cells
    for a given cell (row, col)
    iterate from row-1 to row+1
        iterate from col-1 to col+1
            as long as != row and != col(ignore given (row, col))
                get value of cell
                if get value = true
                    count = count + 1;
function(uv_print)
    iterate through each cell of grid
    from row = 0 to row < row length, row+1
        from col = 0 to col < col length, col+1
            print (row, col)
        print newline char
```

2) life.c:

```
set up getopt options(similar to asgn2)
read first line of input file
call create universe 2x
call populate universe
```

```
set up ncurses
        if not silenced:
                clear screen
                display universe A
                refresh screen
                sleep for 50000 microseconds
call census for each cell in A
        set value of respective cell in B with counts returned from census
swap universe A and B
close screen (endwin())
output universe A to specified outfile
```