Project 1 - Wildfire

Submission Due: Week 7, Thursday March 09, 2017, 11:59PM

Learning Objectives

This project will give you practice using the C language to:

- Process command line arguments of the form `-xN`;
- Use cursor-control functions to output characters;
- Design and use structures, arrays and matrices; and
- Use random number generation functions and basic math operations.
- Write a report on the status and behavior of your program, plus things you learned while doing the work.

Problem Introduction

First read the PDF description of about Spreading of Fire*[Shiflet]* to get an understanding of the problem and an overview of one possible simulation algorithm and the description of the various *Assignment* options.

You will write a program that **implements a combination of [Shiflet] Assignments with variations.** You must read *[Shiflet]* thoroughly to understand the rest of this document.

This simulation will repeatedly compute and display a new state of the system, which shows how a forest fire might progress. Each state represents the start of a simulation cycle.

Your program will represent the simulation with a grid of cells that have display character value, and it will use cursor-control functions to overlay characters showing changes to the grid's cells as the fire spreads. In addition to this default *overlay display mode*, the program will have an optional *print mode* that sequentially prints another grid for each simulation cycle without any overlay.

Summary of the Variations

This summarizes the differences between this project and the *[Shiflet]* description.

- Initializing the simulation will *not create boundary cells*. The forest of trees extends to the left, top, right, or bottom edges of the simulation grid.
- Updating rules will *use 8-way connectivity* of neighbors. This variation extends the description of the *[Shiflet]* `spread` function to take into account the state of cells to the northeast, northwest, southeast and southwest.
- Applying the *spread* to each cell is completely different because you will not use any dynamic memory allocation. The `applySpread` function described in *[Shiflet]* will not return a new grid nor will it ignore

boundary grid cells. Instead, an *update* function will modify the grid in place.
- Updating will extend the possible states of a grid cell to include representing a *burned tree* in addition to the empty, tree and burning states. This means the state sequence of the life of a tree that burns is from 'live tree' to 'burning tree' to 'burned tree'.
- The simulation program will continue until all fires are out and display the results of each simulation cycle using cursor controls in an overlay display manner.
- Display of the simulation will have two options or modes:
    1. cursor-controlled, character overlay display mode; and
    2. printed sequence mode.

    The cursor-controlled character display mode will be a *poor person's visual animation facility* that displays the sequences of simulation changes by overwriting the display of the previous state after a short time delay.

    The printed sequence mode will work as an option known as `-pN`, and the program will print each of the *N* cycles separately to standard output and terminate afterward.

- This project relates to the "*Assignments*" listed in *[Shiflet]* as follows:
    - This project combines *[Shiflet]* Assignments 1, 2, 4, 6, and 7.
    - Assignment 1 corresponds roughly to the `-pN` print mode option. In this case, the simulation runs, prints the result of each update cycle, and terminates.
    - Assignment 2 corresponds roughly to the cursor-controlled overlay display mode. The simulation runs in an open-ended fashion displaying each cycle using cursor control and terminating only when all fires have burned out.
    - Assignment 4 adds the proportion burning requirement. That populates the initial simulation state with some proportionate number of burning trees.
    - Assignment 5, which is for *extra credit*(see details below), makes a tree catch fire based on being hit by lightning. You will model this as the probability of a lightning strike as described in *[Shiflet]*.
    - Assignment 6 in *[Shiflet]* makes a tree burn out in 2 update cycles. When a tree starts burning, that is one cycle. Then it remains burning for the next cycle after that, and on the third cycle, it burns out.
    - Assignment 7 adds the number of neighboring trees on fire using *use 8-way connectivity* of neighbors as a factor in the *catching-fire* rules.
    - Assignment 12, which is for *extra credit*(see details below), introduces tree dampness, which makes trees catch fire more 'reluctantly'.

Projects Planning and Estimation

Don't panic!

Although this might be your first, mostly *from scratch project*, you should be able to design a solution to this program.

You have to plan your time to make sure you understand, and then address, all the details of the requirements.

You will learn to budget your time to handle a variety of tasks as you gain more experience developing programs.

Programs are not just writing the code; there's investigating of the problem, designing the function interfaces (parameters, types, return values), compiling, linking, testing, debugging and assessing the results.

So, **start early** to investigate, and follow a checklist like the one below. When you get stuck, re-read the assignment to see if you missed something, consult the discussion board for new information, and ask your instructor; consider them your coach. If you can't reach your instructor at 1AM, post a new thread if there are none that address your problem, or add a post to an existing thread if you have a follow-up concern.

So... here's a checklist (it's recursive!):

1. Read the assignment.
2. Consult the discussion board for new information.
3. Decompose the problem into components (functions and/or modules).
4. Write code to implement (and save versions).
5. Run tests for system (e.g. use scripts for different test cases).
6. Assess the program results and your code for ways to improve it.
7. Repeat until the results are correct.

The more time you spend up-front on decomposing the problem and making sure the pieces (functions and modules) can work together, the smoother things will go.

Requirements

How the Program Should Work

The default simulation runs as a potentially endless sequence of cycles. Each cycle is a step in time showing the spread of fire. Because the simulation ends if and when there is no more fire, or the user hits the `CONTROL-c` key sequence, the number of cycles is finite, but *indeterminate*.

The default, overlay display mode clears the screen first, and then loops printing the grid starting on the first line of the terminal window. You will use the supplied `display.c` to control the positioning of the cursor for the grid display. Below the grid will be one or more *simulation settings lines*.

The simulation output consists of the grid and lines reporting settings and the state of the simulation. Each cycle of the simulation begins its presentation with the grid and ends with lines displaying settings and changes.

After printing the last line of the grid, the program presents the simulation settings and sleeps to simulate the passage of time before executing another update and allow human viewing.

The `-pN` option runs a fixed number of update cycles and prints each update separately and sequentially

scrolling out in the terminal window. The `-pN` print mode prints the grid starting on the line after the command was entered. The program *does not clear the terminal screen in the `-pN` print mode*.

Example Program Runs

You might invoke your program like this:

```
$ wildfire -s11 -c77 -d50 -b25
```

The program clears the screen before starting the simulation and prints lines of information after the last row of the grid. The ***space*** characters represent empty cells, the letter (***Y***) characters are cells with living trees, the asterisk (*******) characters are cells with burning trees, and the period (**.**) characters are cells with burned trees.

Note: There are comment lines inserted in the outputs of examples in this document. These comments begin with the '#' character.

```
$ wildfire -s11 -c77 -d50 -b25
# The program cleared the screen here.
   YY  ..
.     .
. .. . ...
 .  ...
 ...... ..
   . .... .
. ... .. ..
 .  . .. .
..     ....
 . . Y  ...
 ..  YY .
size 11, pCatch 0.77, density 0.50, pBurning 0.25, pNeighbor 0.25
cycle 9, changes 1, cumulative changes 97.
Fires are out.
$
```

The screen0.txt file shows the simulation running with the default simulation settings (no command line arguments) after all the trees on fire have burned out. Note that there are two lines of information printed at the start of each cycle, and there are three lines of information printed at the end.

Below the presentation of the grid of trees you can see lines of output containing this information:

- The first line prints the following before every cycle:
  1. the size of the grid, which is square.
  2. the probability of a tree catching fire.
  3. the density of the simulation.
  4. the proportion of the tree population initially on fire.
  5. the proportion of neighbors that will influence a tree catching fire.
- The second line prints the following before every cycle:
  1. the cycle of the simulation, which starts at cycle 0 -- the initial state;
  2. the number of changes in the most recent cycle; and
  3. the cumulative number of changes of all cycles to this point.

- The third line prints only after all the fires have burned out. It simply states "Fires are out."

  A single change is a tree state change either from 'live tree' to 'burning tree', or from 'burning tree' to 'burned tree'.

Command Line Inputs

Below are some examples of command lines that would run the program. All arguments are optional; the program must run with default settings if no arguments are given.

It is guaranteed that the command line argument values used in testing, if any, will follow the *-xN* or *-x N* formats, and the *N* value will be a numeric integer value.

```
$ wildfire -H
usage: wildfire [options]
By default, the simulation runs in overlay display mode.
The -pN option makes the simulation run in print mode for up to N cycles.

Simulation Configuration Options:
 -H  # View simulation options and quit.
 -bN # proportion of trees that are already burning. 0 < N < 101.
 -cN # probability that a tree will catch fire. 0 < N < 101.
 -dN # density/proportion of trees in the grid. 0 < N < 101.
 -nN # proportion of neighbors that influence a tree catching fire. -1 < N < 101.
 -pN # number of cycles to print before quitting. -1 < N < ...
 -sN # simulation grid size. 4 < N < 41.


$
```

- The example above is the `-H` option, which prints usage information to `stderr`.
- `-bN` is the proportion of the population that is burning at the start of the simulation. The proportion burning is an initial value, which will vary as the simulation proceeds. The minimum is 1, and the maximum is 100. The program interprets this command line integer as a percentage value between 1% and 100%.
- `-cN` is the probability of a tree catching fire in percentage terms and expressed as an *integer*. A value of `60` means that each tree has a liklihood of catching fire of `60%`. The accepted minimum is 1, and the maximum is 100.
- `-dN` is the proportion of the simulation space that is occupied expressed as an *integer*. The program receives this value and interprets it as the percentage of the whole space (`N X N`) that is to be occupied by trees. The minimum is 1, and the maximum is 100.
- `-nN` is the proportion of neighbors above which a tree will become susceptible to catching fire by the probability of a tree catching fire (`-cN`). The minimum is 0 for no neighbor effect, and the maximum is 100, meaning that all a tree's neighbors must be burning for the tree to become susceptible to catching fire.
- `-pN` is an argument that puts the simulation into a fixed-count, print mode instead of an indeterminate loop, overlay display mode. The `N` in the `-pN` must be an integer that is the desired maximum number of cycles to simulate before terminating the program. For example, `-p4` would run for 4 cycles -- 1, 2, 3, 4

-- and display 5 grids because grid 0 is the starting state, before any simulation update cycle.
- `-sN` is the width of the grid in characters and the height in rows. The simulation grid will be set to (`N` x `N`) grid cells that are empty or filled with trees. The minimum size for `N` is 5, and the maximum is 40.

Outputs of the Simulation

Default Overlay Display Mode

Since C has no built-in visualization tools, you have to simulate an evolving grid of ASCII characters and using cursor control to position the cursor before 'drawing' a character. You will receive source code to a library that implements the cursor control to perform what's known as an overlay display.

The file screen-end.txt shows the result of display mode execution after all fires have gone out.

Print Mode

Using the `-pN` command line option will launch the print mode *instead of the cursor-controlled, overlay display mode*. Rather than clearing the screen and positioning the grid at the upper left, the print mode simply uses `printf` or `putchar` to print the characters of each line of the grid and let the terminal window scroll the output as printing proceeds.

You must not call any of the cursor-control display functions when the user specifies the `-pN` print mode command line option.

The file screen-p7.txt shows the result of a print mode execution that prints multiple cycles in a continuous terminal line sequence. Notice that the simulation stopped before all fires burned out, which happened before the number specified by the `-pN` option.

Error Output

If the command line is incorrect in terms of the values for the options, (i.e. outside the range of the values for a single argument), then the program should *issue an information message* describing the specific problem, print the usage message and terminate with a failure value (e.g. `EXIT_FAILURE`) from the main function.

Whichever simulation setting error occurs first in processing will determine which message is output before the usage message.

Below is the text of the error information messages.

- `(-bN) proportion already burning. must be an integer in [1...100].`
- `(-cN) probability a tree will catch fire. must be an integer in [1...100].`
- `(-dN) density of trees in the grid must be an integer in [1...100].`
- `(-nN) %neighbors influence catching fire must be an integer in [0...100].`
- `(-pN) number of cycles to print. must be an integer in [0...10000].`
- `(-sN) simulation grid size must be an integer in [5...40].`

All information messages and usage messages must be printed to `stderr`. Refer to the examples for the content of the usage message.

Here is a complete example of when the user entered a `-b` value outside of the proper range:

```
$ wildfire -b-1
(-bN) proportion already burning. must be an integer in [1...100].
usage: wildfire [options]
By default, the simulation runs in overlay display mode.
The -pN option makes the simulation run in print mode for up to N cycles.

Simulation Configuration Options:
 -H  # View simulation options and quit.
 -bN # proportion that a tree is already burning. 0 < N < 101.
 -cN # probability that a tree will catch fire. 0 < N < 101.
 -dN # density/proportion of trees in the grid. 0 < N < 101.
 -nN # proportion of neighbors that influence a tree catching fire. -1 < N < 101.
 -pN # number of cycles to print before quitting. -1 < N < ...
 -sN # simulation grid size. 4 < N < 41.
```

Design and Development Details

Program Operation Steps

Your solution will need to do the following:

- Get command line inputs, convert and store them for use by the simulation. Any global, top-level variables you define should be `static` to make them private. Functions that serve private purposes in the source file(s) should also be `static`.
- Define and initialize the data structures using any values on the command line as *overrides* of the defaults.
- Present the initial state of the grid known as *cycle 0*.
- Execute a loop until it detects no burning trees or it reaches the `-pN` limit:
  - Performing a simulation update cycle, and
  - Presenting the resulting grid and simulation information.

  The program will terminate when the most recent cycle finds no trees on fire because the fires have all burned out.

The Spread Algorithm

This corresponds to Assignment 7 with the variation that you must extend the `spread` function to handle *8-way connectivity* of neighbors.

Rather than checking if at least one neighboring tree is burning before applying the random number check as the basic simulation in *[Shiflet]* does, your algorithm first must check that the proportion of neighbors burning is above the level at which a tree will become susceptible to catching fire. (See the `DEFAULT_PROP_NEIGHBOR` and the command line options.)

To know the proportion of neighbors that are burning, you have to count how many total tree neighbors a cell has, and how many of them are burning; this ignores empty neighbor cells. That means you have to count how many neighbors a cell has, and how many neighbors are trees, counting burned out trees as empty cells, and remembering that some of a cell's 8 neighbors will not exist if the cell is a cell at the boundary of the grid.

When it is the case that a high enough proportion of neighbors are burning to exceed the neighbor proportion threshold, a random value between 0.0 and 1.0 will be generated to compare against the liklihood of any single tree catching fire, and that will decide whether a tree should start burning.

If that random value is less than the probability of catching fire, then the tree should catch fire.

For trees that burn, there will be a 2-cycle burn. That means when a tree starts burning (i.e. changes to the burning state for the first time), it will remain in the burning state for the next 2 update cycles. At the third cycle, a burning tree will 'burn out' and become a tree in a burned state.

The Update or `applySpread` Algorithm

The algorithm described in *[Shiflet]* states that a new representation or configuration is created for each consecutive simulation cycle.

While it is possible to implement the algorithm as stated in *[Shiflet]*, you must do it differently for this project because you have not yet learned the dynamic memory management that would be necessary to implement the *[Shiflet]* algorithm in the C language. (This non-dynamic simulation will actually be faster because it will not have to allocate and de-allocate memory.)

Rather than writing the fire, spread and applySpread functions as described in *[Shiflet]*, you will write an *update* function to modify the grid in place.

The update operation applies a spread function to each grid cell to potentially turn a tree into a burning tree or turn a burning tree into a 'burned up' tree. As part of this update, you will need to manage the notion of state transitions so that you check the correct state of neighboring trees during the update.

Note that the decision on whether one tree catches fire depends on the state of its neighbors, and those neighbors may also be undergoing possible state changes. The decision to catch fire must be based on the state of neighboring trees *at the start of the current cycle*, not their current state, which might have already changed due to the update in progress.

The Simulation Loop

Running the simulation involves executing a loop that continually applies the update algorithm, tracks the number of changes and checks whether all fires are out. The loop stops when all fires are out or the number of

cycles specified by the `-pN` option has been reached.

Default Simulation Settings

In the absence of command line arguments, there are default simulation setting values. You should use the names given here as the default values, create variables, and initialize those variables to their defaults. The command line arguments may then override the defaults.

- `DEFAULT_BURN` should specify 10% for the proportion of burning trees.
- `DEFAULT_PROB_CATCH` should specify 30% as the liklihood of of any single tree catching fire.
- `DEFAULT_DENSITY` should specify 50% as the proportion of trees in the grid.
- `DEFAULT_PROP_NEIGHBOR` should specify 25% as the proportion of neighbors above which a tree will become susceptible to catching fire by the `DEFAULT_PROB_CATCH` value.
- `DEFAULT_PRINT_COUNT` should specify a value so that print mode will be turned off and overlay display mode will be on.
- `DEFAULT_SIZE` should specify a 10 by 10 grid.

When a user provides one of the optional command line arguments, the new values will override the defaults, and the simulation settings lines will display the actual settings in operation.

Getting Optional Command Line Arguments

The standard way to process a command line option such as `-pN` is with the `getopt` function, which processes command line arguments that begin with the minus(-) sign. The `man -s 3 getopt` command gives more information about this library function. The supplied file `use_getopt.c` demonstrates `getopt` use.

Random Numbers and Simulation Delay

The functions for generating random numbers and pausing are `srandom`, `random` and `usleep`; these require setting the value `BSD_SOURCE` before including any headers. The `srandom` function seeds the random number generator, and `usleep` pauses/delays the process for a given number of microseconds. A reasonable value for the delay is `750000` to allow analysis.

If you set the seed value to a constant, your process will have repeatable results. For this assignment, `try` uses `41` as the seed.

Supplied Files

There are a few supplied files that will get you started. The command below will fetch these files.

```
get csci243 project1
```

- `display.h`: the *header* file that is the interface for functions that manipulate the terminal window contents to clear the screen, position the cursor or output a single character. You must not change this

file. You will link it with your program object to create the executable.
- `display.c`: the *implementation* file for cursor control. You must not change this file.
- `use_getopt.c`: a *demonstration program* showing how you can use `getopt`. Study this code and adapt it to create your command line option processing functions.
- `report.txt`: a *plain text* template for the report you must submit with your program. This contains questions to answer regarding this project.

Extra Credit

You may earn *up to 10% extra credit* for completing your program with the extra features described here.

- (5%) Implement the *Assignment 5* from the Spreading of Fire so that a tree struck by lightning takes catches fire following the probability mechanism described in *[Shiflet]*.

- (5%) Implement *Assignment 12* which creates trees with some random amount of dampness. The dampness factor reduces the liklihood that a tree will catch fire.

  Since there is no further specification for how dampness reduces the chances of fire, you will have to design something yourself...

Note: You may choose to implement either or both of the extra credit items. Here is a further breakdown for the extra credits for each choice:

- 3% of each extra credit is for implementation and performance.

  You will need to add a command line argument for the additional simulation parameters. **Note: The program must work without those extra arguments by using a default value in their absence.** The default value must be to *turn off* the extra credit behavior. This ensures that your program will not fail when *try* runs tests that do not assume the existence of the extra credit.

- 2% of each extra credit is for the documentation of your extra credit experiments in your report. The experiment results for extra credit must appear in the extra credit section of your `report.txt`.
  - The documentation must specify how to use the optional behavior and some good values for the added simulation parameters so that test runs may verify the extra credit operation.
  - If you use different symbols to represent trees being struck by lightning or damp trees, your documentation must identify this. For example, you might want to use a special character for a tree struck by lightning (e.g. '!'). If you do choose different symbols, remember that these must not appear by default when the extra credit behavior is off.

Grading

Your grade will be based on the program's *design and functionality*, *style and documentation*, and your *written report*.

- **80% Design and Functionality**:
  - 10% Designing the code so that it is structured in a modular fashion with good naming and types (e.g. use of `#define` or `enum`), good data hiding (i.e. use of `static` for non-public functions) and reasonable problem decomposition.
  - 70% Functionality:
    - 10% Display: Initializing, populating and displaying the grid so that the proper proportion of empty, tree and burning tree objects is represented.
    - 15% Settings: Implementing the default simulation settings properly, and processing optional command line arguments that override the settings correctly regardless of option order.
    - 20% Print Mode: Running the simulation loop implementing the `-pN` print mode that prints each evolution in sequence and stops processing after a fixed number of update cycles. (Note: the `try` tests use only this mode because it cannot capture the dynamic display of overlay mode.)
    - 25% Overlay Mode: Implementing the fire spreading simulation correctly so that the overlay display mode displays results and runs until the fire ends.
- **10% Style and Documentation**: The program code conforms to style and documentation guidelines for the course found in C-Style-Recommendations.pdf. This includes adopting and using the *git* **version control system (VCS)** throughout the project to manage your project deliverables.
- **10% Report**: Complete your report following the instructions in `report.txt`. There are specific questions to answer about your program's behavior. Answers should be short, complete and thoughtful phrases or bullet points that contain your reflections on your work.

Submission

You are responsible for submitting these files at a minimum:

1. `wildfire.c`: implements the simulation of spreading fire.
2. `report.txt`: The file reports on your implementation, serves as a user guide and gives your reflections through answers to questions regarding this project.
3. `revisions.txt`: the output of running the `git log` command on your project.

Do not submit the `display.[ch]` files because `try` will supply those files during the submission and testing.

When finished, submit your work from your CS department account using the ***try*** command below.

```
try grd-243 project1-1 wildfire.c report.txt revisions.txt [other files ...]
```

Note that the `[other files ...]` files represent *optional* additional *.h and .c* files which might be part of your solution.

To check your submission, you can use the *try query* command like this:

```
try -q grd-243 project1-1
```

Please refer to try-summary.html for more information and guidance on using *try*.

Revision History:

- Version 1.1: Fri Feb 17 17:59:21 EST 2017 Ben K Steele, bks@cs.rit.edu

  Reformatted Grading section.

- Version 1.0: Wed Feb 15 08:11:09 EST 2017 Ben K Steele, bks@cs.rit.edu

  Fixed size specification in usage message.

- Version 0.9: Tue Feb 7 17:22:21 EST 2017 Ben K Steele, bks@cs.rit.edu

  Initial Release.
