CSE 6010
Assignment 2
Ecological Encounters

Initial Submission Due Date: 11:59pm on Thursday, September 12
Final Submission Due Date: 11:59pm on Thursday, September 19
Peer review and code reflections assignments due September 26 will be posted separately
Submit codes as described herein to Gradescope through Canvas
48-hour grace period applies to all deadlines

In this assignment, you will consider mobile animals occupying an ecological territory. The animals move around the territory in two dimensions looking for food, but without any preferred direction. As each animal moves randomly, it may encounter other animals. It may be of interest to track the number of encounters. (Such tracking can be accomplished in nature by placing tracking devices on the animals and noting when tracked animals are within a certain distance of each other.)

Your task is to write a program in C to simulate the random walks of multiple animals on a square Cartesian (X-Y) grid across multiple time iterations and to count the number of animal encounters that occur.

Specifications:

- Input values will be specified using command-line arguments in the following order:
    1. *Grid length:* The grid will be set to a square with this integer number of cells in each of the two dimensions. (Validate that the grid length is a positive number.)
    2. *Number of animals:* The grid will be occupied by this integer number of animals. (Validate that the number of animals is a positive number.)
    3. *Number of iterations:* The integer number of time iterations. (Validate that the number of iterations is a positive number.)
    4. *Stay probability:* This double value will be the probability that an individual animal will not move in a single time iteration (see below). (Validate that the stay probability is between 0 and 1; a stay probability of 0 means that animals will always move in a randomly chosen direction each iteration, whereas a stay probability of 1 means no animals will ever move.)
    5. *Seed value:* This integer should be converted to an unsigned int and used as the seed value using `srand()`.

五个input，在main function里面validate

- You should use dynamic memory allocation to create the grid as a one-dimensional array. Storing the grid as a one-dimensional array will make it easier to pass the array to functions. Store the value at two-dimensional grid location (i,j) at one-dimensional array index `i*gridLength+j` (where gridLength is the number of gridpoints in each dimension). (You may choose different names for your variables.)

一维的array

- Random numbers should be generated using `rand()` and the seed value specified. Note that the specific sequence of random numbers generated can vary across different systems, but within the autograder environment, the same seed will lead to the same sequence of numbers being generated for everyone. To ensure your output matches the autograder, be sure to use random numbers in the following way.

生成一系列random value

random value初始化动物的位置，loop先给动物1 x value，再给动物1 y value；再给动物2 x，动物2 y...

通过算%的方式确保指定的coordinate在合适范围内

- Random numbers will be used for the initial placement of each animal. Loop over each animal and determine first the x value, then the y value for each animal before proceeding to the next animal. You can generate each grid coordinate to be in the proper range by calculating the random number % (mod) the grid length. Thus, generate two random numbers to get the initial grid coordinates for the first animal, and repeat to place the remaining animals.

random number还决定动物的移动方式。在outer loop移动x，inner loop移动y。如何移到的位置已经有一个动物，那么就不移动那个动物，再生成一个random number决定它下一次的移动

- Random numbers will be used on each iteration to determine the movement of each animal, as described below. Update the position for each animal by sweeping over the grid (increment the first grid index in the outer loop and the second grid index in the inner loop). If the position is occupied, loop over the animals located there and generate a single random number to determine the animal's next position as described below.

将生成的random number除以rand_max，确保random number在0到1之间

- On each time iteration, each animal will either remain stationary or move, depending on a single random number. You should adjust the random number to be in the range [0,1] by dividing the random number by RAND_MAX. Note that you will need to cast at least one of these integer values to a double to avoid dividing two integers! E.g., `(double) rand()/ RAND_MAX`. The value of RAND_MAX is available in stdlib.h, which you already will need to include for dynamic memory allocation.
  - If the adjusted random number is less than the probability of remaining stationary as provided by the user, the animal's position should not change.
  - If the animal does not remain stationary, it will move 1 unit either north, south, east, or west, with the direction for each animal chosen randomly with equal probability. If the chosen direction would move an individual off the grid, their movement should be "reflected" back in the opposite direction. For example, if an animal at location [0][3] is directed to move to [-1][3], instead the animal should be moved to [1][3].
  - As an example, suppose the stay probability is 0.6, and call the random number generated r (after adjusting it to be in the range [0,1]).

(1-probability)/4

    - If $r < 0.6$, the animal should not be moved.
    - If $0.6 \leq r < 0.7$, move the animal from location (i,j) to (i-1,j) (reflecting as needed). 左
    - If $0.7 \leq r < 0.8$, move the animal from location (i,j) to (i+1,j) (reflecting as needed). 右
    - If $0.8 \leq r < 0.9$, move the animal from location (i,j) to (i,j-1) (reflecting as needed). 上
    - If $0.9 \leq r \leq 1$, move the animal from location (i,j) to (i,j+1) (reflecting as needed). 下
- When more than one animal occupies the same grid cell, we will call this event an encounter. Your program should count the number of times encounters occur. Essentially, any time a grid cell is occupied by more than one individual, including in the initial locations of the animals, the number of encounters should be incremented by one.
  - You do not need to differentiate encounters involving different numbers of animals (e.g., 2 vs. 3 vs. 4 animals): count any instance of more than one animal occupying the same grid cell as one encounter.

    - In addition, encounters are considered to last only for one time iteration, such that you should count a new encounter every iteration, even if the animals shared a grid cell the previous time iteration and share the same (or another) grid cell the next iteration.
- At the end of the program, print the number of encounters to the screen as a decimal integer followed by a newline character, with no accompanying text.
- All dynamically allocated memory should be freed. (valgrind will be used in the autograder to test for memory leaks.)
- Invalid input to the program should result in the program printing an error message and returning exit code 1. Make sure that all memory is freed appropriately in this case.

To receive full credit, your code must be well structured and documented so that it is easy to understand. Be sure to include comments that explain your code statements and structure.

Examples are included below to show the random number sequence generated from a particular seed with the autograder (it may be different on your local system but should match once submitted to the autograder), as well as to show how the random numbers are utilized for a small case.

Initial submission (worth 1 point): Submit your code to the Initial Submission on Gradescope. Your initial submission will not be graded for correctness or even tested but rather will be graded based on the appearance of a good-faith effort to complete the majority of the assignment. Any time you wish to test your code for correctness, please submit it to the Final Submission on Gradescope.

Final submission (worth 11 points): Submit your code to the Final Submission on Gradescope.

(1) your code (all .c and .h files); no specific filenames are required for your code but ***note the specification for your executable file below***. Do not zip the files or upload a directory, instead, submit the .c/.h files and the Makefile directly.

(2) the **Makefile** you use to compile your program such that your executable file that will run your program is named **encounters**. You are provided with two Makefiles; in most cases one of these should work for you, possibly with some renaming of filenames.

- One Makefile assumes that all code is written in a single file named encounters.c that is compiled into the executable named encounters.
- The other Makefile assumes the main function is written in main.c and that supporting functions are declared in encounters.h and defined in encounters.c; the compilation command compiles them into the executable named encounters.

Be sure to rename the Makefile you choose to just “Makefile”.

Some hints:

- Start early!
- Identify a logical sequence for implementing the desired functionality. Then implement one piece at a time and verify each piece works properly before proceeding to implement the next.
- Using the math library should not be necessary for this assignment. However, if you do use it, make sure to add the "-lm" flag in your Makefile. (Assignment 1 has an example of this usage.)
- Accessing uninitialized memory can result in very large numbers where they are not expected, leading to very long simulations for this program. None of the test cases should take long to run if implemented correctly. If running your code takes more than a few seconds, you can force it to stop with the keyboard Ctrl+C or command C-c (hold control and press the "c" key).
- The `atoi()` and `atof()` functions used to convert strings into integers and doubles will return a value of 0 if the string cannot be converted. Your program does not need to explicitly check that the input was a valid number. However, it should prevent values of zero being used where they are not valid, whether the input was 0 or could not be parsed as a number.

It is possible that the autograder will generate a different sequence of random numbers than your machine. If this is the case, the results on your computer will not match the example given in the assignment. Check the autograder output to make sure the program is behaving correctly.

Test case:

Given the following parameters:

- Grid length of 4 (4x4 grid),
- 10 animals,
- 100 iterations,
- Stay probability of 0.4, and
- Seed value of 0,

you should find 195 encounters. As mentioned above, your computer may generate a different sequence of random numbers that may lead to different output. Please check the autograder to confirm the results of your code for this test case.

Also see "Detailed examples" below for more information you can use for debugging

Grading

Program correctness will be assessed using the autograder on Gradescope. Detailed grading specifications are described below. Numbers in parentheses indicate the total points available for a specific component.

1. **Initial submission (1)**: Graded manually. The point is earned for making meaningful progress toward completing the assignment. The program does not need to compile or run correctly at this stage.
2. **Autograder (8)**
    a. **Compile the program (0)**: This test is not worth any points, but it will show the output if errors are encountered compiling your program on the autograder. This output should be helpful if your code compiles on your machine but not on the autograder. Fix these errors before moving on to anything else!
    b. **Check for invalid memory usage (2)**: This test uses valgrind to detect memory errors in your program.
    c. **Check correctness of input validation (1)**: Test whether your program successfully detects invalid inputs on the command line and exits with return code 1 in those cases.
    d. **Check correctness for input '4 10 100 0.4 0' (1)**: Test whether your program gets the correct output for the example test case in the assignment description. This test case is useful for making sure that your program generates random numbers in the correct order.
    e. **Check correctness of hidden inputs (4)**: Test whether your program gets the correct output for a variety of hidden cases.
3. **Program structure and implementation (1.5)**: Graded manually. Points are earned for implementing the algorithms and data structures correctly as described in the assignment and making reasonable decisions when designing your program.
4. **Program style and documentation (1.5)**: Graded manually. Points are earned for using abstractions to avoid redundant code and making your program readable by using appropriate comments, variable names, whitespace, indentation, etc. Write your code so someone else can easily follow what is going on.

The autograder will run immediately when the program is submitted and should present the results quickly. The manually graded components of the assignment will not be available until after the grades are published. Autograder feedback will only be available on the final submission for the assignment, as the initial submission will not be graded for correctness.

Detailed examples: The examples below are designed to help you understand the random number generation including the numbers generated on the autograder.

Random number sequence: Below are the first 20 numbers generated using the seed value 1.

1804289383
846930886
1681692777
1714636915
1957747793
424238335
719885386
1649760492
596516649
1189641421

1025202362
1350490027
783368690
1102520059
2044897763
1967513926
1365180540
1540383426
304089172
1303455736

Small example including the random numbers and how they are used: Below is an example of how the random numbers are used for a small example.

- Grid size: 4 x 4
- Number of animals: 2
- Number of iterations: 3
- Stay probability: 0.4
- Seed value: 1

Setup:

- Animal 0
    - Random value: 1804289383, x: 3
    - Random value: 846930886, y: 2
- Animal 1
    - Random value: 1681692777, x: 1
    - Random value: 1714636915, y: 3

Iterations:

- Iteration 0
    - Animal at (1, 3): Random value: 1957747793, Movement: y+1
    - Animal at (3, 2): Random value: 424238335, Movement: stay
- Iteration 1
    - Animal at (1, 2): Random value: 719885386, Movement: stay
    - Animal at (3, 2): Random value: 1649760492, Movement: y-1
- Iteration 2
    - Animal at (1, 2): Random value: 596516649, Movement: stay
    - Animal at (3, 1): Random value: 1189641421, Movement: x+1

The total number of encounters in this example is 0.