For your assignment, you need to simulate animal movements on a 2D grid using random walks and count the number of encounters between animals. Here are some hints to help you get started:

### 1. \*\*Command-line Argument Parsing\*\*:

- Parse the input values: grid length, number of animals, number of iterations, stay probability, and seed value using `atoi()` and `atof()` for integers and doubles.

- Ensure input validation: Check that the grid length, number of animals, and iterations are positive numbers, and the stay probability is between 0 and 1.

### 2. \*\*Grid Representation\*\*:

- Use dynamic memory allocation (`malloc()`) to create a 1D array representing a 2D grid. Map grid coordinates `(i,j)` to the array index using:

```c

int index = i \* gridLength + j;

```

- Initialize the positions of animals randomly using `rand() % gridLength` for both x and y coordinates.

### 3. \*\*Animal Movements\*\*:

- In each iteration, generate a random number to decide if an animal stays or moves.

- If the animal moves, generate a random number to pick the direction (north, south, east, west), and apply the boundary conditions (reflection at the grid edges).

- To ensure randomness, use:

```c

double random = (double)rand() / RAND\_MAX;

```

- Based on the random value, move the animal or keep it stationary.

### 4. \*\*Encounter Tracking\*\*:

- Track encounters by checking how many animals occupy the same grid cell. Increment the encounter counter whenever more than one animal is found in a single cell.

- Ensure that you count encounters during each iteration, even if the same animals remain in the same grid cell.

### 5. \*\*Memory Management\*\*:

- Use `free()` to release dynamically allocated memory at the end of the program.

- Ensure no memory leaks by testing your program with tools like `valgrind`.

### Solution Outline:

```c

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) {

if (argc != 6) {

printf("Error: Invalid number of arguments\n");

return 1;

}

// Parse command-line arguments

int gridLength = atoi(argv[1]);

int numAnimals = atoi(argv[2]);

int numIterations = atoi(argv[3]);

double stayProb = atof(argv[4]);

unsigned int seed = (unsigned int) atoi(argv[5]);

// Input validation

if (gridLength <= 0 || numAnimals <= 0 || numIterations <= 0 || stayProb < 0 || stayProb > 1) {

printf("Error: Invalid input values\n");

return 1;

}

srand(seed);

// Dynamic allocation of the grid (1D array)

int \*grid = (int \*)calloc(gridLength \* gridLength, sizeof(int));

if (!grid) {

printf("Memory allocation error\n");

return 1;

}

// Array to hold animal positions

int \*xPos = (int \*)malloc(numAnimals \* sizeof(int));

int \*yPos = (int \*)malloc(numAnimals \* sizeof(int));

// Initialize positions

for (int i = 0; i < numAnimals; i++) {

xPos[i] = rand() % gridLength;

yPos[i] = rand() % gridLength;

}

int encounters = 0;

// Simulation loop

for (int iter = 0; iter < numIterations; iter++) {

// Reset grid for this iteration

for (int i = 0; i < gridLength \* gridLength; i++) {

grid[i] = 0;

}

// Update positions and track encounters

for (int i = 0; i < numAnimals; i++) {

double r = (double)rand() / RAND\_MAX;

if (r >= stayProb) {

double move = (double)rand() / RAND\_MAX;

if (move < 0.25) {

xPos[i] = (xPos[i] - 1 + gridLength) % gridLength; // North

} else if (move < 0.5) {

xPos[i] = (xPos[i] + 1) % gridLength; // South

} else if (move < 0.75) {

yPos[i] = (yPos[i] - 1 + gridLength) % gridLength; // West

} else {

yPos[i] = (yPos[i] + 1) % gridLength; // East

}

}

int index = xPos[i] \* gridLength + yPos[i];

grid[index]++;

}

// Count encounters

for (int i = 0; i < gridLength \* gridLength; i++) {

if (grid[i] > 1) {

encounters++;

}

}

}

// Output the result

printf("%d\n", encounters);

// Free dynamically allocated memory

free(grid);

free(xPos);

free(yPos);

return 0;

}

```

### Steps in the Solution:

1. \*\*Input Parsing & Validation\*\*: Parses arguments and ensures all input is valid.

2. \*\*Grid Initialization\*\*: Allocates a 1D array to represent the grid.

3. \*\*Random Walk Simulation\*\*: Simulates random animal movement while handling boundary conditions.

4. \*\*Encounter Counting\*\*: Tracks the number of encounters per iteration.

5. \*\*Memory Cleanup\*\*: Ensures no memory leaks.

Would you like help testing this or adding more features?

检查input的合理性

用random number初始化animal的位置

**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.