EECE.2160: ECE Application Programming

Fall 2017

Programming Assignment #4: The "Drunken Sailor" Problem

Due **Monday, 10/16/17**, 11:59:59 PM

1. Introduction

This assignment, which introduces the use of loops, solves the following problem: given a starting location in a city, how long does it take a "drunken sailor" who randomly chooses his direction at each intersection to reach the city's border? You will read input values to set up the problem parameters, run several trials to determine an average number of steps for the sailor to reach the border, and output the results.

This problem is an example of a "random walk," a succession of random steps that can model real world problems like stock price fluctuation or molecules traveling through liquid. The approach is a simple approximation of a Monte Carlo experiment, in which repeated random samples are run to find a numerical result.

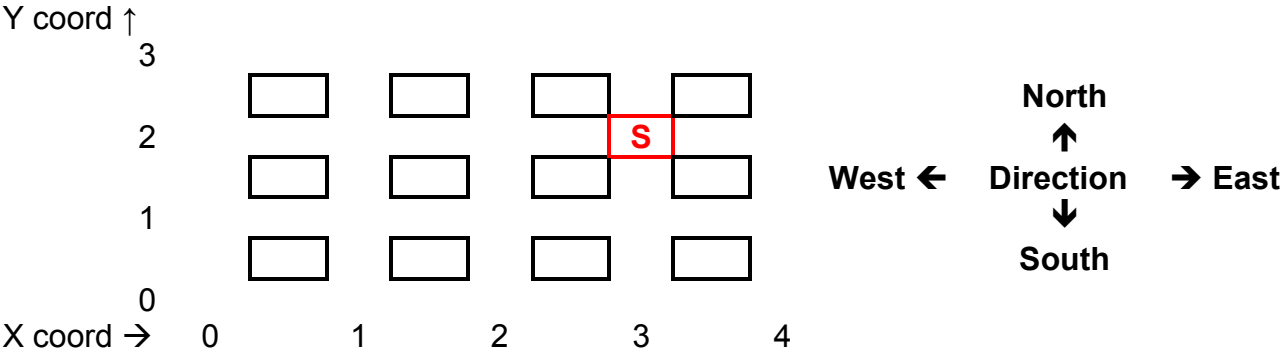
2. Deliverables

Submit your source file by uploading it directly to your Dropbox folder. Ensure your source file name is ***prog4_sailor.c***. You should submit only the .c file. Failure to meet this specification will reduce your grade, as described in the grading guidelines.

3. Specifications

Problem description: The city is organized as a set of M x N blocks. The sailor's position, which must always be an intersection or a point on the border, can be represented as a pair of coordinates (X, Y), where $0 \le X \le M$, and $0 \le Y \le N$.

The example below shows a 4 x 3 city, with the sailor at position (3, 2):



At each step of a given trial, the sailor will randomly choose a direction and walk until he reaches the next intersection. A trial ends when the sailor reaches one of the city borders. Note that each new trial always uses the same starting point.

Input: Your input must be entered in the order listed below. All inputs are integers. Note that your program should prompt the user to enter each value and check that each input fits within the bounds described below, as well as ensure there are no formatting errors:

- A seed value for the random number generator (RNG). If the user enters -1, use the system time to seed the RNG; otherwise, the user input is used. See Section 4, "Hints and Tips," for more details.
- M and N, the number of blocks in the X ($2 \le M \le 10$) and Y planes ($2 \le N \le 10$), respectively. This pair of values will be entered on the same line.
- A starting position for the sailor, input as a pair of integers (X,Y).
    - These values should simply be separated by a space (e.g., `3 5`). You should not format your input as an (X,Y) pair using parentheses and a comma (e.g., `(3,5)`).
    - The sailor must always start within the city, so the starting coordinates are subject to the following bounds: $1 \le X \le (M-1)$, $1 \le Y \le (N-1)$
- T, The number of trials to execute ($1 \le T \le 10$).

Output: As noted above, the program should print a prompt for each input. See Section 5, "Test Cases," for examples of acceptable prompts. If an input does not fit within the bounds described, or the user enters an improperly formatted value, the program should display an error message and then prompt the user again to enter that value. See Section 4 for hints on bounds checking and error messages.

Once the user has successfully input all values, the program executes each trial, starting at the point specified. At each step of a trial, your program generates a random number representing the direction the sailor moves and changes his position accordingly. Your program should print messages at the following points:

- At the start of each trial, the program should print a message indicating the start of a new trial and the starting point being used.
    - Example: `Trial #1 Start: 3 1`
- For each step of each trial, the program should print the direction the sailor moves and his new position.
    - Example: `North: 3 2`
- At the end of each trial, the program should print the total number of steps taken during that trial.
    - Example: `Trial #1 total steps: 8`
- Once all trials are complete, the program should calculate the average number of steps taken per trial and output that value.
    - Example: `Average # of steps over 5 trials: 3.2`

Again, see Section 5: Test Cases for detailed examples of how your output should look.

4. Hints and Tips

Bounds checking and error messages: See Lecture 13 (Friday, 10/6) for an example of how to handle input validation—repeatedly prompting your user to enter input values until the inputs are error-free.

Random number generation: The library function `rand()` generates pseudo-random numbers between 0 and RAND_MAX (a large, constant value defined in the header `<stdlib.h>`, which you must include to use this function). To get a random value between 0 and N, use the modulus operator (%) as follows:

```
X = rand() % (N+1);
```

For example, to generate one of four different values between 0 and 5, you can use `rand() % 6`.

Note that `rand()` is not truly random—if you do not provide a different starting point, or "seed", each time you run your program, `rand()` produces the same values. To specify a seed, use the function `srand(unsigned int seed)`. While your test cases may not match mine, if you repeatedly use the same seed value, your program should produce the same set of random values each time.

A common way to get (close to) true randomness is to use the system time as the seed:

```
srand(time(0));
```

`srand()` should only be called once per program, before any calls to `rand()`. To use the `time()` function, you must include the `<time.h>` header.

The course website contains a short program, `dice_example.c`, that demonstrates the proper use of the `rand()` and `srand()` functions. It also contains examples of an input validation loop like the one described above.

5. Test Cases

Note that, because this program uses random numbers, your outputs for each trial likely will not match the test cases below. (The 2x2 city case is an exception—your sailor may move in different directions, but every trial should take only 1 step to finish.) What should match is the order in which you read input values and the cases in which you print error messages.

I will use these test cases in grading of your program, but will also generate additional cases that will not be publicly available. Note that these test cases do not cover all possible program outcomes. You should create your own tests to help debug your code and ensure proper operation for all possible inputs.

I've copied and pasted the output from each test case below, rather than showing a screenshot of the output window. User input is underlined in each test case, but it won't be when you run the program.

Test Case 1:

```
Enter seed (-1 to use system time): 1
City size in X, Y  (# blocks >= 2 and <= 10): 2 2
Starting position (X Y): 1 1
Number of trials: 3
Trial # 1 Start: 1 1
  South: 1 0
Trial # 1 total steps = 1

Trial # 2 Start: 1 1
  East: 2 1
Trial # 2 total steps = 1

Trial # 3 Start: 1 1
  East: 2 1
Trial # 3 total steps = 1

Average # of steps over 3 trials: 1.00
```

Test Case 2:

```
Enter seed (-1 to use system time): 1
City size in X, Y  (# blocks >= 2 and <= 10): 1 1
# X blocks must be >= 2 and <= 10
# Y blocks must be >= 2 and <= 10
City size in X, Y  (# blocks >= 2 and <= 10): 12 3
# X blocks must be >= 2 and <= 10
City size in X, Y  (# blocks >= 2 and <= 10): 10 10
Starting position (X Y): 0 10
Starting X position must satisfy (1 <= X <= 9)
Starting Y position must satisfy (1 <= Y <= 9)
Starting position (X Y): 2 8
Number of trials: 0
Number of trials must be > 0 and <= 10
Number of trials: 1
Trial # 1 Start: 2 8
  South: 2 7
  East: 3 7
  East: 4 7
  North: 4 8
  North: 4 9
  West: 3 9
  West: 2 9
  North: 2 10
Trial # 1 total steps = 8

Average # of steps over 1 trial: 8.00
```

Remember, if you are using Visual Studio, to get your program to terminate with a message saying, “Press any key to continue …”, use the **Start Without Debugging** command (press Ctrl + F5) to run your code.

If you need to insert extra code at the end of your program to get that program to pause when executing (for example, an infinite loop or the system("pause") function), please remember to comment out or remove that code prior to submitting your program.