Grid Runner

The Grid Runners are a collection of young people dropped in the center of a large grid. The grid consists of straight streets in the cardinal directions of the compass. The interior of the blocks are 1000 foot high concrete blocks, covered with teflon, so the runners can’t climb to the top. The only way that the runners can move is going from one intersection to the next.

A collection runners are dropped at the origin of the grid and each given a device of chance. Each runner rolls their chance device, and it tells them which direction to travel. They move to the next intersection in that direction, then use the chance device again, following its direction. This process is repeated some specified number of times.

The overseers wait until every runner has completed the required number of steps, then they record the number of runners at each location. The runners are then collected, their memories are wiped, and they are returned to society, unaware of their role in this diabolical simulation.

The chance device looks much like a many-sided die. Each of the faces has a single letter on it: N, S, E, or W. Depending on the whims of the overseers, the number of sides on the chance device and the number of times each letter is represented may be different. For example, the device may only have 4 sides: one for each letter. In another case, it may have 10 sides, 3 for N, 3 for S, 3 for E and 1 for W.

Assignment

In this assignment, you will create a program that simulates the progress of a group of runners, and record the result.

Step one is to create a Runner class that allows this program to run correctly:

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <random>
#include <chrono>
#include "Runner.h"

void test_runner(int n_runner, int n_steps,
                 int p_north, int p_south, int p_east, int p_west,
                 std::default_random_engine *generator)
{
    int i,j;
    for(i = 0; i < n_runner; i++)
    {
        Runner runner(p_north, p_south, p_east, p_west, generator);

        for(j = 0; j < n_steps; j++)
        {
            runner += 1;
        }
        std::cout << "Runner: " << runner.getX() << " " << runner.getY() << std::endl;
    }
}

int main(int argc, char *argv[])
{
    if(argc < 7)
    {
        std::cout << "usage: " << argv[0] << " num_runner num_steps prob_north prob_south
        prob_east prob_west" << std::endl;
        std::exit(1);
    }
    int n_runner = std::atoi(argv[1]);
    int n_steps = std::atoi(argv[2]);
    int p_north = std::atoi(argv[3]);
    int p_south = std::atoi(argv[4]);
    int p_east = std::atoi(argv[5]);
    ```
int p_west  = std::atoi(argv[6]);

unsigned int seed = std::chrono::system_clock::now().time_since_epoch().count();
std::default_random_engine generator(seed);

test_runner(n_runner, n_steps, p_north, p_south, p_east, p_west, &generator);

return 0;
}

This Makefile is useful in building and running the test program:

===============================================
CXXFLAGS := -std=c++11 -Wall -Werror
all: test
test: runner_test
    ./runner_test 10 5 1 1 1 1
Runner.o: Runner.cpp Runner.h
runner test.o: runner test.cpp Runner.h
runner test: runner test.o Runner.o
    g++ $(CXXFLAGS) -o $@ $(^)
clean:
    -rm -f *.o *~
spotless: clean
    -rm -f runner_test
clean
===============================================

Potential Session

Example run of the runner test

=================================================================
$ make -f Makefile
g++ -std=c++11 -Wall -Werror -c -o runner test.o runner test.cpp
g++ -std=c++11 -Wall -Werror -c -o Runner.o Runner.cpp
g++ -std=c++11 -Wall -Werror -c -o runner test runner test.o Runner.o
    ./runner test 10 5 1 1 1 1
Runner: 1 2
Runner: 0 1
Runner: 0 1
Runner: -1 0
Runner: 0 -1
Runner: -2 -1
Runner: -1 0
Runner: 0 1
Runner: 0 1
=================================================================

Show Off Your Work

To receive credit for this assignment, you must upload the source code (.h and .cpp files) and the Makefile to the
CIT submission system linked at the top of the page.

Additionally, the programs must build, run and give correct output.