CSCI 689 **Computer Assignment 5** **Fall 2017**
(10 points)

Our Friend the Atom

Write a C++ program to simulate the following experiment, which was included in the 1957 Disney film *Our Friend the Atom*, to illustrate the chain reactions involved in nuclear fission. The setting for the experiment is a large cubical box, the bottom of which is completely covered with 625 mousetraps arranged to form a square with GRID_SIZE = 25 mousetraps on a side. Each of the mousetraps is initially loaded with two ping-pong balls. At the beginning of the simulation, an additional ping-pong is released from the top of the box and falls on one of the mousetraps. That mousetrap springs and shoots its two ping-pong balls into the air. The ping-pong balls bounce around the sides of the box and eventually land on the floor, where they are likely to set off more mousetraps.

In writing this simulation, you should make the following simplifying assumptions:

- Every ping-pong ball that falls always ands on a mousetrap, chosen randomly by selecting a random row and column in the grid. If the trap is loaded, its balls are released into the air. If the trap has already been sprung, having a ball on it has no effect.

- Once a ball falls on a mousetrap, that ball stops and takes no further role in the simulation.

- Balls launched from a mousetrap bounce around the room and land again after a random number of simulation cycles have gone by. That random interval is chosen independently for each ball is always between MIN_CYCLES = 1 and MAX_CYCLES = 4 cycles.

Your simulation should run until there are no balls in the air. At that point, your program should report have many time units have elapsed since the beginning, what percentage of the traps have been sprung, and the maximum number of balls in the air at any time in the simulation.

In the main ( ) routine, define the mousetrap-grid as a two-dimensional vector container of type bool with the size GRID_SIZE, and define another vector container of type unsigned to keep track of the remaining cycles of the balls in the simulation. For each ball, the initial assigned cycles are between MIN_CYCLES and MAX_CYCLES, and this number is decremented by 1 after each cycle. The initial value of the simulation clock is 0 and it's incremented by 1 after each cycle.

The main ( ) routine calls the following subroutines:

- void init_sim (vector < vector < bool > >& traps, vector < unsigned >& ballCycles): This routine initializes each mousetrap in the traps grid to logical value true, it means that none of the mousetraps have been sprung yet. It also initializes the RNG by calling the srand ( ) function with the seed value SEED = time (0), so each time

you run your program, you'll get different output values but they all be in the proximity of each other. Since the first ball is release at the beginning of the simulation, this routine also inserts the cycle 1 for the first ball in the vector ballCycles.

- unsigned release_balls (vector < vector < bool > >& traps, vector < unsigned >& ballCycles): This routine is called in each simulation cycles, and it takes the necessary steps in that cycle. It checks the value of each element in the vector ballCycles, and if the value is not 0, then it decrements that value by 1, and the final value becomes 0, it randomly selects a mousetrap in the grid trap to fall the ball on it. If the chosen mousetrap has not been sprung yet, it computes the random cycles for each of its two balls by using the expression rand ( ) % (high – low + 1) + low, where low and high are the minimum and maximum cycles for a ball, respectively, and changes the logical value of that mousetrap to false in the grid traps. At the end, this routine compute and returns the total number of balls in the air to the main ( ) routine, and the main ( ) routine can update the maximum number of balls in the air by this value. To update the vector ballCycles, the auxiliary routine update_cycles ( ) can be called for each of the two balls of a mousetrap.

- void print_grid (const vector < vector < bool > >& traps, const unsigned& clock, const unsigned& noBallsInAir): This routine is called in each simulation cycles, and it prints out the current values of the simulation clock and the total number of balls noBallsInAir in the air on stdout. After each PRN_CYCLES = 10, it also prints out each corresponding logical value of the grid traps by printing the character 'X' for the logical value true and the character '.' for the logical value false. The main ( ) routine also calls this routine at the beginning and end of the simulation.

- void print_stat (const vector < vector < bool >& traps, const unsigned& maxNoBallsInAir, const unsigned& clock): This routine is called at the end of the simulation. It prints out the total simulation time clock and the maximum number of balls in the air maxNoBallsInAir on stdout. It computes the total number of mousetraps have sprung in the grid traps in the simulation and print out the percentage of sprung traps.

- void update_cycles (vector < unsigned >& ballCycles): This routine is called to update the vector ballCycles for a ball. It computes the initial random cycles for the ball and searches the vector for the first cycle number 0. If the number is found, then it replaces the number with the initial cycles of the ball; otherwise, it inserts the cycle number at the end of the vector.

To compile and link the object file of your program with the system library routines, execute Make N=5. To test your program, execute: Make execute N=5. This command executes your program and displays the output both on the terminal screen and in the output file prog5.out. You can find a correct output file in directory: ~cs689/progs/17f/p5.

After you're done, you don't need the object and executable files any more. To delete them, execute: Make clean.

Submit your source and header files of your program to your TA by executing: mail_prog.689 prog5.cc prog5.h.

If your program does not work properly, then change the constant values used in your program to some other values to debug and trace the steps. For example, instead of using GRID_SIZE = 25, use a smaller value, like 2 or 3, and for the SEED of the RNG, use a fixed seed value, such as 1, instead of a time-dependent value. In that case, for each simulation run, you'll get the same output values. This clearly makes the debugging of your program much easier.