CSE 6010
Assignment 4
Collisions

Initial Submission Due Date: 11:59pm on Thursday, October 26
Final Submission Due Date: 11:59pm on Thursday, November 2
Peer review and code reflections assignments due November 9 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 simulate collisions of particles in a two-dimensional box. Particles can collide with the walls and with each other. Here, you will use an event-driven approach by calculating the times of all possible interactions and allowing the earliest collision to occur. Compared to the alternative approach of updating all particles' positions over short time intervals and checking for collisions, the event-driven approach can offer more accuracy by computing exact collision times as well as more efficiency by waiting until a collision happens to update a particle's position.

Your task is to write a program in C to simulate particle collisions. Specifically, you will calculate the times of all possible collisions. Then, for as long as the time is less than some user-specified end time, you will sort the possible collisions according to the collision time, allow the earliest collision to happen, and update the times of possible collisions involving the particle(s) that just collided. At the end, you will update all particle positions to the specified end time.

Note that you may test your code against the autograder for the final submission; you may submit as many times as you like. Please note that you still must submit your initial submission on time.

Specifications:

- Your code should ***read in the initial particle information from a file along with other key values***. The first line of the file will specify the number of particles (N); you should assume the particles are represented as integers from 0 to N-1. The second line will list the radius of the particles. The third line will specify the length of the box in the x-direction followed by the y-direction. Each following line of the file will specify the following information for particles 0 to N-1, one per line, in order: x-position, y-position, x-velocity, and y-velocity, which should all be stored as doubles. A small test file will be provided; other files will be used to test your final submission, but you should also generate some test cases (e.g., to test for collisions with each wall as well as for particle-particle collisions). You may assume that the values and file format are valid.
- The name of the file containing the initial particle information should be read as the ***first command-line argument***. You do not need to perform any validation and may assume that the user will specify a valid filename.
- The ***second command-line argument*** should be a double specifying the simulation end time. Assume the simulation begins at time 0. You may also assume that the input time will be a positive value; validation is not required. Collisions should be processed until the current collision time would be greater than the end time. At the end, the positions of all particles will be updated to the end time, as detailed below.
- Your code should ***define two structures:*** one to represent a particle and one to represent a potential collision. Each particle will need to store its position and velocity and the time

these values were last updated. In addition, to aid in debugging, each particle should keep track of the number of collisions it has with walls (lump together collisions with all four walls) and the number of collisions it has with other particles. Each potential particle-particle collision should include the two particles involved and the potential collision time; each potential particle-wall collision should include the particle involved and a representation for each wall (for example, you could use various negative integers to represent walls).

- You should define an array of particles to represent the N particles and an array of potential collisions. Assume that all particles have non-zero x- and y-velocity components. Thus, each particle will have 2 potential wall collisions (one with the left or right wall and one with the upper or lower wall), one of which will be earlier than the other, and you can just store the earlier. In addition, every particle potentially could collide with every other particle, for a total of N*(N-1)/2 potential particle-particle collisions. Thus, there will be a total of N+N*(N-1)/2 = N*(N+1)/2 potential collisions. Memory for these arrays should be ***allocated dynamically***.
- You will initially calculate *all* potential collision times. The information at the end of this document specifies how to do so for both particle-wall collisions and particle-particle collisions. Some collisions will not be possible. You should set the potential collision time in that case to a large value that will ensure the collision will not occur in the program.
- Once all potential collisions times have been calculated, you should ***sort the array of potential collisions***. (Note that you could in principle use a priority queue; however, for this assignment you are asked to use sorting instead.) You may use any sorting method you like; we recommend insertion sort because it is easy to implement and because the array will be "somewhat" sorted after the first sort, so that insertion sort should be efficient.
- ***"Process" the earliest potential collision:*** update the current time to the collision time and update the position, velocity, last update time, and the count of the relevant collision type for each particle involved. The information at the end of this document specifies how to update the particle position and velocity values for both particle-wall collisions and particle-particle collisions.
- After processing the collision, scan through the array of potential collisions and ***update the times*** for all potential collisions involving the particle(s) that participated in the most recent collision that was just "processed."
- After the last collision has been processed, advance all particle positions to the time of the last collision (so all particle positions are at the same time), then output the ***final position and the counts of wall and particle collisions (in that order) for each particle to the screen***. Use a format statement like the following for each particle (adjust for your variable names, of course—the point is to use the precision listed, commas and spaces as indicated, and one particle per line).
```
printf("%.6f, %.6f, %d, %d\n", particles[i].xpos,
particles[i].ypos, particles[i].cwall, particles[i].cpart);
```
- All ***dynamically allocated memory should be freed*** (consider using *valgrind* to test for memory leaks or, if you use MacOS, you may wish to try *leaks*).

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.

Test files of initial particle positions/velocities are provided for you; we will check results for other sets of initial particle positions/velocities. In particular, we are including test cases with only particle-wall collisions and with only particle-particle collisions so that you can test these capabilities separately.

Final submission: You should submit to Gradescope through Canvas the following files:

(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***.

(2) the **Makefile** you use to compile your program such that your executable file that will run your program is named '**particles**'. You may use the Makefile provided for previous assignments as a starting place.

(3) a series of 2-3 slides **saved as a PDF** and named **slides.pdf**, structured as follows:

- Slide 1: your name and a brief explanation of how you developed/structured your program. This should not be a recitation of material included in this assignment document but should focus on the main structural and functional elements of your program (e.g., the purpose of any loops you used, the purpose of any if statements you used to change the flow of the program, the purpose of any functions you created, etc.). The goal is to help us understand how you developed your code and your intentions. You are limited to one slide.
- Slides 2-3: a discussion of why you believe your program functions properly, with reference to at least one test case for a particle-wall collision and at least one for a particle-particle collision.

Initial submission: Your initial submission does not need to include the slides or Makefile. It 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.

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.
- Consider implementing just particle-wall collisions first and then adding in particle-particle collisions later.
- Or, consider implementing both particle-wall collisions and particle-particle collisions for just the initial time, then add in the loop and collision time updates later.

Updating potential collision times and colliding particle positions and velocities

The remainder of this document details how to calculate the times for potential collisions as well as how to update the positions and velocities of particles involved in collisions. Don't be alarmed by the length! The exposition is intended to be comprehensive, but what you actually need to implement is fairly small and straightforward. Look for **boldface red text** that specifies the actual update equations to be solved.

For the exposition below, the following notation is used. (The choices here are a mathematical convenience; you do not need to use the same notation.)

(x_i, y_i): position of particle i (if distinguishing particles is not needed, subscripts will be dropped)

(u_i, v_i): velocity of particle i (if distinguishing particles is not needed, subscripts will be dropped)

When needed, we will introduce a functional notation: e.g., $x(t)$ represents the x-position of a particle at time t.

R: radius of each particle

L_x: position of right edge (length in the x-direction)

L_y: position of right edge (length in the y-direction)

You may assume that each particle has non-zero velocity components in both directions: e.g., $u \neq 0$ and $v \neq 0$. Also, the origin is assumed to be located at the lower left corner (e.g., $y = 0$ at the lower boundary). Thus, the walls of the box are at $x = 0$, $x = L_x$, $y = 0$, and $y = L_y$.

Calculation of particle-wall collisions:

In the absence of any other interactions, a particle will travel using the speed and direction given by its velocity until it encounters a wall. We will consider collisions with the left and right walls first.

Collisions with the left and right walls are governed by u: if $u > 0$, the particle is moving to the right and the collision would occur with the right wall; if $u < 0$, the particle is moving to the left and the collision would occur with the left wall.

Collisions occur when a particle comes within R of either wall. Thus, a collision with the left wall would take place when $x = R$ and a collision with the right wall would take place when $x = L_x - R$.

We can express a particle's path over time when we know its position at time t_1 using the following (parametric) description:

$$x(t) = x(t_1) + u * (t - t_1); \quad y(t) = y(t_1) + v * (t - t_1).$$

We will first consider the case of potential collisions with the left and right walls. We wish to solve for the time that results in the particle having x-position of either R (in the case that the particle is moving to the left) or $L_x - R$ (in the case that the particle is moving to the left). Therefore, the **collision time at the left edge**, t_L, for a particle moving left would be given by

$$t_L = t_1 + \frac{R - x(t_1)}{u}.$$

Similarly, the **collision time at the right edge**, t_R, for a particle moving to the right would be given by

$$t_R = t_1 + \frac{(L_y - R) - x(t_1)}{u}.$$

Note that $R \leq x(t) \leq L_y - R$ for all t (that is, the particle cannot get closer to either wall than a distance of R from it), so t_L is in the future only for $u < 0$ and t_R is in the future only for $u > 0$.

When calculating potential collisions, you will only need to record the time of the potential collision, because the collision actually may not occur, so it is more efficient to wait to update positions and velocities. You may choose to have logic that calculates only the appropriate collision time based on the sign of u or you may calculate both options and keep only the collision information for the future collision.

If a particle actually collides with the right or left wall (i.e., it is the earliest collision of those considered), the particle's **position will be set** to the position it would have at that time (e.g., the x-position should be R or $L_y - R$ depending on the sign of u), and its **velocity will be updated** by reversing the sign of u (the x-component of the velocity).

For collisions involving the top and bottom walls, everything is the same after replacing x with y and u with v.

Calculation of particle-particle collisions:

Below, we build on the explanation of particle-wall interactions.

We will define two particles as colliding when they just touch—that is, the distance between the centers of the two particles is twice their radius, $2R$. To find the time of such an event, we begin geometrically by defining the circle with radius R (diameter $2R$) that represents when the two particles just touch, as we will want to find the time that this event occurs. We find the center of this circle using the average of the x positions of the two particles and the average of their y positions, so that the center is given by $\left(\frac{1}{2}\left(x_1(t_c) + x_2(t_c)\right), \frac{1}{2}\left(y_1(t_c) + y_2(t_c)\right)\right)$, at some collision time t_c that we seek. Note that the general equation for a circle with radius R and center (x_{cen}, y_{cen}) is given by

$$(x - x_{cen})^2 + (y - y_{cen})^2 = R^2.$$

The particles thus are on the circle representing when they are just touching when the following condition is satisfied:

$$\left(x_1(t_c) - \frac{1}{2}\left(x_1(t_c) + x_2(t_c)\right)\right)^2 + \left(y_1(t_c) - \frac{1}{2}\left(y_1(t_c) + y_2(t_c)\right)\right)^2 = R^2.$$

Simplifying gives

$$\left(\frac{1}{2}\left(x_1(t_c) - x_2(t_c)\right)\right)^2 + \left(\frac{1}{2}\left(y_1(t_c) - y_2(t_c)\right)\right)^2 = R^2$$

or

$$\left(x_1(t_c) - x_2(t_c)\right)^2 + \left(y_1(t_c) - y_2(t_c)\right)^2 = 4R^2.$$

We can express the position of a particle at an arbitrary time if we know its velocity and its position at some known reference time t_r. We will use the current time (the time of the most recent previous collision) as the reference time. Thus, we will need to update the positions of the two particles to the current time to perform the calculation. We will denote the updated positions as (x_{up1}, y_{up1}) and (x_{up2}, y_{up2}). Note that each particle's position can be updated using the expressions above; in this case, we update from the last updated time to the current time as $x(t_r) = x(t_{last}) + u * (t_r - t_{last});\quad y(t_r) = y(t_{last}) + v * (t_r - t_{last})$.

Thus, continuing the calculation of the potential collision time, we have:

$$x_1(t_c) = x_1(t_r) + u_1\,(t_c - t_r),\quad y_1(t_c) = y_1(t_r) + v_1(t_c - t_r),$$
$$x_2(t_c) = x_2(t_r) + u_2\,(t_c - t_r),\quad y_2(t_c) = y_2(t_r) + v_2\,(t_c - t_r).$$

Substituting these expressions into the above gives

$$\big(x_1(t_r) + u_1(t_c - t_r) - x_2(t_r) - u_2(t_c - t_r)\big)^2 + \big(y_1(t_r) + v_1(t_c - t_r) - y_2(t_r) - v_2(t_c - t_r)\big)^2 = 4R^2,$$

which is just a **quadratic equation in t_c that can be solved** using the quadratic formula. (You may find it simpler to solve for $t_c - t_r$ and add t_r to the result to obtain t_c.) If the discriminant (the value under the square root) is negative, no interaction is possible between the particles. (For example, the particles may be moving away from each other and not intersect at some future time.) If the discriminant is nonnegative, use the *earlier* time that will occur by choosing the smaller of the two roots. (The two roots would occur at the early point of tangency—the time we desire—and as a second point of tangency post-collision, if the collision did not affect the trajectories of the particles.)

In the event that a collision is not possible, choose a value that will ensure it is never selected as the next collision.

If a particle-particle collision actually happens, their **positions will be updated** by setting them to the values given by $(x_1(t_c), y_1(t_c)$ and $\big(x_2(t_c), y_2(t_c)\big)$—that is, let them continue along their current trajectories as specified by their current velocities until the collision time.

To update the velocity, we assume a perfectly elastic collision, where the kinetic energy remains the same, and also assume the particles have the same mass. Essentially, we will have the two **colliding particles switch velocities**. Thus, after the collision,

$$u_1(t_c) = u_2(t_r),\qquad v_1(t_c) = v_2(t_r)$$

and

$$u_2(t_c) = u_1(t_r),\qquad v_2(t_c) = v_1(t_r).$$

(Here we assume that no collisions have occurred since t_r, which will be the case if you set t_r to be the time of the most recent previous collision, since the current collision being "processed" will the next.)

Some verification hints: Note that you can check that the time you predict for a collision is correct.

- For a particle-wall collision with the left or right wall, calculate the new x-position of the particle by adding the product of the elapsed time from the particle's last update to the predicted collision time with the particle's x-velocity. The result should have the position located one particle radius R away from the left or right wall. The same goes for collisions with the upper and lower walls, just use the y-position and y-velocity.
- For a particle-particle collision, calculate the new x- and y-positions of both particles by evolving them to the collision time as described above for a particle-wall collision (essentially, let them follow their current trajectory to the collision time). Then use the new positions of the particles at the time of collision to calculate the distance between the particles at the time of collision. The distance should be twice the particle radius ($2R$).