First, we will go over understanding the problem, which is important to understand what is needed for code. The task was to compute the occlusion angle of random placed and randomly rotated rectangles. The occlusion angle is if standing at the origin and looking outward, how much of the full 360-degree circle is blocked by the rectangle. The overall goal is to generate these random rectangles, calculate their angular coverage, merge where they overlap, and report the total angle covered.

Then over the algorithm approach, which is a bit more difficult to article. Starting with the random rectangle generation. First is making them set the center of that rectangle (hypothetically [x,y]), width, height, rotation angle. Then position will need to be chosen at random in a bounded region (like [-100, 100]). Then the rotation must be chosen from a uniform standard of [0,360). Then corner calculation, which is to computer the four axis corners relative to the center of the rectangle. Apply that rotation (2d) matrix with the following formula: $[x^1] = [\cos\theta , -\sin\theta] [dx]$, $[y^1] = [\sin\theta , \cos\theta] [dy]$, this will tell the algo how the rectangle corners need to be rotated to what coordinates.

Converting the angles, which means that we must compute the polar angle using atan2(y, x). Then I need to go back to normalizing angle in the aforementioned standard of [0, 360). Then record down the minimum and maximum angle across corners to form an interval. Then merge the intervals, where we collect the intervals from all the generated rectangles. Then sort the starting angle, then merge that overlap intervals to avoid double counting. Then compute the total covered angle equaling to the sum of merged interval lengths.

Implementation was using C++ used for generation + interval merging (fast, and easy). The output was exported to csv for plotting in python (matplotlib). Visualization used matplotlib, confirming the correctness.

Complexity Analysis:

- Rectangle generation: O(n).

- Corner-to-angle conversion: O(n).
- Interval merging (sort and linear scan): O(nlogn).
- Overall Total: O(nlogn).
- Efficient for large n.

I would like to figure out in the future how to use matplotlib for c++ for my output in the future. I would like to also like to give myself to figure out more ways to do this with c++ that might help me better understand the language going forward.