a. **A concise description of how the program works. This should be a couple of paragraphs, describing your overall approach at a conceptual level. Some questions to consider: What choices did you make in designing the program, and why? What specific parameters are part of your method, and how did you select values for those parameters?**

Our initial approach to this project also made use of two nodes, however we didn't have a great grasp of the data types and overall concepts being utilized at the time. This resulted in a lopsided design in which the subscriber_node was doing a significantly larger amount of work than the publisher_node which virtually served no purpose. Therefore we took a different approach in which the subscriber_node would be responsible for reading the lidar sensor's distance data and determining which points were moving points, corresponding to the people moving in front of the sensor, and then sending the list of all moving points to the publisher_node. To interpret which points were moving points, the subscriber_node continuously averages the points collected, when it notices that some points are repeatedly shorter than the long standing average, it decides to classify those as moving points and send them on. Publisher_node then takes the list of moving points and distinguishes them from each other, classifying them by which person they belong to. To do this, we calculate the distance from every moving point to all the other moving points and the different clusters of points are different people. We find the center of each cluster and append that to a list of people_locations. Throughout this process, we keep track of the 5 or so most recent people_locations and use that data to track them as they move across the lidar sensor. Each iteration, the most recent people_location is compared to the previous 5 and makes sure that the location of each person has not varied by too much, because that would likely be an outlier in the data.

To keep track of the people_count_total, we use a majority voting method commonly used in machine learning as well as a difference threshold for the person's displacement. The 5 most recent readings on people_locations are used to determine how many people are on the screen. If this number is greater than the current people_count_total, then we update accordingly. If a person's location does not vary much from their location in the 5 previous iterations, then we do not increment the people_count_total.

To keep track of people_count_current, we simply take the max of the majority vote or the most recent people_locations. This is because sometimes there is a split second where people_locations may be 0 due to noise in the data, but there is still a person on the screen. Averaging the 5 previous readings will account for this error.

Something clever about our design comes from exploiting the z dimension of the Point32 object available in ros. For this project, the lidar readings are projected in 2 dimensions, the x and y plane. However, Point32 objects can carry a third dimension z. In reading the data from the lidar sensor, we store the angle at which the point is relative to the camera in the z dimension of the reading. This can then be used to help us distinguish whether a person has walked behind another person and then reappeared, or if a new person has entered the screen entirely. When people cross paths, the angle at which they cross in front of the lidar sensor is

the same (within some threshold) so we can use this to determine whether or not someone entered or two people just crossed paths.

There are a few specific parameters that were utilized in different parts of the code. One of these can be seen in the find_people_points() method in which we used hardcoded threshold values of 0.3 and 0.04 in the if statement logic to determine whether a range measurement is part of a person or not. Similarly in the distinguish_people() function we see if the distance from the current point to all other points in the pointcloud is less than 0.5. If there are less than 5 points to a cluster, then we determine that the cluster is not a person, rather a couple of points of noise. To determine whether or not a person has moved too far from their previous locations, we use a threshold value of 0.8 or a theta difference between other people_locations of 0.08. These values were chosen through trial and error through the various bags of example data for the lidar. Our initial design had much looser bounds and we found that oftentimes we would drastically overshoot or undershoot the number of the people location points due to the natural error within the lidar data. Furthermore, when people would walk through the room in close proximity to one another, loose bounds in distinguishing people would oftentimes identify them as a single person. After testing with each of the example bags, we found these specific bounds allowed for the most accurate number of people points being found, as well as uniquely identifying each of the people walking through the environment.

b. A short answer to these questions: Did the results meet your expectations? Why or why not?

In our case, the results actually exceeded our expectations. When we initially started on our approach to this project, we thought the natural error present in the lidar data wouldn't allow us to get the exact right answer for each of the example problems given. This sentiment was furthered through our early experience with our hardcoded threshold values often resulting in extra people points, inconsistent tracking and misidentifying background objects as people. However, through the repeated trial and error and the redesign of our scan_callback() function we were eventually able to get the exact result and have a consistent and accurate tracking of each person in the environment. Furthermore, we ended with an efficient and meaningful distribution of work between both the subscriber_node and publisher_node which was an idea that we didn't really know how to approach early on in this project.