Architecture message types (I'm only specifying the types that are pubished... see Prof. Newman's document for all the places each of these should be subscribed to):

Note - any message any message field that is an array (such as the array of poses in PoseArray) can is actually a std::vector in C++. See std::vector for available methods of std::vector. To insert things, either use the push_back function or resize the vector to make sure it can hold all of your data and then use the standard array syntax ([n]) to stick an element at a certain position.

Also note that, because I am reusing many built-in ROS message types, you should be able to visualize them in rviz. For example, PoseArrays can be displayed in rviz, so you could see visually what the goals you are publishing are. Also, the OccupancyGrid type we are using for C-Space maps is the same format that is used for the map that shows you the second floor, so that will be quite easy to visualize as well.

- Goal publisher node:
  - Publishes: List of goals. Type: geometry_msgs/PoseArray
- PSO Node:
  - Publishes: Current odometry state. Type: nav_msgs/Odometry
- Lidar Driver Node:
  - Publishes: LIDAR scans. Type: sensor_msgs/LaserScan . Note that you you should pay attention to the angle_min, angle_max and angle_increment fields, as we will eventually put your LIDAR into a 0.5 degree mode. This will make your data better, but not every scan will start at exactly 90 degrees. range_min and range_max will also give you the valid range for values in the ranges array - any range value outside of range_min and range_max should be discarded.
- LIDAR-based Mapper Node:
  - Publishes: LIDAR-based C-Space map. Type: nav_msgs/OccupancyGrid
    - frame_id should be whatever frame the origin point is relative to.
    - origin should be the position in frame_id frame of the 0,0 pixel in the OccupancyGrid
    - data should, at this time, really only contain 100 for obstacle, 0 for free or -1 for unknown... you can add the actual probabilities later if desired.
    - See nav_msgs/MapMetaData for the meaning of other useful fields in the MapMetaData member of the OccupancyGrid.

    * You can find some sample code for reading an OccupancyGrid here: `https://github.com/cwru-robotics/cwru-ros-pkg/blob/master/cwru_semi_stable/cwru_nav/src/map_as_sensor.cpp`. The important bit is the callback from lines 46-64... note the nested for loops that calculate the x and y position for the Point p. That should give you a good idea of how data is structured in the message.
    * Huge plus for using this data type: You could just feed in the /map topic that already has a map of the 2nd floor to test a planner before you have C-Space done! They are the same exact data type!

- Path Planner Node:
  - Publishes a pathList. Type: eecs376_msgs/PathList. 'rosmsg show eecs376_msgs/PathList' to view the fields. 'rosmsg show eecs376_msgs/PathSegment' to show the fields of each segment in the PathList's list of segments
- Desired Path Crawler Node:
  - Publishes a desired state. Type: eecs376_msgs/CrawlerDesiredState. Same deal with rosmsg show for this one.
- Speed profiler node:
  - Publishes a eecs376_msgs/CrawlerDesiredState, but with the des_speed field filled in with a correct value
- Steering Node:
  - Publishes a geometry_msgs/Twist, just like your current command_publisher node