Joshua Wang CS313 HW2 Q7, 8, 9 White Paper

Q7: This question asks us to solve a maze using a stack, and thanks to the stack's pop function and lifo order, backtracking is incredibly convenient with a stack. First I had to decide how to represent a maze, and so I used a 2d array matrix to represent it, filling it with 1s to represent a valid path, 0 to represent a wall, and 2 to represent the end goal. Since solving a maze is a series of choices, I made a choice class that stores the location of choices along with the direction made. This class could be stored in our stack, and every pop would be like a backtrack of a wrong choice. All that was left of the problem was to figure out what made a valid choice, and so all 4 directions would be checked if they were a 1 or if they were not visited before. If all 4 directions are exhausted, a backtrack occurs, and this process repeats until finally we reach 2. This question made me realize how useful stacks are for "redos", because of how capable it is at accessing and deleting the last action taken.

Q8: This question asks us to make a priority queue that represents orders from clients, with 1-3 possible extra priorities. Luckily, there is a priority queue data type in c++ that immediately sorts the queue, and with a function that compares priority, we can make it sort the queue to be from highest to lowest priority. With the order class that stores priority and serial number to give a name to each order, we simply push them onto the queue and the highest priority orders will be at the top. Then we "process" the top order, calling sleep to simulate a wait for processing time, then pop them because they are done processing, and now we are able to access the next in line at top. This then repeats until all the orders are completed.

Q9: This question asks us to represent an airport with 4 runways, where 70% of planes are assigned to one runway, where 20% choose the shortest of two, and 10% can choose the shortest of three. Since each runway is supposed to hold multiple planes, they can be represented with a container, in which I chose a priority queue that would sort based on their time for takeoff. Another priority queue holds the roster for planes, so that they can be pushed onto each runway. A randomly generated variable from 1-100, eligibility, is tracked for each plane to determine what choices they can make. There are also 4 distinct time counters, separated so that takeoffs can happen at different runways simultaneously, and these counters are incremented whenever a plane is added so that the next plane can automatically be scheduled at the next hour. Each runway can then be displayed by popping each plane off the runway queues and printing them. Then to create the calendar, we take a clone of the queues that were previously popped and print them in a similar fashion, incrementing the days whenever 24 hours have been reached. After completing the program, I noticed that one flaw is that the time management could be more robust, as 1 hour gaps between every single flight is unrealistic, though I am proud it simply does what the questions asks.