Data structures description:

We created this airplane boarding system using linked lists, a doubly linked list, a queue, and a stack to efficiently manage passengers, seat assignments, and luggage.

The passenger and luggage lists are organized with a singly linked list, which stores each passenger's information (name, age, ticket, and luggage). This structure makes it easy to add and remove passengers without shifting data. This makes it good for managing dynamic lists.

For seat assignments we used a doubly linked list which allows quick navigation forward or backward through seats. This could be useful for managing changes in seating arrangements.

For the boarding and luggage processes we used a queue and a stack. The queue ensures an orderly boarding process where passengers board in the order they arrive, following a first-in, first-out structure. The stack, on the other hand, manages luggage unloading in reverse order, like a last-in, first-out process, simulating how luggage is stored and retrieved in an airplane's compartment.

a singly linked list can go ahead effectively without requiring access to elements in both directions, we utilized it to generate the passenger and luggage list. Our needs for managing a simple list that may be used in conjunction with a doubly linked list to seat people in their assigned seats are met by this data structure. Each node in a singly linked list has a data field and a reference to the node after it. The last node in the list points to null to signify the list's end. Because of its linear structure, which makes insertion and deletion operations more efficient, it can be used in a wide range of situations. By building a new node with the given value and connecting it to the existing final node, we were able to add passenger nodes to the end of the list in our implementation. The new node becomes the head if the list is empty; if not, we move through the list until we get to the last node, at which time we update the next pointer of the final node to link to the new node, so adding it to the end of the list.A while loop iterates through the list until the current pointer hits null after initializing a pointer called "current" to the top of the list. We transfer the current pointer to the next node and print the current node's data inside the loop.