THE REPORT OF ASSİGNMENT 2

Problem Definition : There are a couple of problems for this assignment. I will explain one by one.

1- Data Structures: We need to keep some data in some data structures. For example the seats of a flight must be kept in a stack or the people who want to buy ticket must be kept in a priority queue.
2- File input-output: There are two files in this problem. One is an input file for the commands, the other one is an output file for the outputs.
3- Modularity: Since there are many data type and command type, every task must be handled in a different function or file.
4- Dynamic Memory Allocation: to use the memory efficient we need to allocate the variables(except primitive types).

Methods

Variables : There are a couple of structures to keep data. For persons there are 3 structure; one of them is a struct to keep the individual person, the other one is a stack to keep persons in flights' classes, last one is a priority queue to keep people who are in queue. There are also a struct to keep a class e.g. Business, a struct to keep a flight itself and a list to keep flights.

Functions: Every data structure has it its own functions. For example stack has push function, queue has also a push function, list has push and get functions. There are 3 constructor functions. One creates a new person, the other creates a new class and the last one creates a new flight. I have 3 functions for flights. First one is "pushFlight" which adds the flight to flight lists, the other one is "getFlight" to get a flight by its name, and the last one is "sellFlight" which handles the ticket selling operations. Lastly, I have 4 function to deallocate the variables which is used during the execution of the program.

Functions implemented: All the functions are implemented but they have different names from the ones in pdf.

(Person*) NewPerson: Takes 4 arguments; person name, flight name, class of person’s, priority of person’s. Creates a new Person with these arguments.
(Class*) NewClass: Takes 2 arguments; the type of the class as integer, the capacity of the seats of class.
(Flight*) NewFlight: Takes 4 arguments; flight name, business capacity, economy capacity, standard capacity.
(void) PushList: Takes 2 arguments; pointer to the pointer of person list, pointer to the person. Adds person to the list.
(Person*) getList:Takes 2 arguments; pointer to the pointer of person list, name of person. Returns Null if person does not exists, otherwise returns the address of persons.
(void) pushStack: Takes 2 arguments; pointer to the pointer of person stack, the address of person. Adds person to the stack.
(void) pushQueue: Takes 2 arguments; address of address of queue of persons and the address of person. Adds person to the queue.
(int) indexQueue: Takes 2 arguments; address of address of queue of persons and the class id. Returns the number of persons who waits to buy tickets from that class(For example the number of people who waits for Business).
(void) pushFlight: Takes 2 arguments; address of address of the list of flights and the name of flight. Adds flight to the list.
(Flight*) getFlight: Takes 2 arguments; address of the list of flights and the name of flight. Returns the address of flight if exists, otherwise returns Null.
(void) sellFlight:Takes 1 argument which is the address of flight. First a loop checks every person in queue and tries to sell ticket to that person. For those who could not buy ticket are added to a second queue. Then this queue is traversed and a standard ticket is sold if there is place.
(void) flightFree-classFree-stackFree-queueFree : At the end of program this functions deallocate the ones that are allocated before.

Functions NOT implemented: All the functions are implemented.