Programming 2

Christian Grévisse (christian.grevisse@uni.lu)

Lab 7 – Structures

Exercise 26 – Airport #Struct #StructDeclaration #Typedef #malloc #Pointer

Write a C console application using structures to implement a basic airport management system. The program must at least contain the following functionalities:

1° Define the following structures by choosing the appropriate data types for their members:

- An airport has a name (e.g. *Paris Charles-de-Gaulle*) and a 3-letter IATA code (e.g. *CDG*).
- A flight is operated by an airline, has a flight number, an origin and destination airport, a departure time and a gate. In addition, a seat map represents the allocation of seats, i.e. a seat can either be assigned to a passenger (which can simply be modeled by her name) or left empty.

2° The flight schedule of the airport is modeled by an array containing pointers to the flight structures.

3° The airport manager can set the current airport, which is taken as origin for all flights in the schedule.

4° The airport manager can perform the following actions, proposed in a menu (which is shown again each time an operation has been done):

- add a flight
- print the schedule (with all relevant flight information)
- remove an existing flight
- show the seat map of a selected flight and print the list of passengers checked-in
- check in a passenger on a flight by selecting the row and seat number (first check if any seat allocation is still possible to avoid overbooking)
- quit the program

5° Use dynamic memory allocation to keep the memory footprint as low as possible. In concrete, you should:

- allocate the necessary memory for new flight/airport structures
- enlarge respectively shrink the memory allocated for the schedule when flights are being added or removed
  - use memory reallocation
  - upon deletion of a flight, shift all subsequent flight pointers to the left to remove the space for 1 flight pointer
- keep the space allocated for a string limited to the actual characters of a string entered by the user
- free all memory you do not need anymore (including the members of a structure before deallocating the space of a structure itself), while paying attention to dangling pointers
- all previously allocated memory must be freed before the program terminates
- quit the program if memory allocation fails

A file with an example run is available on Moodle.