Report for LAB 1

Mafalda Candal
Deadline: October 2nd 23:59

Task description

LAB 1 presented a list of conditions and set the challenge to manage the daily departures from a small airport (number of planes inputted by the user, up to a maximum of 1000). These conditions included 1) checking whether the arriving planes needed repairing (and would go to the hangar) or not (and would go to the waiting runway), 2) move the planes which were in the hangar whenever it reached the total of 5, 3) order the departure of all the airplanes in the waiting runway whenever it reached a total of 7 or when planes were about to get moved from the hangar and 4) finish the day by ensuring that both the hangar and the waiting runway were empty.

Structures and Algorithms

I recognized that I would need to use some data structure to simulate the hangar and the waiting runway. The hangar has only one exit, meaning that the last plane to go in will be the first to leave. This fundamental characteristic made me opt for a LIFO data structure, the stack. The waiting runway has a separate entrance and exit, meaning that the first plane entering the runway will also be the first one to leave. This aspect made me look for a FIFO data structure, the queue.

Both these data structures could save the plane ID, move these IDs from the hangar to the waiting runway, and be emptied whenever the limit of the hangar/waiting runway was reached.

Considering how much we discussed in class the importance of making our programs modular, I made it a top priority to do so. When I read the assignment, I listed out the names of possible functions I could create to answer each new condition: CheckRepairs(), MoveToRunway(), EmptyHangar(), etc. Most of these actions could be achieved with one or two lines of code and/or would make more sense in the function main(). For example, instead of a CheckRepairs() function, I created a big if-statement which structures my code and clearly describes what needs to happen depending on whether the plane needs repairing.

The number of listed functions started diminishing until only EmptyHangar(), EmptyRunway() and EndofDay() were left. I compared how easy it was to read the code with and without helper functions. I consider that the intuitive names of the helper functions facilitate tremendously when reading and understanding the code, and finding a particular stage of the day.

Evaluation of the program

Explain how you tested your program.

Programming in racket taught me the importance of being precise with the data types and inputs. Upper and lowercase made a difference, using 11 as a number or as a string, etc. For this reason,

my first instinct was to add many restrictions and possible error messages when the input was not what was expected. However, when I reread the section “Input and Output format” of the assignment pdf, I understood that we should assume that the input would be rigid and always have the desired format.

For this reason, my use of the “make run” command was mostly to fix the errors that VSCode signaled, check the scope of my variables, test extreme cases like 0 planes arriving at the airport, the difference between 6, 7 and 8 planes needing repairs and not (what happens when the hangar does not get full in the middle of the day, or when the runway is empty at the end of the day…), among other things.

The first versions of my program were not producing an output because I forgot to bring the hangar and runways counters back to zero when they were emptied. Later, my input was very similar to the desired, but I had a double count for “Ready for takeoff!” which was fixed by adding an if-statement before printing the statement to check whether the runway counter was not zero. Otherwise, every time the function was run and the runway was empty (for example, at the end of the day) it would still print “Ready for takeoff!”.

Conclusion and Self-Reflection

I really enjoyed the real-life characteristics of this assignment. Even though it is an extremely simplified scenario, it was still a sample of what can be achieved by using and combining the data structures we have been studying.

It was really interesting to go about choosing the right number of variables, functions and names for the desired result. It was also the first time I challenged myself to write modular code, and that required my understanding of what needs to change when part of the code is no longer in the main() and is now part of a helper function: keeping track of the scope of the variables and where I needed to declare and initialize the variables and data structures, understanding what inputs need to be used when calling the helper functions in the function main(), among other aspects.

I found it easy to understand the assignment and I felt like it was quite intuitive which data structures I would be using and the way I chose to structure my code.

The most challenging part was mostly keeping track of when I needed to use the address, the pointer or the actual variable. Specially when I was more tired, I felt that I relied heavily on VSCode’s warnings to be sure I was using the right input.

I enjoyed the application of the content to a real-life scenario, and I thought that the length of the assignment was very manageable and that writing the report gave an opportunity to reflect on the stages of creating this project and the learning moments.

I am proud of the code I submitted, and I am looking forward to getting personalized feedback in order to develop better coding habits. Thank you.