# Programming Paradigms Assignment 1

Author: Martin Raunkjær Andersen
AAU study number: 20136030
AAU email adress: marand13@student.aau.dk

## The status of the program (relative to the assignment)

All done except problem 8.

## Sources

The Swipl-Prolog environment[1] was used to develop and run the program. Besides this, no code from external sources was used.

In the *Program* folder in the deliverable, the `main.pl` is the source code of the mini project. It is loaded into the swipl environment in a unix system terminal that has the swipl environment installed like so:

```
swipl -s main.pl
```

## An enumeration of possible non-completed tasks

- Some predicates from problem 1 are not used in problem 2
- Problem 8 has not been completed (and not begun either)

## Other remarks of relevance

I have made predicates for my entities but I do not always use them because sometimes the use of them creates many duplicates when querying. That they are not used makes the program more fragile and error prone, but it still works correctly when given correct input.

To specify that a predicate should return false when one of a list of sub-predicates fail, I take the length of the list of actual results and compare it to the length of the list of results that would be expected if everything passes. It is inconceivable that there is not a better way, but I could not find it.

Even though it is written in prolog, everything is basically datalog, as there are no unsafe predicates and no recursion, except for problem seven. I could not think of a way to calculate all leg combinations between two airports without recursion, as there may be arbitrarily many legs between two airports. I pay the

[1] http://www.swi-prolog.org/

price for my transgression here, because `findLegs` results in a stack overflow if origin or destination values not corresponding to an airport code is put in. Recursion is used for almost everything in problem seven because lists are used to store the leg results of the queries, so every time some kind of condition needs to be met, like the aircraft having the same owner, the lists are looped through recursively to check each leg for the condition.