# COA256 Object-oriented Programming & Algorithms
# Assignment 1 Specifications

**Date set: 24/2/2015 Deadline: 25/3/2015 4pm**

**Flight booking system**

You are to develop a flight booking system in C++.

Flights leave from one airport and arrive at another airport. Airports are identified by a 3-letter code (e.g. EMA for EastMidlandsAirport – you can assume that there are no spaces in airport names). Each particular flight (e.g. from East Midlands to Milan) is run by a particular airline (e.g. Ryanair), takes a certain amount of time, and seats on it cost a particular fare. Two airports can be linked by more than one airline. Fares are fixed for all routes (that is, for flights between two airports run by a particular airline) and all days. In addition to the flight fare, the ticket price also includes airport departure taxes. Each airport has its own departure tax, which will be the same for all flights leaving this airport.

Your program should allow the user to query possible flight connections between two cities (for simplicity, we consider only one-way connections) on a particular day. The user should be able to enter the two cities (i.e. their airport codes) and a particular date (for simplicity, you can assume the year to have 12 months of 30 days each) and should receive a list of possible connections, each stating the airline(s), time and cost. The user should furthermore have the possibility to sort the list by flight time or by cost. If one of the flights suits the user, (s)he should be able to select the flight and "book" it. When making a booking, the complete flight information should be displayed and in addition a receipt (containing the same information) written to a file.

The data for airports, flights etc. are contained in text files. At the start of the program, you should read in these data files and populate your data structures. Data files for some airports and flights are available on learn, though note that the data provided is unlikely to be sufficient to fully test the functionality of your application, and that you might hence have to expand the data files (while maintaining the given file structure!).

Clearly, there will be cases when there is no direct connection between two cities. In such cases, the program should search for a connection via other airports (e.g. flying from East Midlands to Milan via Paris). Up to two stops (e.g. East Midlands-London-Paris-Milan) should be supported. For simplicity, we assume that flights go at times of our choosing (rather than having to worry about complete timetables), each airport however has a (fixed) connection time that will need to be taken into account when calculating total flight times (the connection times are different for different airports). Ticket costs for itineraries that contain connections are calculated as the sum of the individual flight segment prices plus the total of all airport departure taxes.

Your program should also be able to list all information that is available for a particular airport entered by the user. This information includes the airport name and code, the minimum connection time and airport tax, and the flights that leave from and arrive at the airport respectively.

While dates should be entered as numbers (e.g. 26/5/2015), the output should be more verbose with the name of the month written out (e.g. 26 May 2015).

Make sure that your program is able to cope appropriately with unexpected input (e.g. an invalid date or non-existent airport), both from the user and the data files, and that all the output is clear and neatly formatted.

Also, ensure that your code is neatly laid out, that you use meaningful names, and that you include appropriate comments in the code.

**Documentation**

Your coursework should be submitted with a report, which should (only) contain the following sections:

- Design: to contain class diagram(s) and explanations on how/why you have designed the classes that you have in your solution. Any other information/assumptions that you think useful can also go in here.
- Functionality: an indication of which of the required functionality is working and which was not completed; see the Appendix for the table to copy in here.
- Test plans: to contain evidence (e.g. in form of screen shots with accompanying explanations) that your program is indeed working correctly and the tests that you have run to determine that.

**Hand-in**

Hand-in is electronically on learn. You should submit a single ZIP archive that contains all the files of your submission. The filename of the files must be *lastname_firstname.zip* (with *lastname* and *firstname* obviously being replaced by your family and first names). Note, that only ZIP files will be accepted and that upload of other formats will result in loss of marks.

Your submitted file should contain the following:

- A filled-in coursework cover sheet.
- Your report (either in PDF or Word format; no other formats will be accepted).
- All the source files of your program.
- Visual Studio/Visual C++ 2012 project files for your application (note that these MUST be for this version of Visual Studio, i.e. the version you were using in the in N.001/002/003. Not including project files or including project files for another IDE will result in loss of marks).
- Your final test data files.
- A Windows executable that you obtained as the result of compiling your project code with Visual Studio 2012. The executable MUST be runnable on our lab machines in N.001/002/003; if it is not you will lose marks.
- Any other material you might wish to include.

(The easiest way of getting all the source and project files as well as the executable into an archive is to compile your project on one of the lab machines and zip your project folder.)

**Marking**

Marks are split according to the following scheme:

**Program design 10%**
Classes, class functionality, etc. as evidenced by your documentation in the report.

**Functionality 60%**
The more functionality you add to the program the higher your mark. Begin by getting the easier parts working and add the more complex ones (e.g. looking for flights via stops) later.

**Testing and error handling 15%**
Evidence of testing, suitable test data, error handling and error recovery.

**Output, code layout and documentation 15%**
Clear output formatting, code layout (indentations etc.), splitting into files, *useful* comments in code, documentation (report).

**Appendix**

The following is the table that, filled in, you have to copy into the Functionality part of your report.

| **Functionality** | **Y (complete)<br>P (partial)<br>N (none)** | **Comments (e.g. more details on what is not working etc.)** |
|---|---|---|
| Data import (from files) | | |
| User input (from keyboard) | | |
| List flights for an airport | | |
| Search for direct flights | | |
| Search via 1 connection | | |
| Search via 2 connections | | |
| Sorting itineraries by cost/time | | |
| Book flight/"print" ticket | | |
| Handling dates correctly | | |
| Error handling | | |