Description

Data Engineers regularly collect, process and store data. In this task you will develop a deeper understanding of how C programming language can be used for collecting, processing and storing data. In this assignment you get the opportunity to build an interactive program that can manage the list of flights departing Sydney Airport.

The list is stored as an array of `flight_t` type structures

```
    flight_t flights [MAX_NUM_FLIGHTS];
```

The `flight_t` is a structure typedef for `struct flight`. The `struct flight` contains the following fields

- `flightcode` - array of `MAX_FLIGHTCODE_LEN+1` chars (string)
- `departure_dt` - a structure of `date_time_t` type as defined below
- `arrival_city` - array of `MAX_CITYCODE_LEN+1` chars (string)
- `arrival_dt` - a structure of `date_time_t` type as defined below

Note that we now have a struct nested within a struct. The `date_time_t` is a structure typedef for `struct date_time`. The `struct date_time` contains the following fields,

- `month` - integer between 1 and 12 (inclusive)
- `day` - integer between 1 and 31 (inclusive)
- `hour` - integer between 0 and 23 (inclusive)
- `minute` - integer between 0 and 59 (inclusive)

Your program interacts with the nested struct array in your memory (RAM) and simple database file in your hard disk. It should provide the following features:

1. add a flight

Add a new `flight` to the `flights` through the terminal. You should collect the input by asking multiple questions from the user.

```
    Enter flight code>
    Enter departure info for the flight leaving SYD.
    Enter month, date, hour and minute separated by spaces>
    Enter arrival city code>
    Enter arrival info.
    Enter month, date, hour and minute separated by spaces>
```

2. display all flights to a destination

Prompt the following question

```
    Enter arrival city code or enter * to show all
    destinations>
```

The user may enter the abbreviation of `MAX_CITYCODE_LEN` characters for the arrival city. The program should display all flights to the requested destination. If the user input is `*`, display all flights.
The display format should is as follows.

```
    Flight Origin          Destination
    ------ --------------- ---------------
    VA1    SYD 11-26 09:54 LAX 11-26 18:26
```

Pay attention to the strict formatting guide:

- Flight - left aligned, `MAX_FLIGHTCODE_LEN` (i.e. 6) chars at most.
- Origin and Destination
- City - left aligned, `MAX_CITYCODE_LEN` (i.e. 3) chars at most.
- Month, day, hour, minute - two digits with leading zeros

3. save the flights to the database file

Save the `flights` in the hard disk as a binary/text file named `database`. You may use your own format to save the data. You should overwrite if database file already exists.

4. load the flights from the database file

Read the `database` file and put the data into `flights`. You may only read the data files created by your own program. You should overwrite the `flights` array you had in memory when loading from the file.

5. exit the program

Exit the interactive program.

Careless Users

Your program may assume that the input data type is always the expected type i.e. when the program expects an integer the user must enter an integer. However, a **careless user** may enter an input that is outside the expected range (but still of the expected data type). Your program is expected to handle careless users. e.g.

```
    Enter choice (number between 1-5)>
    -1
    Invalid choice
```

Or a careless user may try to add 365 as the month (month should be between 1 and 12). Or try to add a `flight` to the `flights` array when it already contains `MAX_NUM_FLIGHTS` flights, etc.
Run the sample executable to futher understand the expected behaviour.

Check the formatting of the `flightcode`

WARNING: Attempting this feature is recommended only for advanced students who enjoy a small challenge. You may need to do your own research, but more than that you may have to be creative. By using incorrect techniques you could very well introduce more bugs in your code and it could be time consuming. The special techniques required for this purpose will not be assessed in the final exam.

Your program should be able to check the format of the `flightcode`. The first two characters of the `flightcode` should be uppercase letters (A-Z) representing the airline. The rest of the `flightcode` should be numerals (0-9) representing the flight number. There must be 1-4 numerals as the flight number part of the `flightcode`. No spaces in the `flightcode`.
Run the sample executable to further understand the expected behaviour.

The `database` file
It is up to you to create your own data storage format for the `database` file. Your program should be able to read the `database` that was created by itself. You can create the `database` as a text or binary file.
You do NOT need to be able to create a `database` identical to the `database` of the sample executable. You do NOT need to be able to read the `database` of the sample executable.