Excercise 3 – Vehicle registry

Write a program to represent a vehicle registry that can store information about vehicles and drivers. The system should have a text-based UI to allow the user to manipulate the contents of the registry. Below you find a list of data that should be handled by the system as well as functionality that should be implemented.

Data

Person – This type should be used in the program to store information about the owner of a vehicle. Required information is the owner's *name* and *age*.

Vehicle – This type should be used in the program to store information about a vehicle. A vehicle should have the properties *type, brand, license plate* and *owner*. All vehicles have exactly one owner.

Registry – The program must be able to store 10 vehicles. The program should keep the information between exiting and restarting it. Due to this requirement, information must be stored in a file when the program is exited and read back in when restarted. If no file exists, a new one should be created.

Functionality

When the program is running, the user can select between the following options:

1. **Add a vehicle**. All information about the vehicle and its owner should be read in from the user and the vehicle should be stored in the registry.
2. **Remove a vehicle**. The user should enter a position and the vehicle at that position should be removed. Note that arrays start at position 0 and not 1 and how to compensate for that. Also note that not all positions are taken and hence that the user might enter a wrong position.
3. **Sorting the registry**. Sort the registry by owner name.
4. **Show information about one vehicle**. The user should enter a position and information about the vehicle at that position, including owner data, should be printed to screen. Note that not all positions entered might be okay.
5. **Show entire registry**. One vehicle per row. Think about how to present the information as readable as possible..
6. **Add random vehicle**. Automatically generate a new vehicle and add it to the registry. All fields should be randomized and contain plausible values e.g. a valid car brand, and a plausible owner (name, age etc.). The minimum randomization range for each field should result in at least 10 different possible outcomes.
7. **Search for owner**. The user should enter a string and the program responds by printing out the first vehicle whose owner name contains the string. Note that the registry must be sorted before searching, as you are required to use Binary search as the search algorithm.

If the user tries to do something incorrectly, e.g. adding a vehicle when the registry is full, a clear error message should be displayed.

Controls

Please make sure you have performed these before testrunning your program.

- Remove the registry file from disk and start the program. The program should not stop working, only show an empty registry.
- Add a vehicle when the registry is full
- Try to show or remove a vehicle on an illegal position, e.g. -1 and 13
- Try to show or remove a vehicle on an empty position, e.g. position 7 when there are only 5 vehicles in the registry
- Try to enter an unexpected input, text for age, text instead of a choice or a too long text. The program should not stop working (crash).

Code Requirements

No global variables
Variables may not be declared outside of functions, e.g.:

```
#include<stdio.h>
int antal;                    // Wrong
int main()
{
        ...
}
```

Instead, move the variable into the function:

```
#include<stdio.h>
int main()
{
        int antal;            // Correct
}
```

No magic numbers

All numbers, whose purpose is not clear from its context in the code must be defined as constants. Numbers in code is what is known as magic numbers, e.g.:

```
#include<stdio.h>
int main()
{
        int vektor[100];      // Wrong, 100 = magic number.
}
```

Rather, define a constant for the number:

```
#include<stdio.h>
```

```c
#define SIZE 100
int main()
{
      int vektor[SIZE];     // Correct
}
```

As a rule of thumb, all numbers except 1, 0 and -1 are magic numbers, but there are exceptions to that rule.

Safe Input

All input of information from the user should be safe, i.e. the program should not crash or misbehave because of it. This means that information must be read using the function `fgets`. Here is an example on how to read text:

```c
char text[32];
fgets(text, 31, stdin);
strtok(text, "\n");
```

And here is an example of how to read an integer:

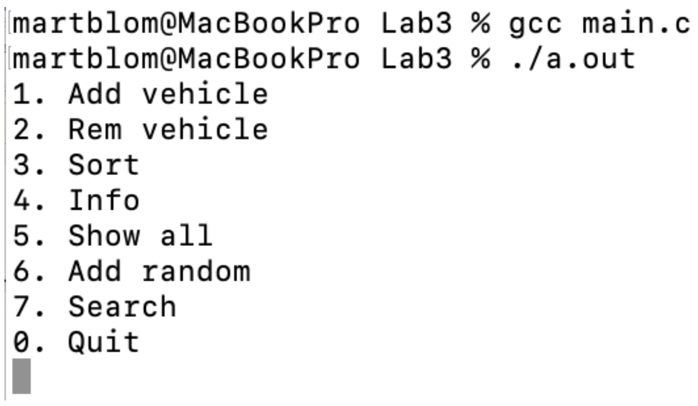
```c
char buf[32];
fgets(buf, 31, stdin);
int number = atoi(buf);
```

Modularity

The project should have at least four files: `main.c file.c, file.h` and `Makefile`. The right components should be in the right file (see lecture on modularity). The make-command should correctly compile and build the program, with separate rules for each pair of files. It is optional to have a `main.h`-file. All functions related to file handling should be in `file.h/file.c`. All other functions should be in `main.c`.

Example of process

Below you can see the process from source code file, through compilation, build and run of the program



```
martblom@MacBookPro Lab3 % gcc main.c
martblom@MacBookPro Lab3 % ./a.out
1. Add vehicle
2. Rem vehicle
3. Sort
4. Info
5. Show all
6. Add random
7. Search
0. Quit
```

: