Exercise 3 Report

David Malášek
Born: 13 May 2004

September 28, 2025

DVGA12 Programming and Data Structures
Computer Science
Faculty of Health, Science and Technology

I hereby affirm that this report represents my own independent work and that I have neither given nor received unauthorized assistance in its completion.

Contents

1 Introduction

This report represents the implementation of Exercise 3 – a car registry system written in C. The program simulates a registry storing up to ten vehicles and their respective owners. User can manipulate with the registry through text-based user interface. The aim of the assignment is to provide students with practical experience of file handling, safe input, structured data management, error handling and modularity.

2 Overview

In the text-based interface, user can choose between eight options. Seven options perform registry operations, while the eighth terminates the program. A summary of their functionality is provided below:

1. **Add vehicle**
   The program prompts the user for vehicle type, brand, license plate, owner name and owner age. A new record with this information is added to the registry.

2. **Remove vehicle**
   Program requests an index of a vehicle in the registry, the record is then removed from the registry.

3. **Sort vehicles**
   Vehicles in the registry are sorted alphabetically based on the owner name.

4. **Info about vehicle**
   The program requests an index of a vehicle. Information about this vehicle is printed.

5. **Show all vehicles**
   Program prints information about all the vehicles in the registry.

6. **Add random vehicle**
   A randomized vehicle record is created and inserted to the registry.

7. **Search vehicle**
   The program prompts the user to enter an owner's name. The input is then used to search the registry, and the first matching vehicle record is displayed.

8. **Quit**
   The program terminates.

To enhance the readability of the interface, the program highlights titles (e.g. ERROR, WARNING, OPTIONS) using color and displays the registry contents in a tabular format.

Error messages are displayed when a program fails or when invalid input is provided. Warning messages are issued in the case of non-critical problems that are not directly caused by user input.

3 Detailed Description

3.1 Data Structures

Program defines primary data structures: `vehicle` and `person`. Structure `vehicle` contains three string attributes `type`, `brand`, `license_plate` as well as a `person` structure, which represents the owner. The structure `person` contains a string property `name` and an integer attribute `age`. This way a vehicle is associated with exactly one owner, which is required by the assignment.

```
typedef struct {
    char *type;
    char *brand;
    char *license_plate;
    person owner;
} vehicle;

typedef struct {
    char *name;
    int age;
} person;
```

Listing 1: Definition of data structures

3.2 Registry

From a storage perspective, the registry is implemented as a plain CSV file. Each value is separated by a comma and every record is stored on a separate line.

```
Hatchback,Skoda,TDL7231,Michael Jackson,40
Cabrio,Skoda,PNT6810,John Does,37
Electric,Tesla,UYL7990,Mike Spirit,29
SUV,Skoda,QHJ5220,Bob,75
```

Listing 2: Sample contents of the registry

3.3 Main functionalities

This section provides a detailed description of the logic implemented for each option.

1. **Add vehicle**

   First, the function verifies whether the registry has reached its maximum capacity. If the registry is full, the execution ends. Otherwise, the program proceeds to input validation for each field. The program prompts the user to enter specific value (eg., vehicle type) and the input is read using the `fgets()` function.

   For each input there is a two step validation process. If the input was read successfully, the function first verifies whether the value lies within the specified limits. For strings, this refers to the maximum number of characters, while integer values are validated against specified minimum and maximum limits. Second, the function checks whether the string input consists only of allowed characters.

   The program repeatedly prompts the user to enter a license plate until a unique value is provided for the registry.

   Following code illustrates the retrieval and validation of the `type` value:

   ```
   printf("\nType:␣");
   vehicle.type = read_string();
   if (!vehicle.type) {
       fancy_print("ERROR", RED);
       printf("Could␣not␣read␣the␣vehicle␣type.\n");
       return 0;
   }
   if (strlen(vehicle.type) > TYPE_LIMIT) {
       fancy_print("ERROR", RED);
       printf("Entered␣vehicle␣type␣exceeded␣the␣
          allowed␣length␣of␣%d␣characters.\n",
          TYPE_LIMIT);
       return 0;
   } else if (!is_valid(vehicle.type, TYPE)) {
       fancy_print("ERROR", RED);
       printf("Invalid␣vehicle␣type.␣Use␣only␣
          alphabet␣letters.\n");
       return 0;
   }
   ```

   Listing 3: Retrieving and validating vehicle data

The same procedure is applied to all string fields. The `age` field is validated only against its allowed numerical range.

After successful validation, the values are stored in the registry using a dedicated function:

```c
int write_to_registry(char *path, vehicle v)
{
    FILE *file = fopen(path, "a");
    if (!file) return 0;

    fprintf(file, "%s,%s,%s,%s,%d\n", trim(v.type)
       , trim(v.brand), ...);
    fclose(file);
    return 1;
}
```

Listing 4: Writing to the registry

Each value is trimmed to remove leading and trailing whitespace before being stored to the registry. The above example shortens the `fprintf()` statement for readability.

2. **Remove vehicle**

   The function first verifies that the registry is not empty. Then it prompts the user to enter the index[1] of a vehicle to be removed. If the index is valid, the function then proceeds to the deletion.

   The removal process is implemented using a temporary file. All lines are copied from the original registry file into the temporary file, except for the line with the specified index. After the copying is completed, the original file is replaced with the temporary file.

[1]Although the program expects the user to provide a one-based index, the internal logic operates on a zero-based index.

```c
int delete_line(int line_index)
{
    FILE *in = fopen("./data/registry.csv", "r
       ");
    FILE *out = fopen("./data/temp.csv", "w");
    if (!in) return 0;
    if (!out) {
        fclose(in);
        return 0;
    }
    char buffer[(INPUT_LIMIT - 1) * 5 + 4 + 1
       + 1];
    int current_line = 0;
    while (fgets(buffer, sizeof(buffer), in))
       {
        if (current_line != line_index) {
            fputs(buffer, out);
        }
        current_line++;
    }
    fclose(in);
    fclose(out);
    remove("./data/registry.csv");
    rename("./data/temp.csv", "./data/registry
       .csv");
    return 1;
}
```

Listing 5: Deleting a line in the registry

3. **Sort vehicles**

   First, the function verifies whether the registry is empty or contains only one record. In both cases sorting is unnecessary, and the function ends. Otherwise, the entire registry is loaded into an array of `vehicle` structures using `get_vehicles()` function, which is shown below:

```
vehicle *get_vehicles()
{
    int vehicle_count = count_lines();
    vehicle *vehicles = malloc(vehicle_count *
        sizeof(vehicle));
    if (!vehicles) return NULL;
    int i = 0;
    char *current_line;
    while (i < vehicle_count) {
        current_line = get_line(i);
        vehicles[i] = get_data_from_line(
           current_line);
        free(current_line);
        i++;
    }
    return vehicles;
}
```

Listing 6: CSV line to data structure transformation

The sorting procedure is implemented using the bubble sort algorithm. Adjacent elements are compared by comparing the owner names with the `strcmp()` function, and their positions are swapped if required.

```
while (i < vehicle_count - 1) {
    e = 0;
    while (e < vehicle_count - i - 1) {
        if (strcmp(vehicles[e].owner.name,
           vehicles[e + 1].owner.name) > 0) {
            vehicle temp = vehicles[e];
            vehicles[e] = vehicles[e + 1];
            vehicles[e + 1] = temp;
        }
        e++;
    }
    i++;
}
```

Listing 7: Bubble sort algorithm

After sorting, the program writes the reordered vehicles to a temporary file. The original registry file is then removed and replaced with the sorted version. Because the `get_vehicles()` function dynamically allocates memory for the array of `vehicle` structures, this memory is freed at the end of the function.

4. **Info about vehicle**

   First, the program verifies whether the registry is not empty. Then it prompts the user to enter an index, which is then validated. If the index is valid, the program retrieves the corresponding record and displays all information about the vehicle and its owner in the terminal.

5. **Show all vehicles**

   The function first verifies that the registry is not empty. It then iterates through the registry and prints information about each stored vehicle and its owner.

6. **Add random vehicle**

   The function first verifies that the registry is not full. If not, it creates a new vehicle by randomly selecting the type, brand, and owner name from predefined data sets, generates[2] a unique license plate in the `AAA0000` format, and assigns a random owner age within the range of 18-85. The randomly generated vehicle is then added to the registry.

   The random number generator is seeded once at program start with `srand(time(NULL))`[3], ensuring that each program execution produces a different sequence of random values.

7. **Search vehicle**

   If the registry is empty, the function terminates with a warning message. If it contains a single vehicle, the record is displayed immediately. In other cases, it prompts the user to enter a query corresponding to the owner's name. The input is then validated.

   The assignment requires the usage of binary search. A binary search is a *divide-and-conquer* algorithm that requires the array to be sorted prior to searching. The algorithm is called binary because it repeatedly splits the array into two halves. It begins by comparing the target value with the middle element of the array, and then continues the search in the half where the element may be located [1].

[2]License plate generation is implemented within a `do-while` loop to ensure that each generated plate is unique within the registry.

[3]The `srand()` function seeds the random number generator with the current time ensuring `rand()` gives different results each run.

In this case, the query string is compared with the owner names stored in the registry. If a match or substring occurrence is found, the corresponding vehicle information is displayed. If no match is found, the program displays a warning message.

A dynamically allocated memory used for storing the vehicle array is freed at the end of the procedure.

8. **Quit**

   The program terminates.

3.4 Program structure

To ensure a modular architecture, the program is divided into four C source files (`main.c`, `file.c`, `functions.c`, `utils.c`), one header file (`file.h`), and one CSV file (`registry.csv`).

All C source files are located in the `src/` directory. The file `main.c` contains the main loop, which displays the available options and retrieves the user's selection.

The implementation of each option is contained in `functions.c`. All file operations are handled in `file.c`, while helper functions, such as `is_valid()` and `count_fields()`, are located in `utils.c`.

The single header file, `file.h`, is placed in the `include/` directory. It contains all required library imports, constant definitions, structure declarations and function prototypes.

The registry data are stored in the `data/` directory as `registry.csv`. Finally, the `Makefile` is located in the root directory.

4 Lessons learned

Since I had some prior experience with C programming, I did not encounter major obstacles during this project.

One important new skill for me was handling user input during program runtime. Previously, I had only worked with input using `argc` and `argv`. Accepting input during runtime required making decisions about buffer sizes and safe input handling. I eventually set the input limit to 32 characters, reserving the final position for the null terminator. This restriction also applied to the program's value limits: each attribute of a vehicle (type, brand, ...) was given its own maximum length. The program first verifies that the input does not exceed the buffer capacity, and only then checks whether it respects the allowed limit. From these processes I learned the importance of preventing buffer overflows and ensuring safe handling of user input.

Another new area for me was file handling in C. After researching possible data storage approaches, I determined that a CSV file was the most suitable solution for this project. The two main challenges I faced were implementing deletion of a specific line in the file and reading and parsing data from it.

Finally, regarding the technical restrictions defined in the assignment, I had no difficulty avoiding the use of global variables nor magic numbers.

5 Conclusion

This report has presented the implementation of Exercise 3, a vehicle registry program written in C. The project was designed with a focus on clarity and maintainability of code, supported by meaningful comments and a modular structure.

Special attention was given to handling edge cases and error checking, so that the program performs reliably under a variety of user inputs.

Lastly, the interface was designed to provide a smooth user experience, balancing functionality with readability.

References

[1] BBC Bitesize. Binary search, n.d. Accessed: 23 September 2025.