COMP 1631 Assignment 6 ("Car Database")

Given: Monday, November 30, 2020
Due Date: Wednesday, December 9, 2020 before 11:59 pm sharp
The late policy is NOT in effect for this assignment – so ABSOLUTELY NO LATES ACCEPTED!

Objectives

- to learn about using structures and arrays of structures as a way to consolidate mixed data types
- to practice designing and writing a variety of functions, using both value and reference parameters
- to learn to read, understand and modify an existing program written by someone else
- to continue to develop problem solving skills
- to practice incremental program design and testing

Introduction

You have been hired to finish a project by a previous developer. That developer was approximately 50% done with a license plate database program when they won the lottery and moved to Iqaluit. You must finish up the program by completing its last four functions.

The Problem

The company is not willing to restart the project and wants you to complete the previous developer's design. **You are not to re-write any of the existing program, but simply complete the functions that are missing!**

As a first step, you should review the incomplete program, `asg6_start.cpp`, to understand what has been done and how. In addition, you can compile and run the program to see how the menu system operates. As you will see, the *list all* and *search* functions are incomplete; these are two of four functions you will be writing.

Car License Data

Once you have a feel for how the program operates, you need to understand its data. Each car license is stored in a structure, and the program collects these car licenses together in an array. The array can hold up to 100 licenses (as defined by the `MAX_SIZE` constant). In addition, a fill level variable is kept alongside the array to hold the number of licenses in the array. Each license has an associated license plate, owner, make and model. The license plate is stored as a c-string of length 7 (8 including the null terminator). A license plate can either be 6 or 7 characters in length and the characters are guaranteed to be a combination of upper-case letters (A-Z) and digits (0-9). The remaining fields are c-strings of length 25 (26 including the null terminator). The structure is declared as follows:

```
const int PLATE_LENGTH = 7;
const int NAME_LENGTH = 25;
struct CarLicense
{
      char licensePlate[PLATE_LENGTH + 1];
      char owner[NAME_LENGTH + 1];
      char make[NAME_LENGTH + 1];
      char model[NAME_LENGTH + 1];
};
```

Program Modifications

As previously stated, a skeleton solution has been started. You **must** use the provided solution and **only complete these missing functions**:

1) `printLicense`: This function takes a single `CarLicense` and prints it nicely to the screen similar to the following:

```
License Plate: HANDY
        Owner: Manny
         Make: Toyota
        Model: Tacoma
```

2) `listAllLicenses`: This function prints all of the licenses to the screen. It accepts the entire license array and its fill level as parameters. It **must** make use of the `printLicense` function to print each license.

3) `doPlatesMatch`: This function will compare a license plate to a **search pattern** to see if they match. The order of the function's parameters should be license plate first and search pattern second. The search pattern represents the complete or partial license being looked for. To handle searches involving partial matches, a **wildcard** character is allowed in the search pattern. This wildcard is the question mark (`'?'`) which matches **any character**. So, for example, if we wanted to search for plates of length 6 which end in **38**, we would use the search pattern `????38`.

   In order for a plate to match a search pattern, all of their characters must match, position by position, from the start of both arrays. Two characters match if *both* are the same character, or if the character in the search pattern is a wildcard. A valid search pattern can be 6 or 7 characters long. If it is 6 characters long, it should only match 6-character license plates. A 7-character pattern can match a 6-character license plate, but only if the last character in the pattern is a wildcard, as it will match the null-terminator in the license plate.

   For example both the patterns `HA????` and `HA?????` would match the license plate `HANDY1`. But the pattern `?????Y1` would not match the license plate `HANDY1`, because while `?????` would match `HANDY` character by character, the `Y` in the pattern does not match the `1` in the license. Realize that the first pair of non-matching characters guarantee that a match is not possible, so any further comparison of remaining characters is not necessary.

4) `searchForLicense`: This function will search for and print all licenses which match the given search data. It accepts the entire license array, the fill level and a `CarLicense` structure filled with the search data. Each license that must be printed should be printed using the `printLicense` function. To determine if a license matches the search data, you should check each field in turn. For checking whether the license plate field matches, you must call the `doPlatesMatch` function. For all of the other fields, they must match the search data exactly unless the field in the search data is "ignore", in which case no match is necessary.

So, if the search data was:

```
License Plate: HAN?????
        Owner: ignore
         Make: Toyota
        Model: ignore
```

Then you would need to compare only the license plate fields (using `doPlatesMatch`) and the Make field (using `strncmp`). This comparison must be made to each of the licenses in the array and any licenses that match should be printed.

Important Reminders

- Be sure to follow the standards described in the handout **Coding Standards,** which outlines documentation requirements, coding "dos and don'ts" and other expectations regarding coding style.
- You should use named constants where appropriate (don't use hard–coded literal values).
- You may not use any C++ techniques or system functions which have not been covered in class.

Documentation

Because you are completing a program someone else started, you are **not** required to document **their code**. However, you are required to have a **complete identification block** and completely document every function **you** write.

Hand In

As usual, official test data and detailed submission instructions will be released 1-2 days before the due date.