Taxi Company Management System

1. Specification

1.1 Overall specification

This a taxi management system whose target users are taxi drivers and Taxi Company's administrators. This system is convenient for taxi drivers to check their up-dated information, such as rewards or punishment records and cars' information. This system also will improve the Taxi Company's management efficiency when it comes to integrating, viewing and modifying vehicle information and driver information.

1.2 Customer specification

The system should be able to store the information of the vehicles and drivers of a Taxi Company. The administrator of this system should have higher authority to perform all the editing, while the driver user can only search for their own information and rewards and punishment records.

The system should be able to provide functionality as listed below:

- User interface is friendly and clean.
- User can search for driver or vehicle information by inputting job number or car's license number respectively.
- Administrator logs in by entering correct password.
- After logging in, user has authority to browse, add, modify and delete the vehicle information and driver information.
- After logging in, user also has authority to register the payment information for vehicle and register rewards and punishments for every driver.
- Information stored in the system can be emailed to administrator.
- Each vehicle can be driven by up to two drivers.

1.3 System specification

Base of the system:

This system is constructed based on two hash-tables, one of which is used to store drivers' information and another is to store vehicles' information. The hash-tables are set based on the database file storing information.

By applying algorithm, hash-keys are obtained. Each hash-key points to a node in the hash-table. For example, for the hash-table storing drivers' information, when a node is pointed, the node includes information such as a specific driver's vehicle information, reward and punishment record, etc.

User interface:

To provide uncluttered interface and simple views and control, the system simulates LINUX SHELL using C++ codes. There are two threads consisted in the system. The main thread contains function commands, such as registration and editing information. User can call function by inputting corresponding number. The second thread is about system commands, such as exiting the program, returning to previous menu and getting help. User can call system command at any time when the main thread is operating. In particular, a practical system command is designed that is to send database file to a preset e-mail address.

Authority of driver:

Driver can search for his own information by inputting his unique job number. Every time when the administrator adds a new driver's information, a job number is generated. This job number is related to the calculation of hash-key. Thus by inputting the job number, the corresponding node of the driver storing hash-table can be found. Then the driver's all information will be shown on the screen.

Authority of administrator:

- **Add Vehicle or Driver information**

The adding process will take the form of Questions and Answers.
When add a driver, a generated job number will be assigned to whom. Then according to the job number, a new hash-key is attained, pointing to a new node in the hash-table. The information collected through Questions and Answers will store in the node.
When add a vehicle information, the process is similar. The only difference is that this time the hash-key is generated based on car's license number.

- **Modify**

Administrator firstly need to input job number or car's license number to find the specific node in the hash-table. This process is quite similar with the process for drivers searching for their own information. Then the object is located and then be changed. The modified object will then take the place of the old one.

- **Delete**

Delete user of vehicle information. For the integrity of this function, not only a node can be deleted from the hash-table, but also the data inside the node can be deleted. More specifically, deleting all the information of a driver is called deleting a node. However, if just delete the driver's vehicle information is not easily deleting the node.

- **Registration**

This system can register the payment information for vehicle and register rewards and punishments for every driver. Node in the hash-table can be found according to job name and license number and then add information to the node.

- **Edit password**

The administrator's password can be changed out of practical consideration.

- **Show records**

Any operation when it comes to changing the data in the database, such as adding driver information and registering punishment will be recorded and displayed on the screen as well as the operating time.

2. Analysis and Design

2.1 System structure

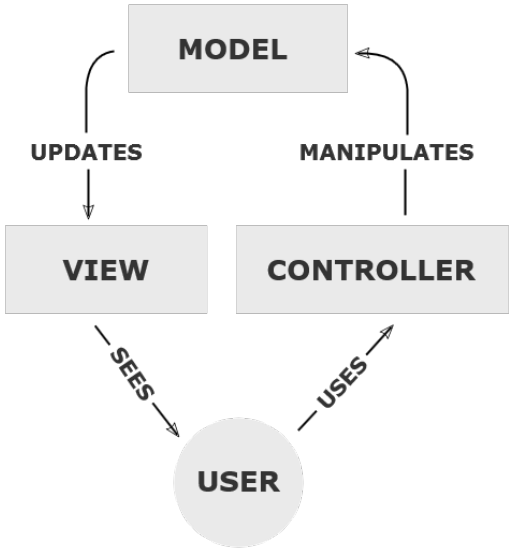
The system structure is contained in the html file. For each part, click “+” for viewing more information. The inputs and outputs for each part are also declared in the file.

2.2 Design

Development Mode

The development mode is generally similar with MVC.

Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces on computers. It divides a given application into three interconnected parts in order to separate internal representations of information from the ways that information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.



For developing this system, the core of MVC is drawn and used not only for implementing user interfaces but influencing design.

MVC's M: Hash-table

M means model, which can be interpreted as the base of the system. This system is used to store information and make changes to the information. Thus how to store and modify data is the most important process for programming this system. Hash-table is applied. There is one thing need being clarified, the difference between hash-table and the file to store information. Every time when the program starts running, hash-table is created based on the file, for much more convenience of realizing the functions such as viewing, searching or modifying. Below are the concrete steps of using hash-table.

Part1: Overview of creating table.

```
while (fread(&Current, sizeof(struct Admin), 1, ptr_account)) {

    key = Hash_Key_Function(Current.name);

        Hash_Expand(Hasharray[key], h);       //put each account into its 'bucket'

        fflush(stdin);

}
```

After the calculation of the 'Hash key', they are sent to the function: 'Hash_Expand()' and stored into a bucket depends on the value of the 'Hash key'.

/*function: int Hash_Key_Function(char name[10]);*/

```
int Hash_Key_Function(char name[10]) {

        int key = strlen(name);

        return key;

}
```

This function is created for calculating the hash key which represents for the position it should be placed into the bucket of a certain account. As an important part of the Hash-table application, there still exists room for improvement.

/*function: void Hash_Expand(struct Admin_Node *head, int *h);*/

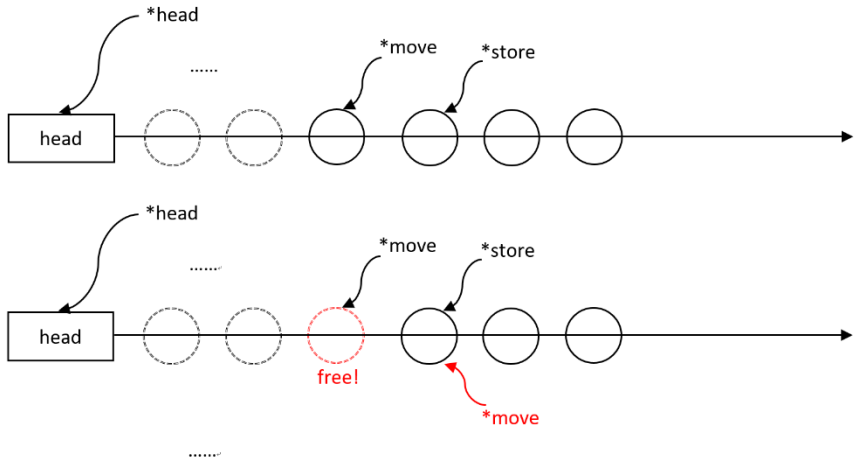
```
void Hash_Expand(struct Admin_Node *head, int *h) {
        while ((head->next) != NULL)
                head = head->next;
    head->next = (struct Admin_Node*)malloc(sizeof(struct Admin_Node));
        head = head->next;
        head->next = NULL;
        strcpy(head->name, Current.name);
        strcpy(head->password, Current.password);
        head->position = Location_In_File;
        head->money = Current.money;
        if ((*head).name[0] != '0') {
                printf("HashTable is expanded.\n");
                (*h)++;
    }
        Location_In_File++;    //real location of each account in the file
}
```

As we see, the 'Hasharray[key]' passed in becomes '*head' in this function. Specifically, the nature of each bucket is an independent link-list, and its 'head' is the part of the 'Hasharray'. As the key has been already figured out, the account information record by 'Current' will be added in a precise location of the table. Each time a new driver or a new vehicle (a new node) is added, a new memory space is allocated for it, and the pointer to 'NULL' (the last position of the link-list) will be put back a position.

/*function: static void Hashtable_Free(struct Admin_Node **Hasharray);*/

```
static void Hashtable_Free(struct Admin_Node **Hasharray) {
        struct Admin_Node *head, *move, *store;
        for (int i = 0; i < 10; i++) {
                head = Hasharray[i];
                move = head;
                while ((move->next) != NULL) {
                        store = move->next;
                        free(move);
                        move = store;
                }
        }
        free(Hasharray);
}
```

This function is designed to release the whole structure of the Hashtable before user exiting from the program. Three ‘struct Admin_Node’ structure pointers ‘*head’, ‘*move’ and ‘*store’ work together to free the memory space of each node from the head of the linklist to its tail (Figure1).



MVC's V and C: LINUX SHELL

View and control functions have close relationships. Control can be interpreted as buttons. Every time when a button is pressed, a function is called to run. Thus it makes the programming algorithms much more explicit. All the functions are object orientation. Besides the functions required according to customer specifications, some system commands or controls are added to the system, such as exiting the program, returning to previous menu and getting help. This operation takes the inspiration of LINUX SHELL.

A Linux Shell is a command-line interpreter or shell that provides a traditional Linux-like command line user interface. Users direct the operation of the computer by entering commands as text for a command line interpreter to execute, or by creating text scripts of one or more such commands.

Inspired by the tidy interface, there are two threads consisted in the system. The main thread contains function commands, such as registration and editing information. User can call function by inputting corresponding number. The second thread is about system commands, such as exiting the program, returning to previous menu and getting help. User can call system command at any time when the main thread is operating. In particular, a practical system command is designed that is to send database file to a preset e-mail address.

Below are the code of system command.

```
else if (strncmp(input_check, "rm/",3) == 0)
              {
                     len_tmp = strlen(input_check);
                     for (count_tmp = 0; count_tmp < (len_tmp-3); count_tmp++)
                     {
                            input_tmp[count_tmp] = input_check[count_tmp + 3];
                     }
                     input_tmp[count_tmp] = '\0';

                     if (strcmp(input_tmp, "driver.dat") == 0)
                     {
```

```
                        sc->DeleteAllDriver();
                        check_file_driver = 1;
                }
                else if (strcmp(input_tmp, "vehicle.dat") == 0)
                {
                        sc->DeleteAllVehicle();
                        check_file_vehicle = 1;
                }
                else if (strcmp(input_tmp, "editing_history.txt") == 0)
                {
                        sc->deleteHistory();
                        check_file_history = 1;
                }
                else if (strcmp(input_tmp, "password.txt") == 0)
                {
                        cout << "Permission denied.";
                }
                else {
                        cout << "rm: "<<input_tmp<<": No such file or directory\n";
                }

                system_Show();
                memset(input_tmp, '\0', 30);
                memset(input_check, '\0', 30);
                continue;
        }
        else if (strcmp(input_check, "exit") == 0)
        {

                char* szStr = "final_AS6.exe";
                WCHAR wszClassName[256];
                memset(wszClassName, 0, sizeof(wszClassName));
                MultiByteToWideChar(CP_ACP, 0, szStr, strlen(szStr) + 1,
wszClassName,
                        sizeof(wszClassName) / sizeof(wszClassName[0]));

                STARTUPINFO si = { sizeof(si) };
                PROCESS_INFORMATION pi;
                si.dwFlags = STARTF_USESHOWWINDOW;
```

```
        si.wShowWindow = TRUE;
        BOOL bRet = CreateProcess(
                NULL,
                wszClassName,
                NULL,                                   NULL,
                FALSE,
                CREATE_NEW_CONSOLE | REALTIME_PRIORITY_CLASS,
                NULL,
                NULL,
                &si,
                &pi);
        memset(input_check, '\0', 30);
        exit(0);
}
else if (strcmp(input_check, "quit") == 0)
{
        int lp_tmp;
        int count;
        system("cls");
        for (int lp_tmp = 0; lp_tmp < 2; lp_tmp++)
        {
                cout << "\n\n\n\n\n\t\tGoodBye  ";
                for (int count = 0; count < 3; count++)
                {
                cout << "^w^";
                        Sleep(400);
                }
                system("cls");
        }

        exit(0);
```

Through the LINUX-SHELL like interface, user can view all the functions clearly and easily control the system.

2.3 API

3. Testing

4. Bugs report

5. User manual

5.1 Purpose

This a taxi management system whose target users are taxi drivers and Taxi Company's administrators. This system is convenient for taxi drivers to check their up-dated information, such as rewards or punishment records and cars' information. This system also will improve the Taxi Company's management efficiency when it comes to integrating, viewing and modifying vehicle information and driver information.

5.2 Functions

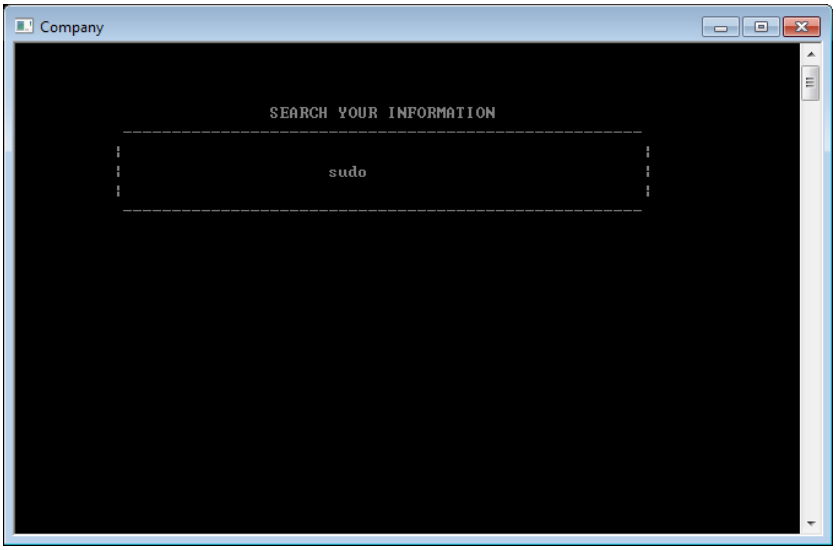
5.2.1 Common users

- Search vehicle or driver's information

5.2.2 Administrator

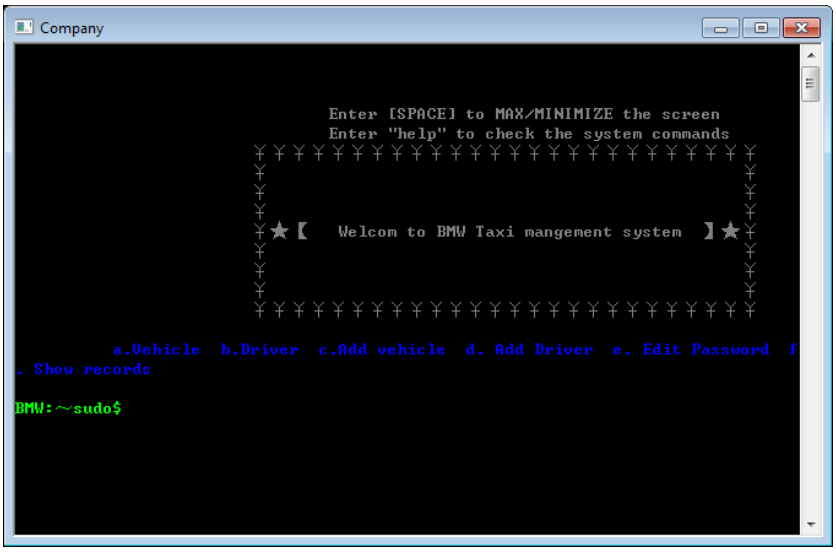
- Log in

Input 'sudo' to enter the mode of administrator.



Enter correct password to log in. The initial password is 1111.

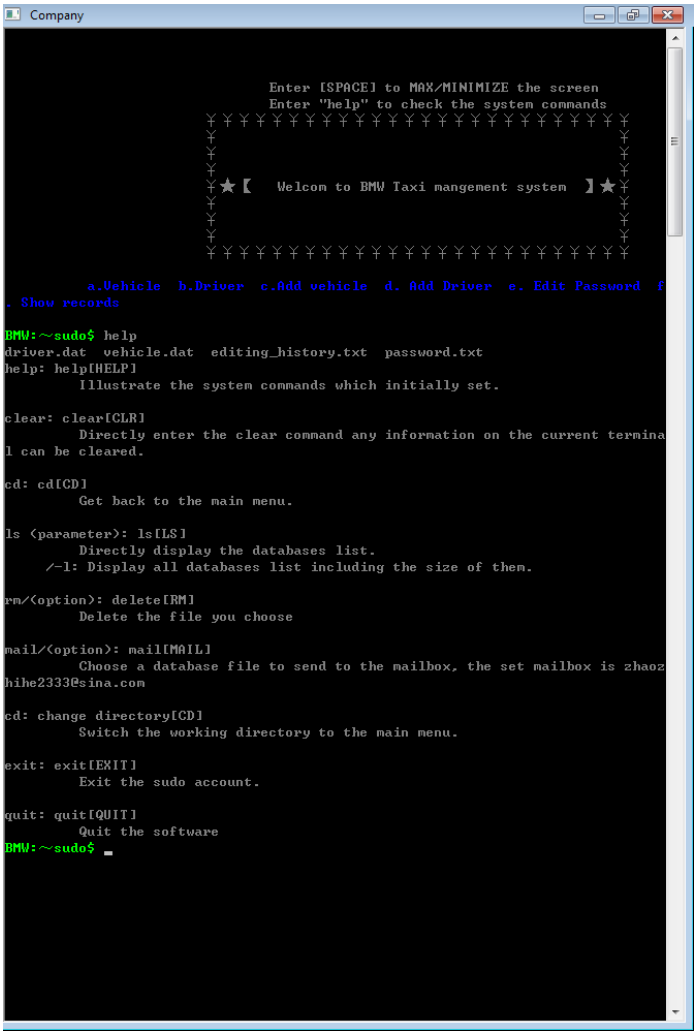
Log in successfully.



a) System Commands

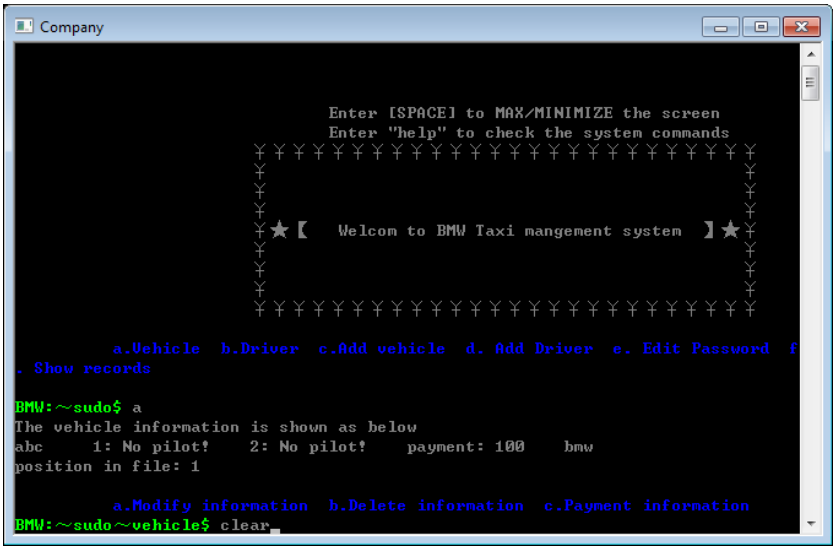
- Help

Input ‘help’ to get all the descriptions of system commands.

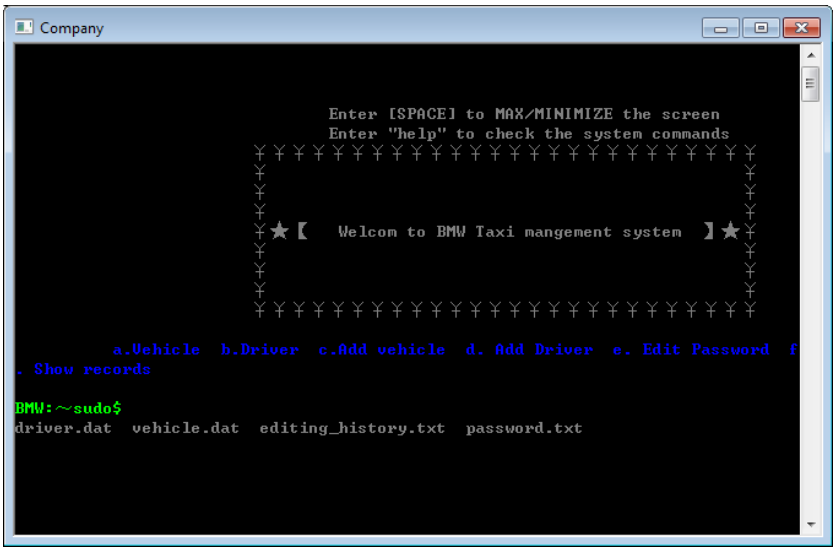


- Clear

Input ‘Clear’ to clear the screen.



Screen is cleaned.

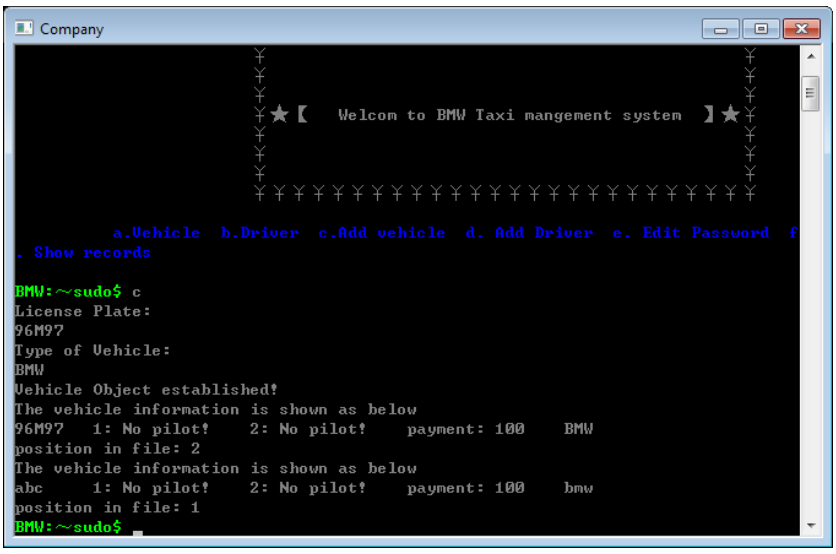


- Ls
- Rm
- Mail
- Cd
- Exit

b) Management Commands

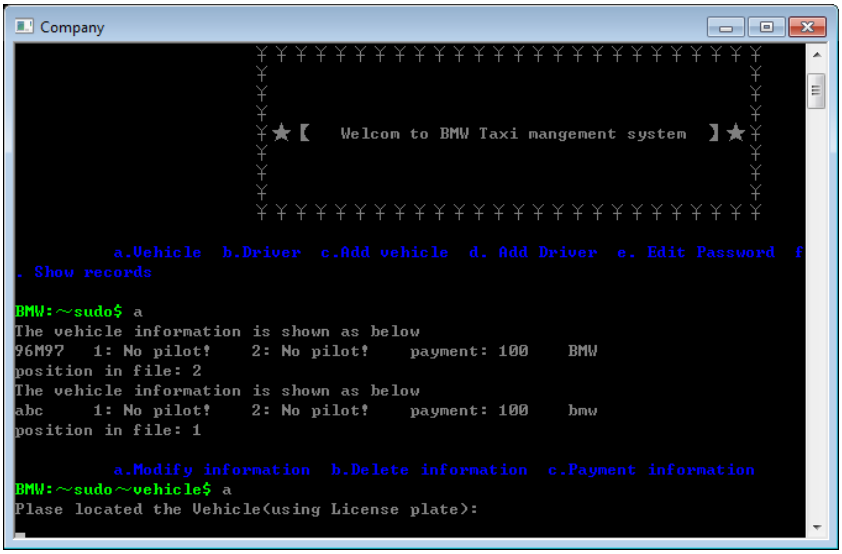
- Browse, add, modify and delete the vehicle information
- Add vehicle information

Input ‘c’ to choose the function and type in the information followed by each question.

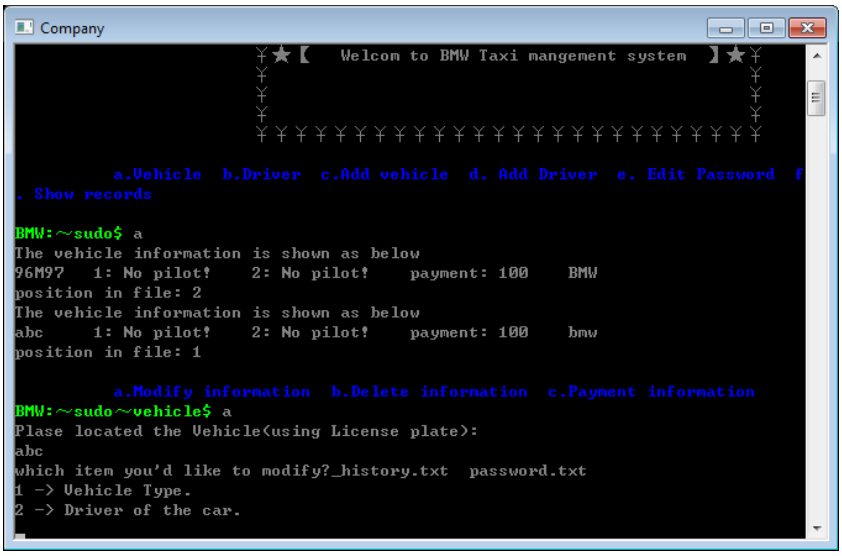


- Modify vehicle information

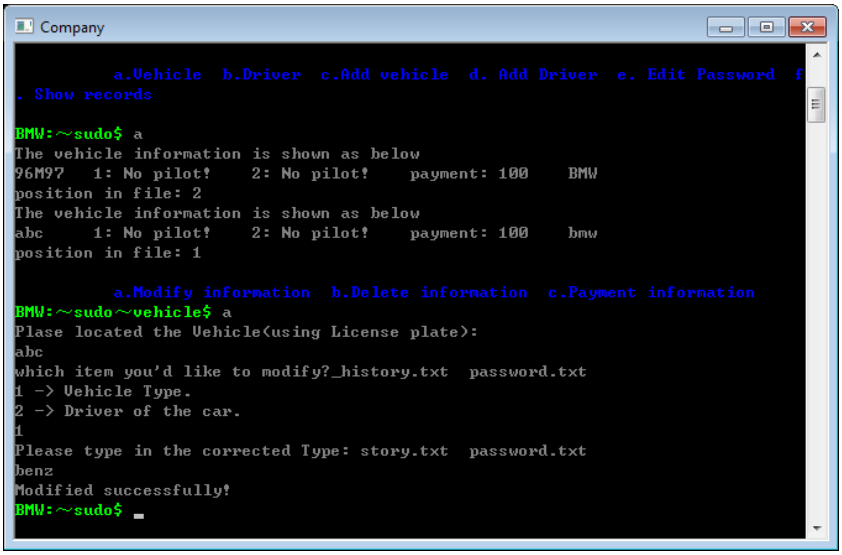
Input ‘a’ to enter next level then input ‘a’ to choose the function.

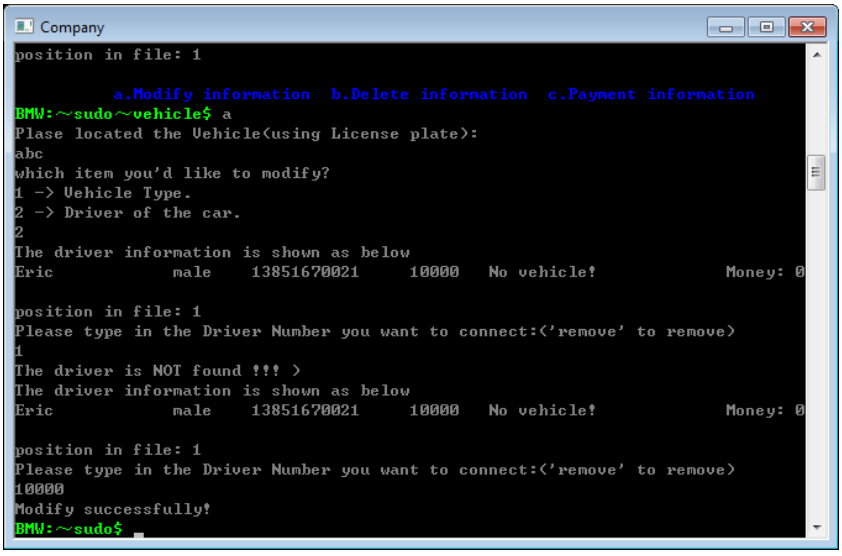


Input license plate to locate vehicle.



Input ‘1’ to modify vehicle type and input ‘2’ to change the driver of the car.





- Browse, add, modify and delete the driver information
- Register the payment information for vehicle
- Register rewards and punishments for driver