Object Oriented Programming Report

Parking System

Group Members:

Lê Quốc Hoàng - Coordinator
Cao Ngọc Nhật Huy – Main Coder
Lê Tuấn Minh – Report Writer

12/27/2022

Table of Contents

I. Introduction

This project aims to create a program for administration to keep track of the vehicles, including cars and motorbikes, going in and out of a parking lot. The program is expected to perform commands as below with no error:

- Adding a vehicle.
- Removing a vehicle.
- Displaying all vehicles.
- Browsing for a vehicle.
- Calculating the parking fee of a vehicle.

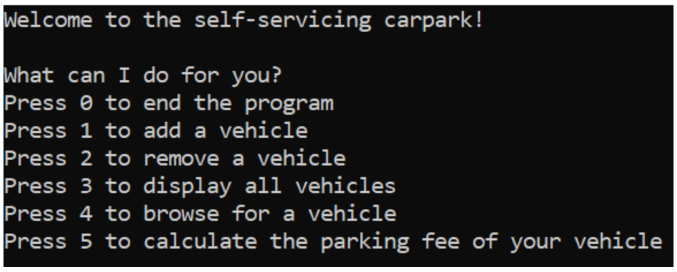
Additionally, this program can be modified to become the one that vehicles' owners can use to self-check-in and self-check-out at an entry of a parking lot. Actually, the program still have to connect with a data system using NFC cards that have been applied widely in the real world to collect actual parking time then calculate the parking fee. To solve this problem, we build a function that can random the actual parking time.

In order to build the program, we ultilize some concepts, including classes, strings, inheritance, polymorphism, case-switch and basic algorithms. To be more specific, there are one base class called "Vehicle", which has two derrived class named "Car" and "Motorbike"; and one separate class "Manager" used to operate all functions. Details of the program is clarified in next sections.

II. Demonstration

1. Home menu

As seen in the home menu display (Figure 1), there are six commands the user can choose, including Adding, Removing, Displaying, Browsing, Calculating and Exit. For each button the user chooses, the system will clear the screen and change to the screen of the command.

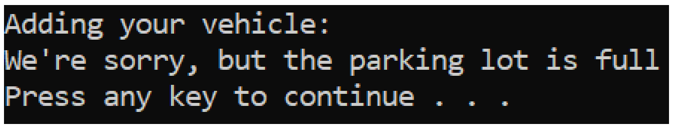


```
Welcome to the self-servicing carpark!

What can I do for you?
Press 0 to end the program
Press 1 to add a vehicle
Press 2 to remove a vehicle
Press 3 to display all vehicles
Press 4 to browse for a vehicle
Press 5 to calculate the parking fee of your vehicle
```

Figure 1: Home menu display.

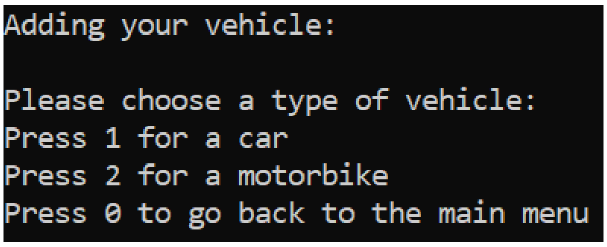
2. Adding a vehicle

When the Adding command is called, it first checks whether the slots are totally full or not. If there is no room for the vehicle, the system will send an anouncement about that (Figure 2) and take the user back to the home menu, otherwise it will prompt 3 options "Cars", "Motorbikes" and "Exit" (Figure 3).



```
Adding your vehicle:
We're sorry, but the parking lot is full
Press any key to continue . . .
```

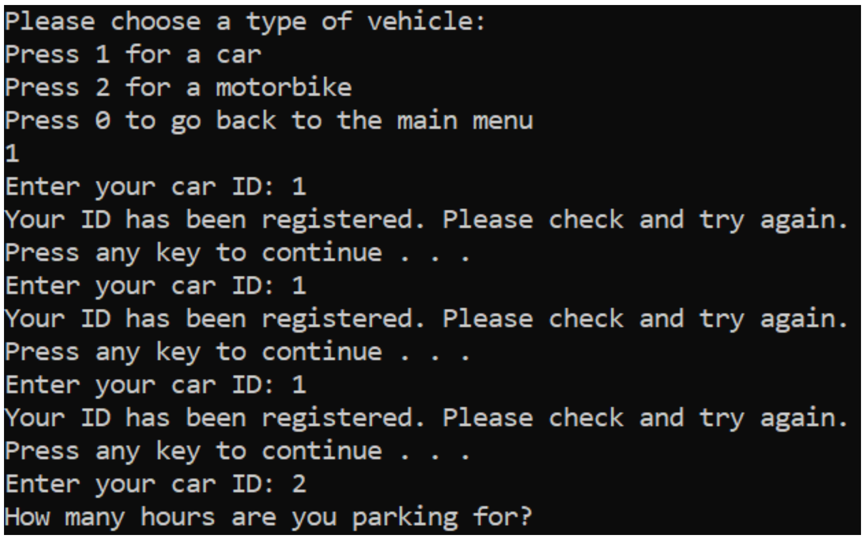
Figure 2: Announcement of full capability.



```
Adding your vehicle:

Please choose a type of vehicle:
Press 1 for a car
Press 2 for a motorbike
Press 0 to go back to the main menu
```

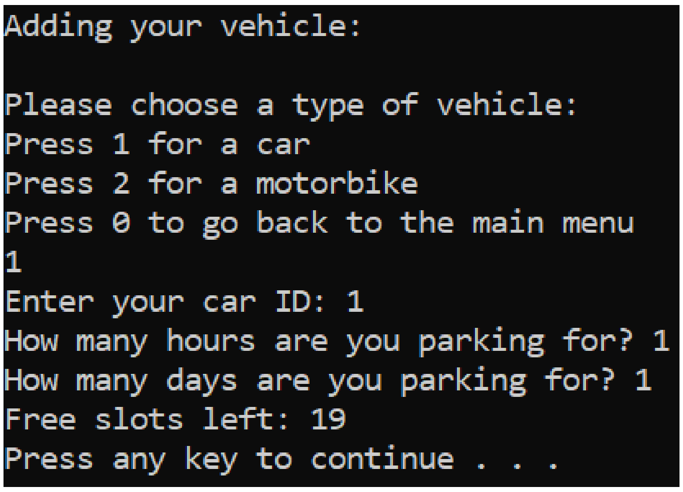
Figure 3: Adding menu display.

If the user choose "Exit" option, the system will return to the home menu. If "Cars" or "Motorbike" option is chosen, the user will be asked to provide their vehicle's plate ID. There are also functions to check if the ID has been registered or not and force the user to provide the ID again until it is accepted if there is a vehicle with the same ID in the parking lot already (Figure 4). Once the ID registration has been completed, the user then provide more information about the amount of time their vehicle stays at the slot (Figure 5) and return to the Adding a vehicle menu.



```
Please choose a type of vehicle:
Press 1 for a car
Press 2 for a motorbike
Press 0 to go back to the main menu
1
Enter your car ID: 1
Your ID has been registered. Please check and try again.
Press any key to continue . . .
Enter your car ID: 1
Your ID has been registered. Please check and try again.
Press any key to continue . . .
Enter your car ID: 1
Your ID has been registered. Please check and try again.
Press any key to continue . . .
Enter your car ID: 2
How many hours are you parking for?
```

Figure 4: Request for re-entering the plate ID.

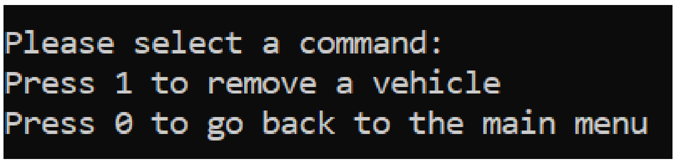


```
Adding your vehicle:

Please choose a type of vehicle:
Press 1 for a car
Press 2 for a motorbike
Press 0 to go back to the main menu
1
Enter your car ID: 1
How many hours are you parking for? 1
How many days are you parking for? 1
Free slots left: 19
Press any key to continue . . .
```

Figure 5: Successfully adding a vehicle.

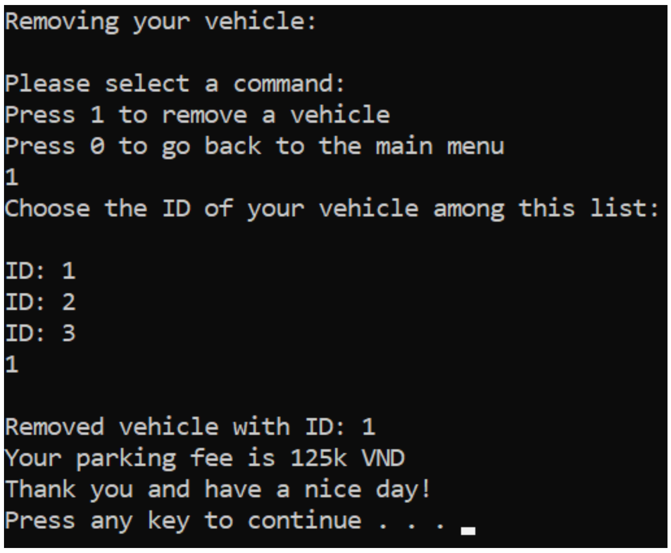
3. Removing a vehicle

When the Removing command is called, the system prompts two options Removing and Exit (Figure 6). If the user hit the Exit option, they will go back to the home menu. When Removing option is chosen, the system will display all vehicles' ID for the user to choose one and enter to remove the vehicle with that ID as well as receive the actual parking fee of the vehicle (Figure 7). Then the Removing screen appears again and the user can choose to keep removing or go back to the home menu.



```
Please select a command:
Press 1 to remove a vehicle
Press 0 to go back to the main menu
```

Figure 6: Removing menu display.

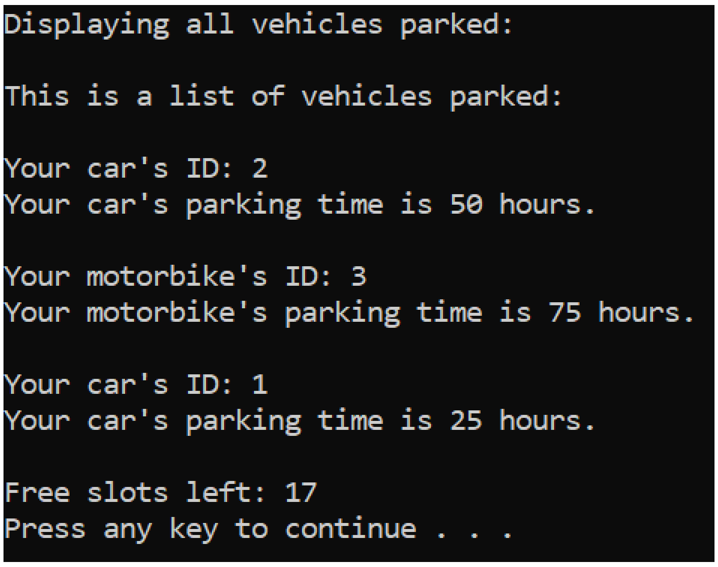


```
Removing your vehicle:

Please select a command:
Press 1 to remove a vehicle
Press 0 to go back to the main menu
1
Choose the ID of your vehicle among this list:

ID: 1
ID: 2
ID: 3
1

Removed vehicle with ID: 1
Your parking fee is 125k VND
Thank you and have a nice day!
Press any key to continue . . .
```

Figure 7: Successfully removing a vehicle.

4. Displaying all vehicles

When the Displaying command is called, a list of all vehicles' information is displayed for the user to observe their ID and registered parking time (Figure 8).

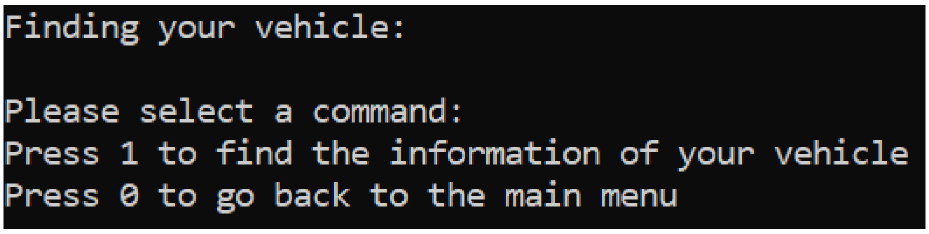


```
Displaying all vehicles parked:

This is a list of vehicles parked:

Your car's ID: 2
Your car's parking time is 50 hours.

Your motorbike's ID: 3
Your motorbike's parking time is 75 hours.

Your car's ID: 1
Your car's parking time is 25 hours.

Free slots left: 17
Press any key to continue . . .
```

Figure 8: The list of all vehicles' information is being displayed.

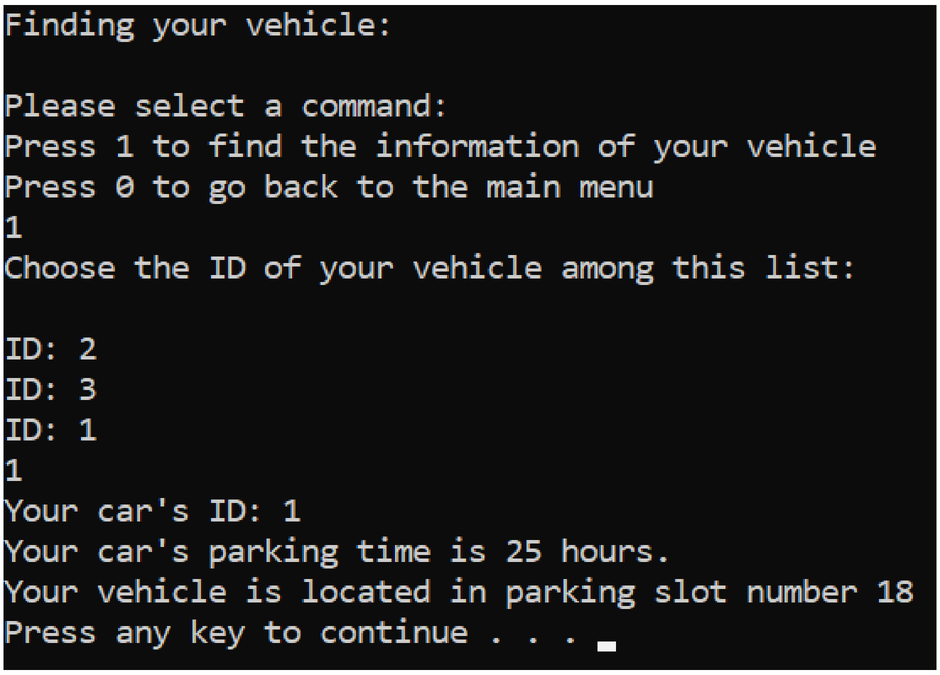
5. Browsing a vehicle

When the Browsing command is called, the system prompts two options Browsing and Exit (Figure 9). If the user hit the Exit option, they will go back to the home menu. When Browsing option is chosen, the system will display all vehicles' ID for the user to browse for a specific vehicle and receive its registered parking time and position. Then the Browsing screen appears again and the user can choose to keep browsing or go back to the home menu.



```
Finding your vehicle:

Please select a command:
Press 1 to find the information of your vehicle
Press 0 to go back to the main menu
```

Figure 9: Browsing menu display.

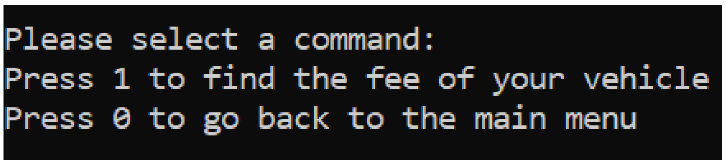


```
Finding your vehicle:

Please select a command:
Press 1 to find the information of your vehicle
Press 0 to go back to the main menu
1
Choose the ID of your vehicle among this list:

ID: 2
ID: 3
ID: 1
1
Your car's ID: 1
Your car's parking time is 25 hours.
Your vehicle is located in parking slot number 18
Press any key to continue . . .
```

Figure 10: Successfully browsing for a vehicle.

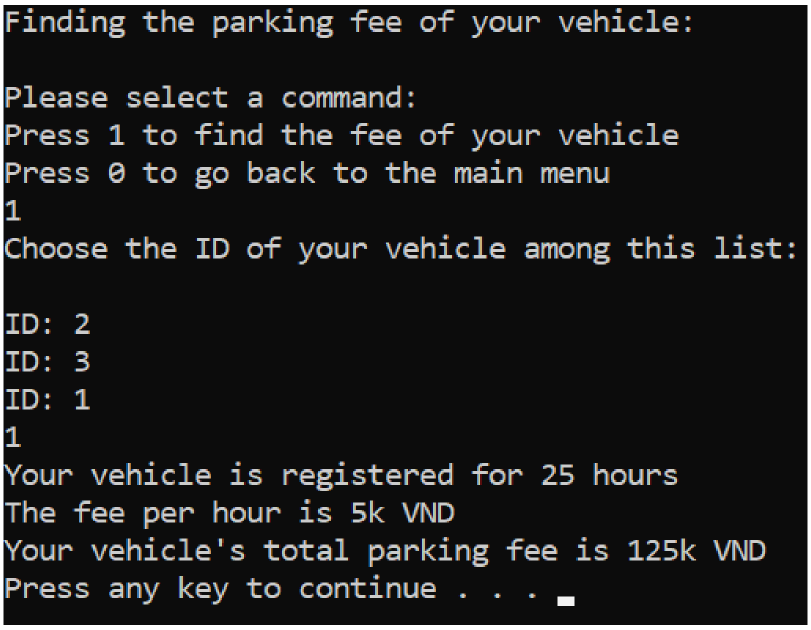
6. Calculating the parking fee of a vehicle

When the Calculating command is called, the system prompts two options Calculating and Exit (Figure 11). If the user hit the Exit option, they will go back to the home menu. When Calculating option is chosen, the system will display all vehicles' ID for the user to choose a specific vehicle and receive its parking fee arcording to its registered parking time (Figure 12). Then the Calculating screen appears again and the user can choose to keep calculating or go back to the home menu.



```
Please select a command:
Press 1 to find the fee of your vehicle
Press 0 to go back to the main menu
```

Figure 11: Calculating menu display.



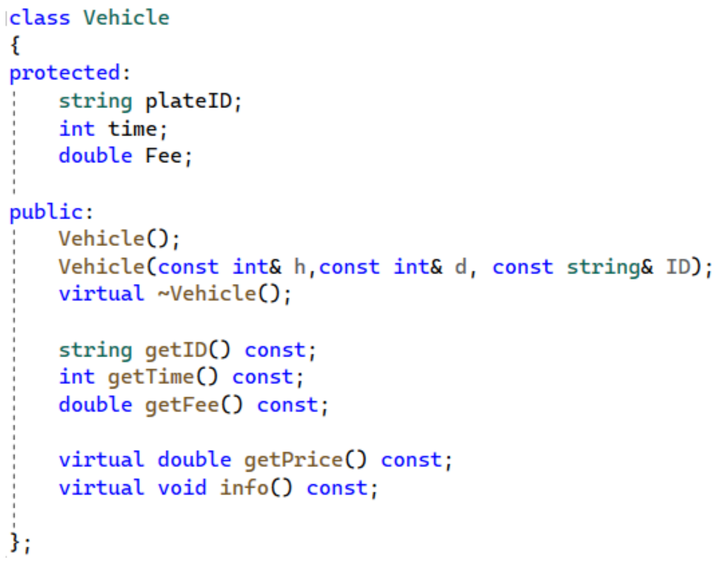
```
Finding the parking fee of your vehicle:

Please select a command:
Press 1 to find the fee of your vehicle
Press 0 to go back to the main menu
1
Choose the ID of your vehicle among this list:

ID: 2
ID: 3
ID: 1
1
Your vehicle is registered for 25 hours
The fee per hour is 5k VND
Your vehicle's total parking fee is 125k VND
Press any key to continue . . .
```

Figure 12: Successfully calculating a vehicle's parking fee.

III. Detailed Explanations

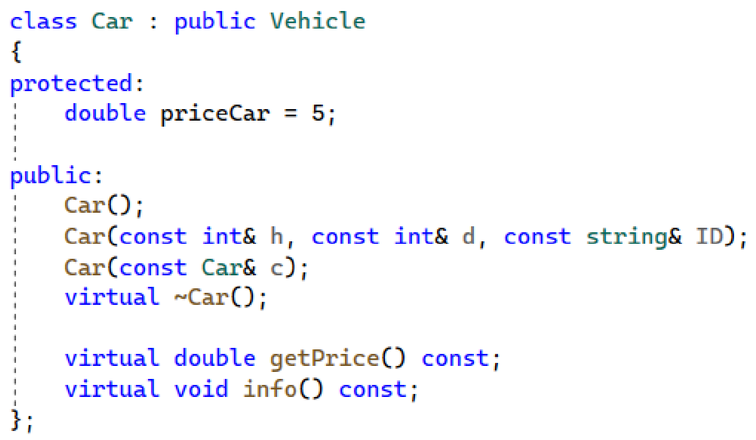
1. Header Introduction

1.1. Vehicle.h



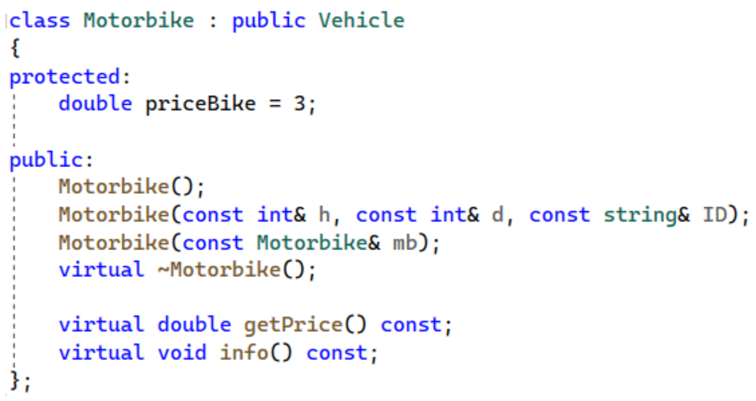
```cpp
class Vehicle
{
protected:
    string plateID;
    int time;
    double Fee;

public:
    Vehicle();
    Vehicle(const int& h,const int& d, const string& ID);
    virtual ~Vehicle();

    string getID() const;
    int getTime() const;
    double getFee() const;

    virtual double getPrice() const;
    virtual void info() const;

};
```

1.2. Car.h



```cpp
class Car : public Vehicle
{
protected:
    double priceCar = 5;

public:
    Car();
    Car(const int& h, const int& d, const string& ID);
    Car(const Car& c);
    virtual ~Car();

    virtual double getPrice() const;
    virtual void info() const;
};
```

1.3. Motorbike.h



```cpp
class Motorbike : public Vehicle
{
protected:
    double priceBike = 3;

public:
    Motorbike();
    Motorbike(const int& h, const int& d, const string& ID);
    Motorbike(const Motorbike& mb);
    virtual ~Motorbike();

    virtual double getPrice() const;
    virtual void info() const;
};
```

1.4. Manager.h

```
class Manager
{
private:
    Vehicle* Parkingslot[20];
    int Counter = 20;
    Car* tempCar = new Car();
    Motorbike* tempMotorbike = new Motorbike();
    Vehicle* emptyVehicle = new Vehicle();

public:
    Manager();
    ~Manager();

    //a menu and its options/methods
    void menu();
    void priceCalculation();
    void display();
    void addVehicle();
    void removeVehicle();
    void browseVehicle();

    //late fee simulation
    void randLateFee(const Vehicle& vec);
};
```

2. Algorithms:

2.1. Home menu

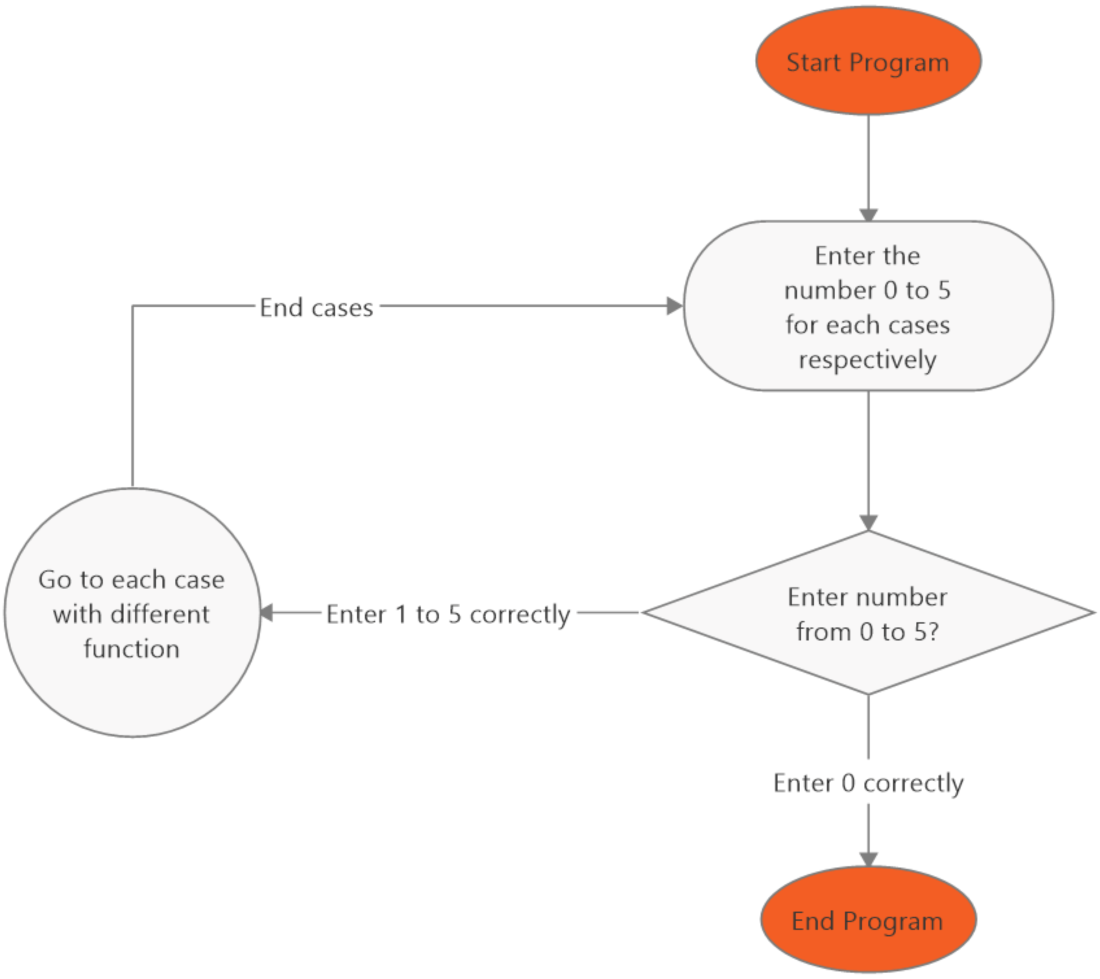
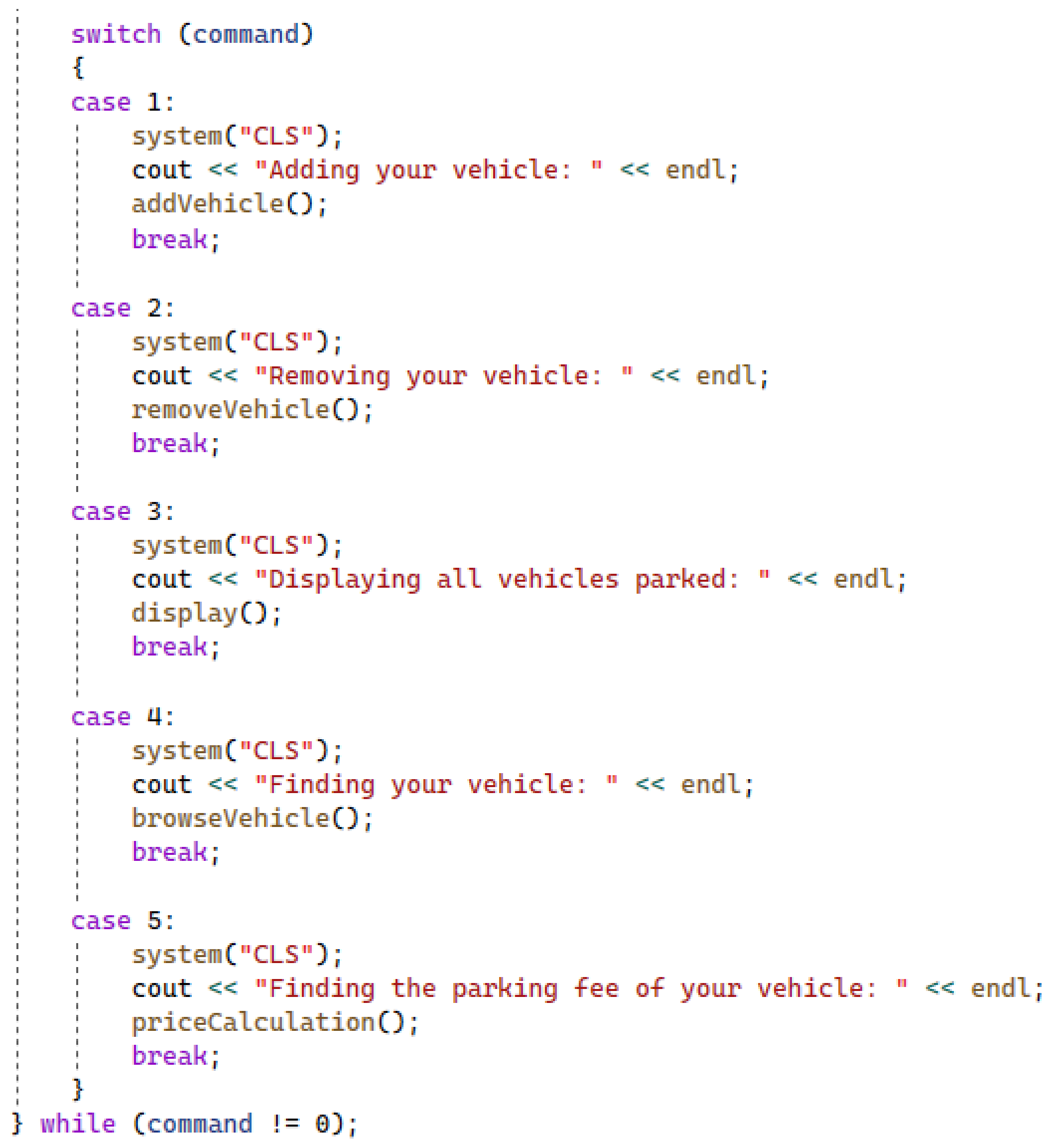

Start Program
Enter the number 0 to 5 for each cases respectively
End cases
Go to each case with different function
Enter 1 to 5 correctly
Enter number from 0 to 5?
Enter 0 correctly
End Program

Figure 13: Pseudocode for displaying home menu.

To display the home menu, we write a method *"menu()"* in the class *"Manager"* then create a variable and call this method in our main program *"Project Parking Lot.cpp"*. As seen in the home menu display (Figure 1), there are six commands the user can choose. Switch-case (Figure 14) is used here to call methods *"addVehicle()", "removeVehicle()", "display()", "browseVehicle()", "priceCalculation()"* when the corresponding button, including 1 to 5 respectively, is pressed. Before these methods are called, *"system("CLS")"* is used to clear the screen for following displays. When the user press 0, the program ends. This is actually a *"do…while"* loop which will repeat whenever the variable *"command"* is not 0.



```
    switch (command)
    {
    case 1:
        system("CLS");
        cout << "Adding your vehicle: " << endl;
        addVehicle();
        break;

    case 2:
        system("CLS");
        cout << "Removing your vehicle: " << endl;
        removeVehicle();
        break;

    case 3:
        system("CLS");
        cout << "Displaying all vehicles parked: " << endl;
        display();
        break;

    case 4:
        system("CLS");
        cout << "Finding your vehicle: " << endl;
        browseVehicle();
        break;

    case 5:
        system("CLS");
        cout << "Finding the parking fee of your vehicle: " << endl;
        priceCalculation();
        break;
    }
} while (command != 0);
```

Figure 14: Switch-case algorithm to operate commands.

2.2. Adding a vehicle

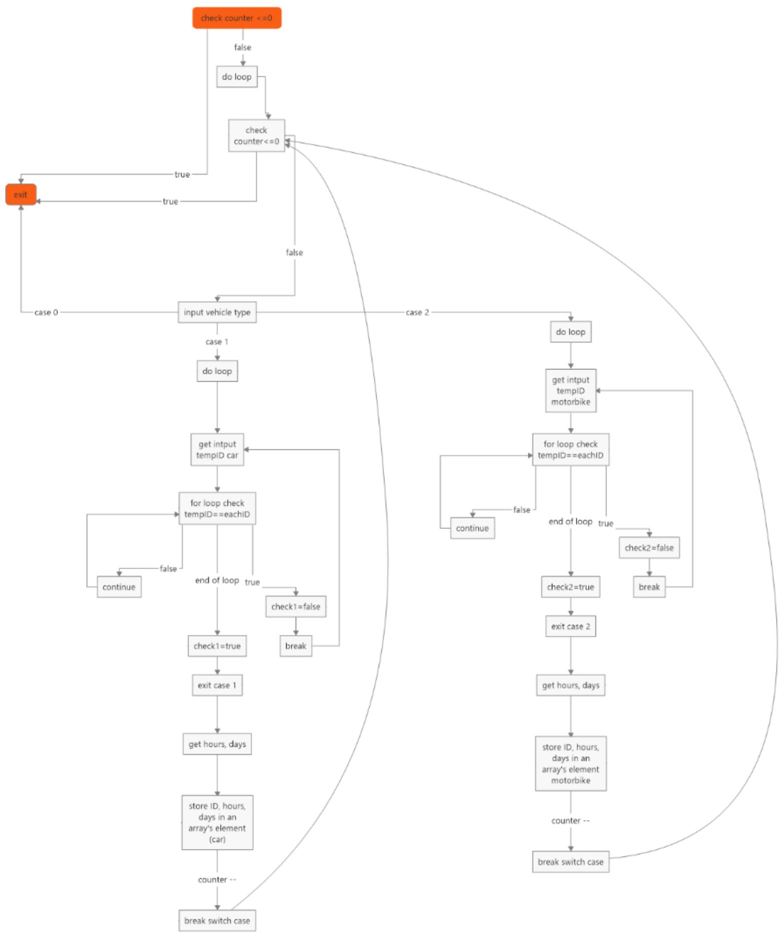
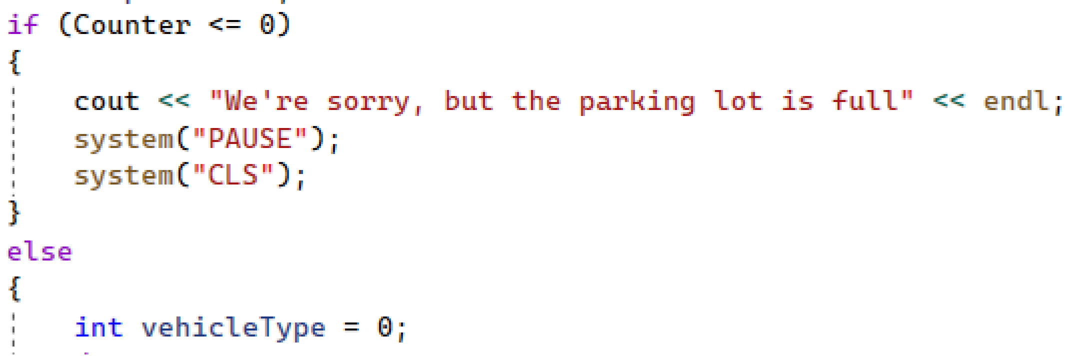


Figure 14: Pseudocode for adding a vehicle.

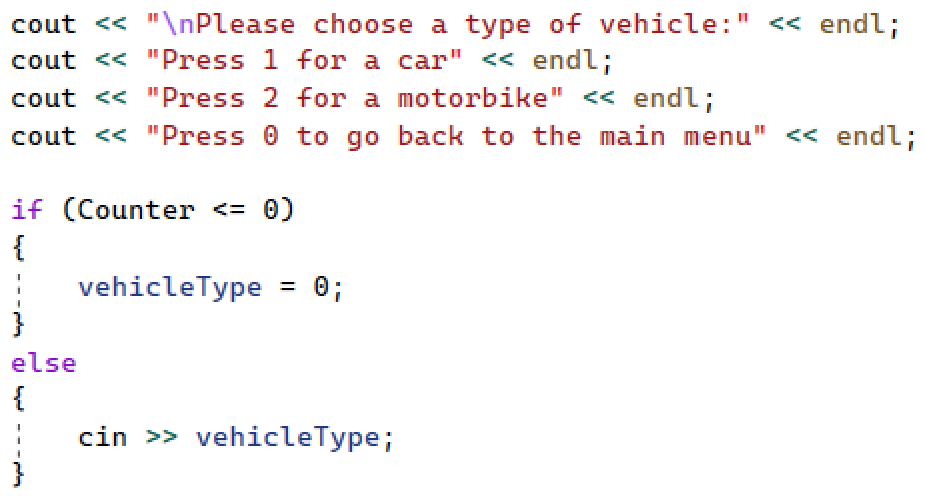
When the user press 1, method *"addVehicle()"*, which is defined in class *"Manager"*, is called. This method first check whether the slots, which is represented using *"Counter"* variable (starts from 20), are totally full or not with the condition *"Counter <= 0"* (Figure 15). If there is no room for the vehicle, the system will send an anouncement about that, otherwise it will initialize a local variable *"int vehicleType"* and set it at 0 then get in to a *"do...while"* loop (1) which will repeat whenever the variable *"vehicleType"* is not 0. The system after that prompt 3 options *"Cars"*, *"Motorbikes"* and *"Exit"* (Figure 3).



```
if (Counter <= 0)
{
    cout << "We're sorry, but the parking lot is full" << endl;
    system("PAUSE");
    system("CLS");
}
else
{
    int vehicleType = 0;
```

Figure 14: If-else condition for checking the availability of parking lot.

At the beginning of the aforementioned *"do…while"* loop, after some *"cout"* functions, there is an *"if-else"* algorithm to check whether the value of the variable *"Counter"* is larger than 0 or not (Figure 16). The reason why it is placed there is because after each time the user successfully add a vehicle, variable *"Counter"* will decrease by 1, so the *"if-else"* algorithm can get the user back to the home menu, which occurs so quickly that the user will not be able to see the result of *"cout"* functions.



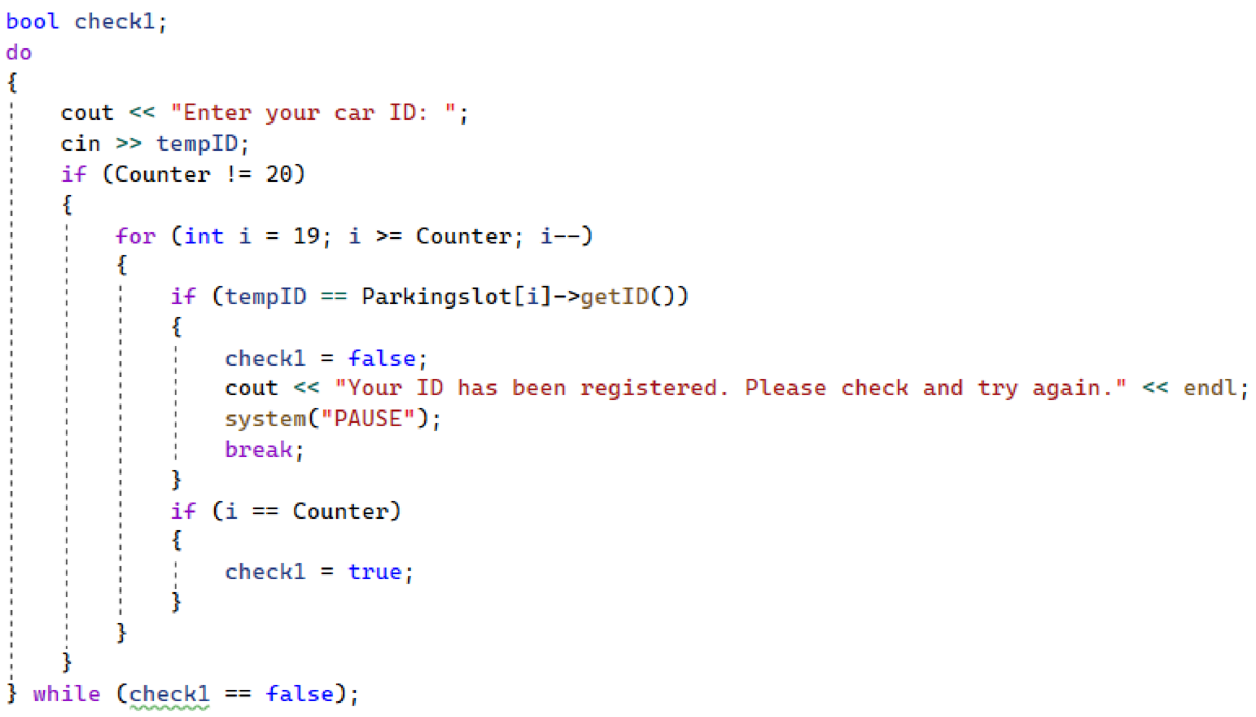
```
cout << "\nPlease choose a type of vehicle:" << endl;
cout << "Press 1 for a car" << endl;
cout << "Press 2 for a motorbike" << endl;
cout << "Press 0 to go back to the main menu" << endl;

if (Counter <= 0)
{
    vehicleType = 0;
}
else
{
    cin >> vehicleType;
}
```

Figure 15: If-else condition for re-checking the availability of parking lot.

If the parking lot is still available, which means the variable *"Counter"* is larger than 0, the number that is received from user's input will become the value of variable *"vehicleType"* through *"cin"* function. Note that there are only 3 buttons 0, 1 and 2 that can activate further functions correctly, other circumstances may lead to many errors.

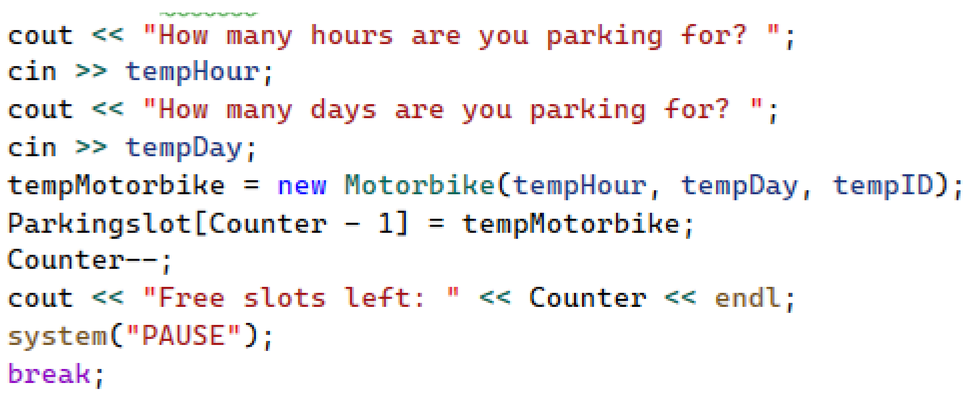
If the user press 0, the system will return to the home menu because the value of the variable *"vehicleType"* is still 0.

If button 1 or 2 is pressed, a local variable *"check1"* or *"check2"* is initialized correspondingly to check whether there is any vehicle with the same ID in the parking lot or not. The program will then get into a *"do…while"* loop (2) which will repeat whenever the variable *"check1"* or *"check2"* is false. The program check the ID by using a *"for"* loop (3) to compare the input ID and others, which starts at 19 and ends at *"Counter"* (Figure 16). During the loop (3), if there is any other vehicle that has the same ID, *"check1"* or *"check2"* will be set to false, the loop (3) stops immediately due to *"break"* function and the program will prompt a message asking the user to enter the ID again then starts a new loop (2). Otherwise, if the loop (3) can complete with no familiar ID, *"check1"* or *"check2"* will be set to true and the user can continue to enter inputs for parking time. These information will be stored in temporay variables then transferred to the array *"Parkingslot[]"* (Figure 17).



```
bool check1;
do
{
    cout << "Enter your car ID: ";
    cin >> tempID;
    if (Counter != 20)
    {
        for (int i = 19; i >= Counter; i--)
        {
            if (tempID == Parkingslot[i]->getID())
            {
                check1 = false;
                cout << "Your ID has been registered. Please check and try again." << endl;
                system("PAUSE");
                break;
            }
            if (i == Counter)
            {
                check1 = true;
            }
        }
    }
} while (check1 == false);
```

Figure 16: Do…while loop for checking ID.



```
cout << "How many hours are you parking for? ";
cin >> tempHour;
cout << "How many days are you parking for? ";
cin >> tempDay;
tempMotorbike = new Motorbike(tempHour, tempDay, tempID);
Parkingslot[Counter - 1] = tempMotorbike;
Counter--;
cout << "Free slots left: " << Counter << endl;
system("PAUSE");
break;
```

Figure 17:Storing parking time inputs.

2.3. Removing a vehicle

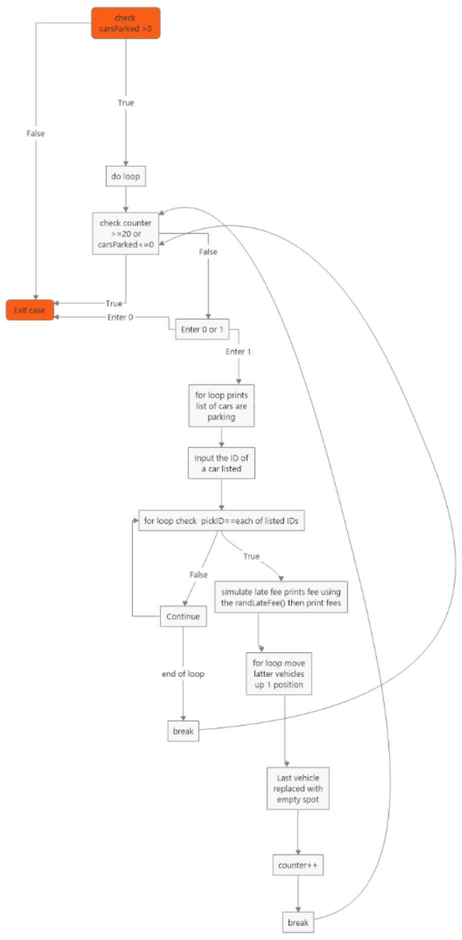
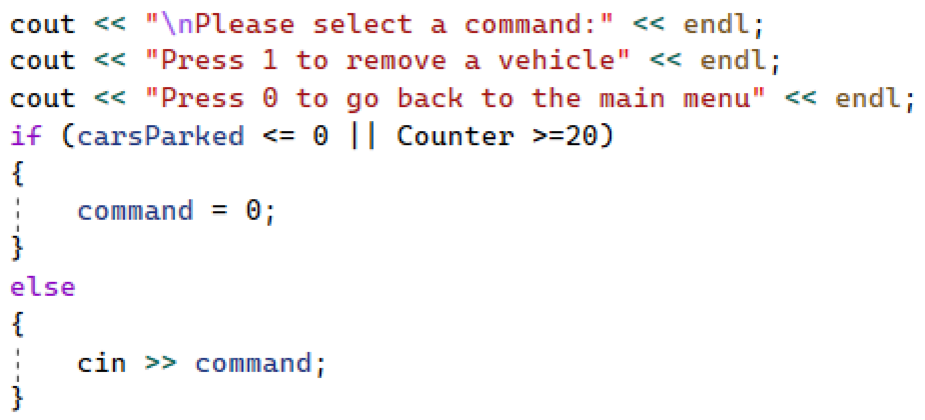

check carsParked >0
True
False
do loop
check counter >=20 or carsParked<=0
False
True
Exit case
Enter 0
Enter 0 or 1
Enter 1
for loop prints list of cars are parking
Input the ID of a car listed
for loop check pickID==each of listed IDs
True
False
simulate late fee prints fee using the randLateFee() then print fees
Continue
end of loop
for loop move latter vehicles up 1 position
break
Last vehicle replaced with empty spot
counter++
break

Figure 18: Pseudocode for removing a vehicle.

When the user press 2, method *"removeVehicle()"*, which is defined in class *"Manager"*, is called. At the beginning, there is a *"if-else"* condition to check if no vehicle has parked there or not. The *"if-else"* condition simply check that *"carsParked"* is larger than 0 or not, if it is false the program will anounce the user and end the method, otherwise it get into a *"do…while"* loop (1) which will repeat whenever the variable *"command"* is not 0. At the beginning of the loop (1), there is also is a *"if-else"* condition to re-check if no vehicle has parked there or not (Figure 19). If *"Counter"* is equal or larger than 20, or *"carsParked"* is equal or smaller than 0, *"command"* will be set to 0 and the program end. Otherwise, the user can enter input for *"command"* for further operations. The reason why it is placed there is because after each time the user successfully remove a vehicle, variable *"Counter"*and *"carsParked"* will change. Note that there are only 2 buttons 0, 1 that can activate further functions correctly, other circumstances may lead to many errors.

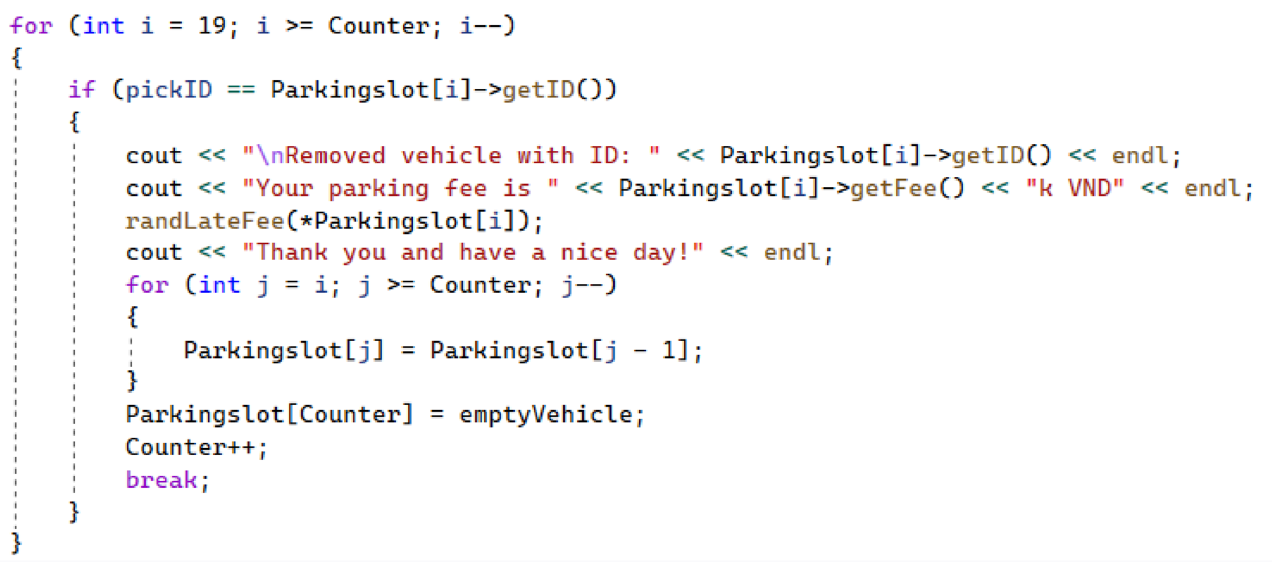


```
cout << "\nPlease select a command:" << endl;
cout << "Press 1 to remove a vehicle" << endl;
cout << "Press 0 to go back to the main menu" << endl;
if (carsParked <= 0 || Counter >=20)
{
    command = 0;
}
else
{
    cin >> command;
}
```

Figure 19: If-else condition for re-checking the emptiness of parking lot.

If the user press 0, the system will return to the home menu because the value of the variable *"command"* is set to 0.

If button 1 is pressed, a *"for"* loop (2) runs to display all ID in the screen, then the user will enter ID of the vehicle they want to remove. This ID is stored in variable *"pickID".* After that, another *"for"* loop (3) will runs to find that ID amongst IDs in the array *"Parkingslot[]"* (Figure 20). If the system finds the vehicle with that ID, it will display necessary information of that vehicle using *"getID()", "getFee()"* and *"randLateFee(Parkingslot[i])",* which will be explained later and stops the loop (3) immediately using *"break"* function. Then there will be another *"for"* loop (4), which starts at i and ends at *"Counter",* to shift other vehicles' position in the array and remove the selected vehicle (Figure 20) then starts a new loop (1). Otherwise, if the system cannot find the vehicle with that ID, it starts a new loop (1) too.



```
for (int i = 19; i >= Counter; i--)
{
    if (pickID == Parkingslot[i]->getID())
    {
        cout << "\nRemoved vehicle with ID: " << Parkingslot[i]->getID() << endl;
        cout << "Your parking fee is " << Parkingslot[i]->getFee() << "k VND" << endl;
        randLateFee(*Parkingslot[i]);
        cout << "Thank you and have a nice day!" << endl;
        for (int j = i; j >= Counter; j--)
        {
            Parkingslot[j] = Parkingslot[j - 1];
        }
        Parkingslot[Counter] = emptyVehicle;
        Counter++;
        break;
    }
}
```

Figure 20: For loop for removing the selected vehicle.

2.4. Displaying all vehicles

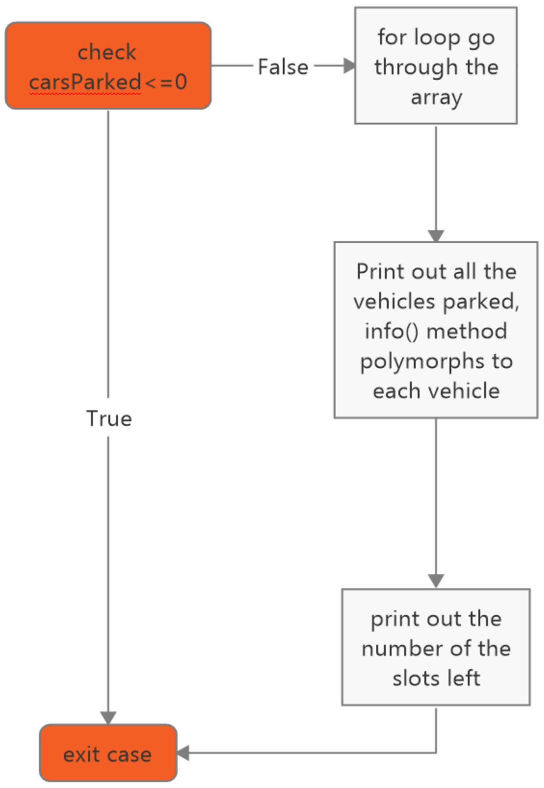

check carsParked<=0
False
for loop go through the array
Print out all the vehicles parked, info() method polymorphs to each vehicle
print out the number of the slots left
True
exit case

Figure 21: Pseudocode for displaying all vehicles.

When the user press 3, method *"display()"*, which is defined in class *"Manager"*, is called. This method only consists of one *"if-else"* condition for checking whether there is any vehicle in the array and one *"for"* loop to print information of all vehicles in the array *"Parkingslot[]"* using *"info()"* and a number of available slots(Figure 22).

```
int carsParked = 20 - Counter;
if (carsParked <= 0)
{
    cout << "\nThe car park is empty." << endl;
}
else
{
    cout << "\nThis is a list of vehicles parked: " << endl << endl;
    for (int i = 19; i >= Counter; i--)
    {
        Parkingslot[i]->info();
        cout << endl;
    }
    cout << "Free slots left: " << Counter << endl;
}
system("PAUSE");
system("CLS");
```

Figure 22: Program for displaying all vehicles.

2.5. Browsing a vehicle

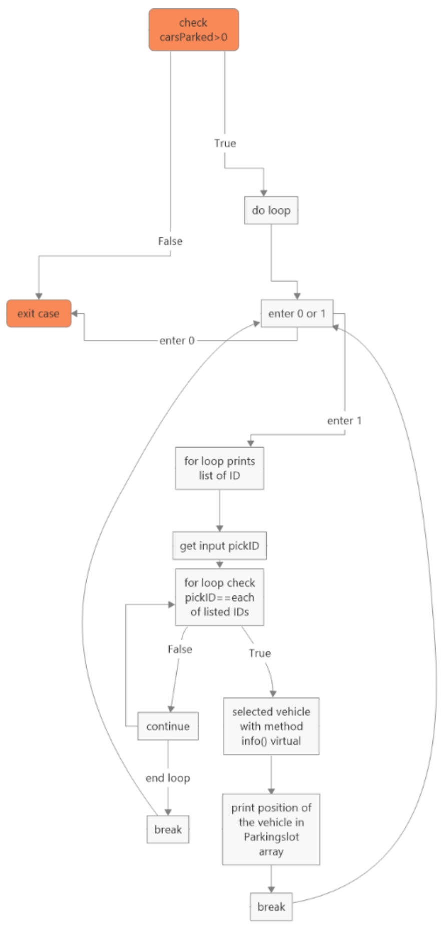
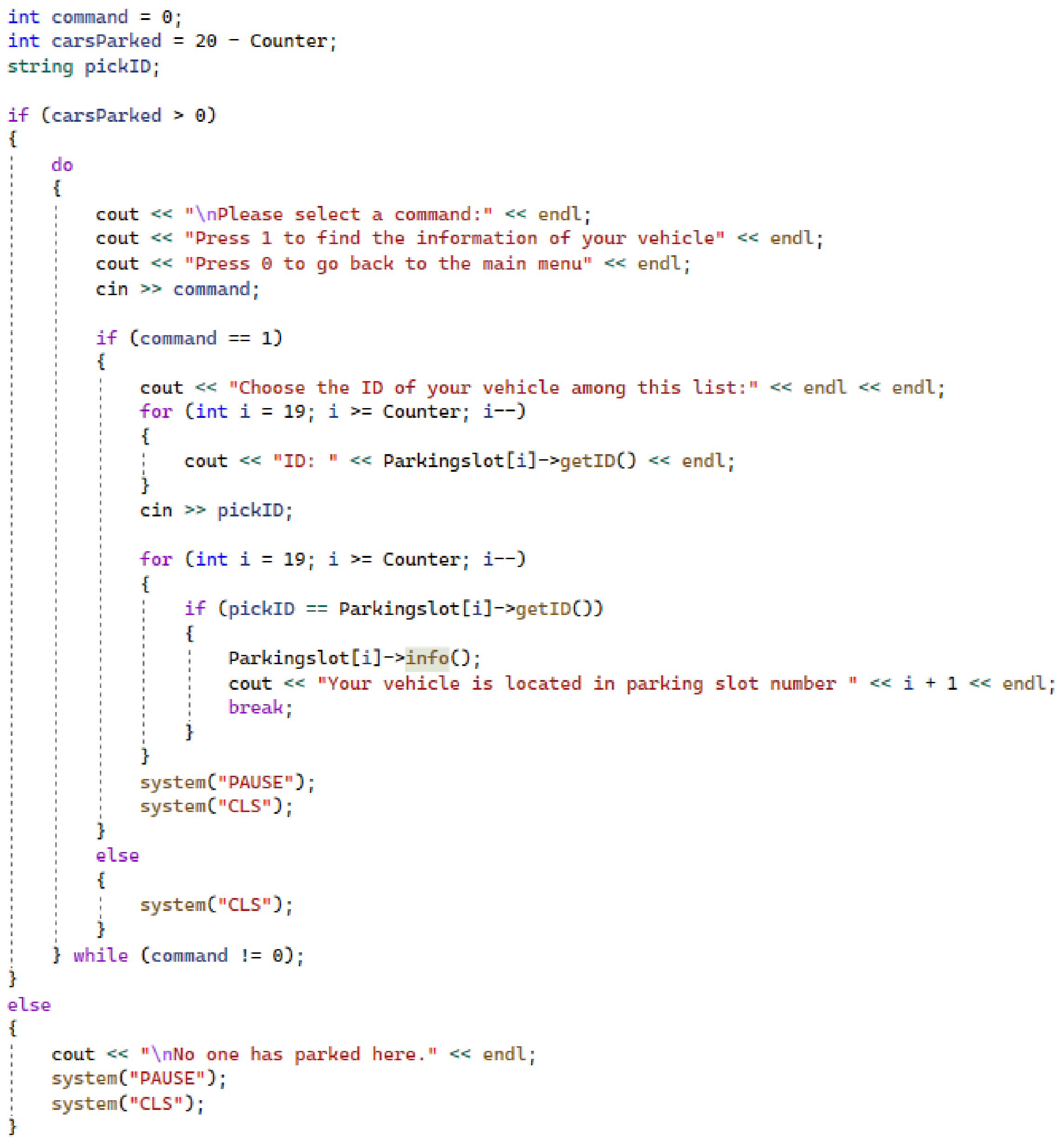

check carsParked>0
True
False
do loop
exit case
enter 0 or 1
enter 0
enter 1
for loop prints list of ID
get input pickID
for loop check pickID==each of listed IDs
False
True
continue
selected vehicle with method info() virtual
end loop
break
print position of the vehicle in Parkingslot array
break

Figure 23: Pseudocode for displaying all vehicles.

When the user press 4, method *"browseVehicle()"*, which is defined in class *"Manager"*, is called. At the beginning, there is a *"if-else"* condition to check if no vehicle has parked there or not. The *"if-else"* condition simply check that *"carsParked"* is larger than 0 or not, if it is false the program will anounce the user and end the method, otherwise it get into a *"do...while"* loop (1) which will repeat whenever the variable *"command"* is not 0. At the beginning of the loop (1), the user will enter input for *"command"*. Note that there are only 2 buttons 0, 1 that can activate further functions correctly, other circumstances may lead to many errors.

If the user press 0, the system will return to the home menu because the value of the variable *"command"* is set to 0.

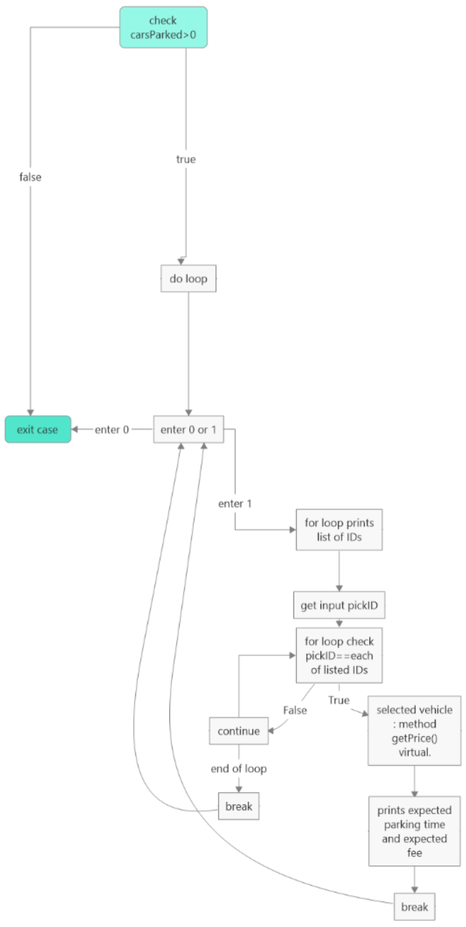
If button 1 is pressed, the system will get into a *"for"* loop (2) to print ID of all vehicles in the array *"Parkingslot[]"* using *"getID()"* (Figure 24) so the user can use the list to enter the ID they want to browse. This ID is stored in variable *"pickID"*. After that, another *"for"* loop (3) will runs to find that ID amongst IDs in the array *"Parkingslot[]"* (Figure 24). If the system finds the vehicle with that ID, it will display information of that vehicle using *"info()"*, and the position of the vehicle in the array then starts a new loop (1). Otherwise, if the system cannot find the vehicle with that ID, it starts a new loop (1) too.



```
int command = 0;
int carsParked = 20 - Counter;
string pickID;

if (carsParked > 0)
{
    do
    {
        cout << "\nPlease select a command:" << endl;
        cout << "Press 1 to find the information of your vehicle" << endl;
        cout << "Press 0 to go back to the main menu" << endl;
        cin >> command;

        if (command == 1)
        {
            cout << "Choose the ID of your vehicle among this list:" << endl << endl;
            for (int i = 19; i >= Counter; i--)
            {
                cout << "ID: " << Parkingslot[i]->getID() << endl;
            }
            cin >> pickID;

            for (int i = 19; i >= Counter; i--)
            {
                if (pickID == Parkingslot[i]->getID())
                {
                    Parkingslot[i]->info();
                    cout << "Your vehicle is located in parking slot number " << i + 1 << endl;
                    break;
                }
            }
            system("PAUSE");
            system("CLS");
        }
        else
        {
            system("CLS");
        }
    } while (command != 0);
}
else
{
    cout << "\nNo one has parked here." << endl;
    system("PAUSE");
    system("CLS");
}
```

Figure 24: Program for browsing a vehicle.

2.6. Calculating the parking fee of a vehicle



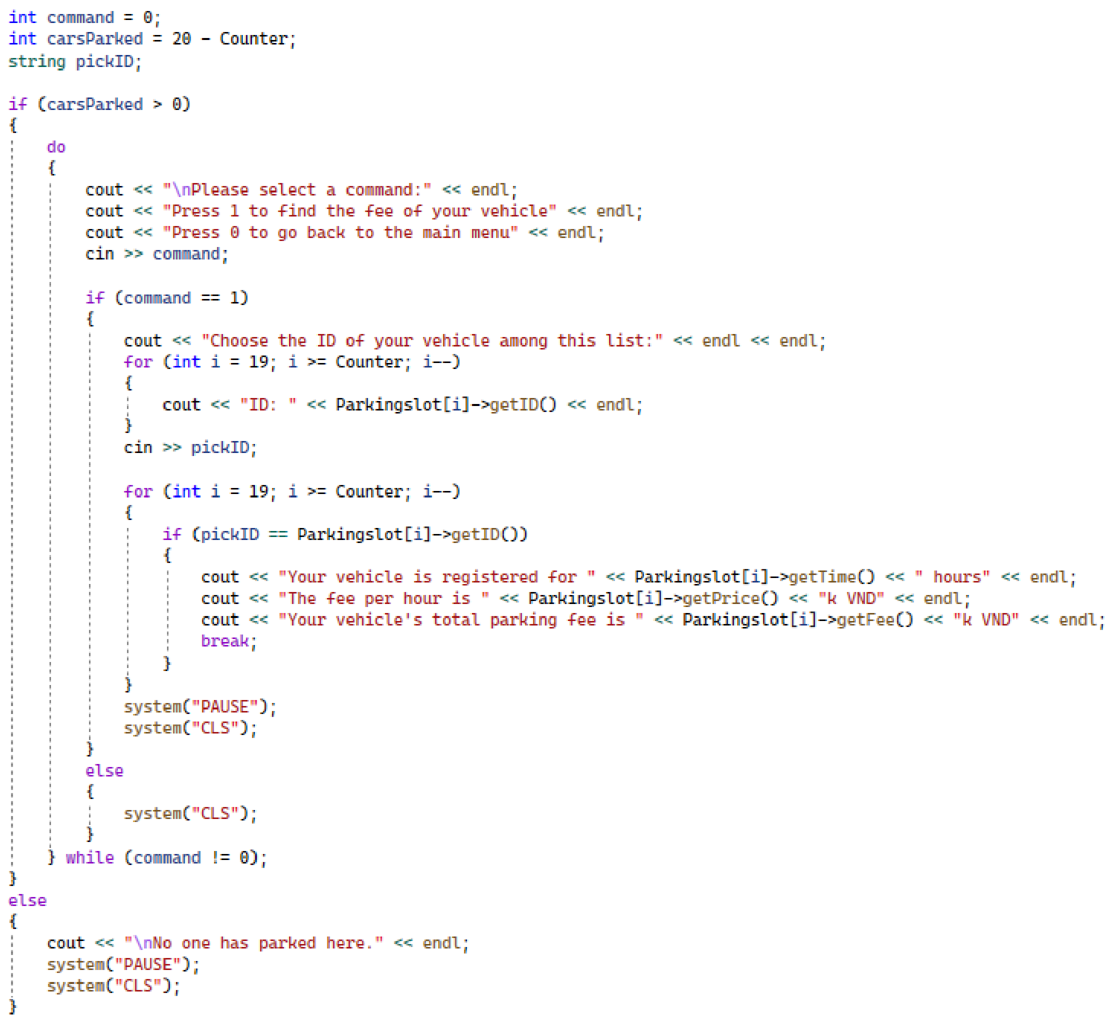
check carsParked>0
false
true
do loop
exit case
enter 0
enter 0 or 1
enter 1
for loop prints list of IDs
get input pickID
for loop check pickID==each of listed IDs
False
True
continue
selected vehicle : method getPrice() virtual.
end of loop
break
prints expected parking time and expected fee
break

Figure 25: Pseudocode for calculating the parking fee of a vehicle.

When the user press 5, method *"priceCalculation()"*, which is defined in class *"Manager"*, is called. At the beginning, there is an *"if-else"* condition to check if no vehicle has parked there or not. The *"if-else"* condition simply check that *"carsParked"* is larger than 0 or not, if it is false the program will anounce the user and end the method, otherwise it get into a *"do...while"* loop (1) which will repeat whenever the variable *"command"* is not 0. At the beginning of the loop (1), the user will enter input for *"command"*. Note that there are only 2 buttons 0, 1 that can activate further functions correctly, other circumstances may lead to many errors.

If the user press 0, the system will return to the home menu because the value of the variable *"command"* is set to 0.

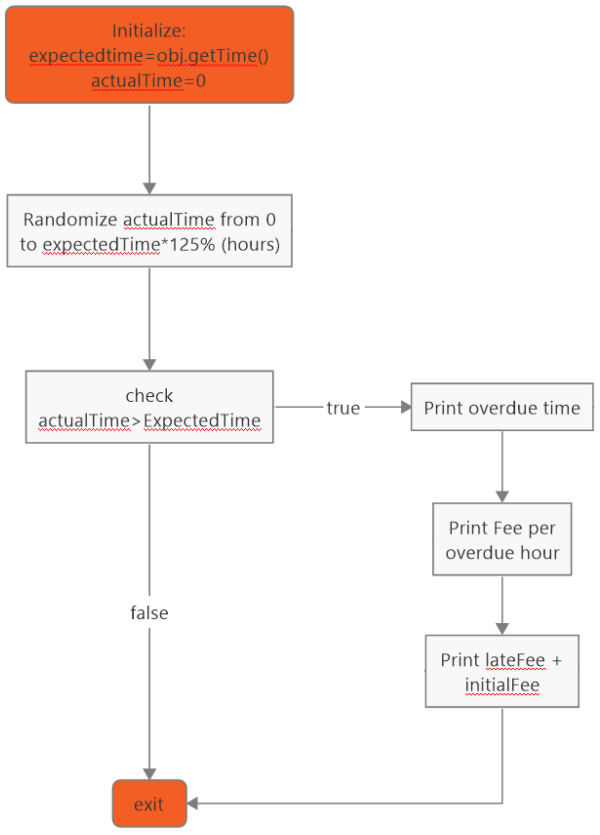
If button 1 is pressed, the system will get into a *"for"* loop (2) to print ID of all vehicles in the array *"Parkingslot[]"* using *"getID()"* (Figure 26) so the user can use the list to enter the ID they want to browse. This ID is stored in variable *"pickID".* After that, another *"for"* loop (3) will runs to find that ID amongst IDs in the array *"Parkingslot[]"* (Figure 26). If the system finds the vehicle with that ID, it will display parking fee according to registered parking time of that vehicle using *"getTime()", "getPrice()", "getFee()"* then starts a new loop (1). Otherwise, if the system cannot find the vehicle with that ID, it starts a new loop (1) too.



```cpp
int command = 0;
int carsParked = 20 - Counter;
string pickID;

if (carsParked > 0)
{
    do
    {
        cout << "\nPlease select a command:" << endl;
        cout << "Press 1 to find the fee of your vehicle" << endl;
        cout << "Press 0 to go back to the main menu" << endl;
        cin >> command;

        if (command == 1)
        {
            cout << "Choose the ID of your vehicle among this list:" << endl << endl;
            for (int i = 19; i >= Counter; i--)
            {
                cout << "ID: " << Parkingslot[i]->getID() << endl;
            }
            cin >> pickID;

            for (int i = 19; i >= Counter; i--)
            {
                if (pickID == Parkingslot[i]->getID())
                {
                    cout << "Your vehicle is registered for " << Parkingslot[i]->getTime() << " hours" << endl;
                    cout << "The fee per hour is " << Parkingslot[i]->getPrice() << "k VND" << endl;
                    cout << "Your vehicle's total parking fee is " << Parkingslot[i]->getFee() << "k VND" << endl;
                    break;
                }
            }
            system("PAUSE");
            system("CLS");
        }
        else
        {
            system("CLS");
        }
    } while (command != 0);
}
else
{
    cout << "\nNo one has parked here." << endl;
    system("PAUSE");
    system("CLS");
}
```

Figure 26: Program for calculating the parking fee of a vehicle.

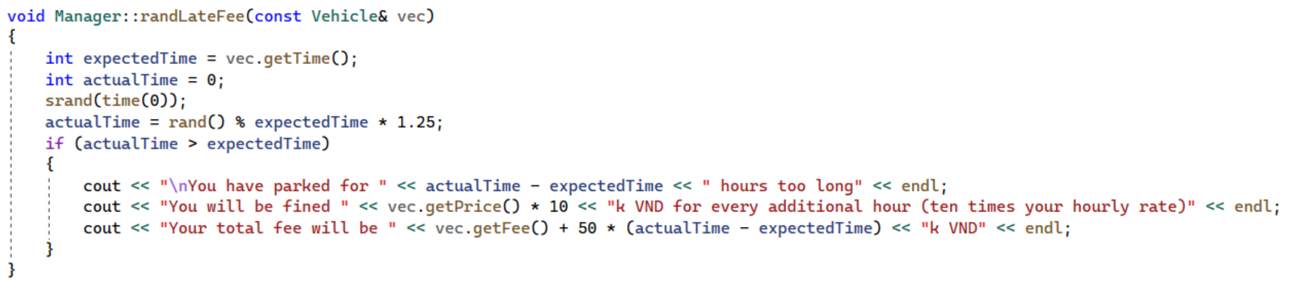
2.7. Simulating actual parking time:



Initialize:
expectedtime=obj.getTime()
actualTime=0
Randomize actualTime from 0 to expectedTime*125% (hours)
check actualTime>ExpectedTime
true
Print overdue time
Print Fee per overdue hour
Print lateFee + initialFee
false
exit

Figure 27: Pseudocode for simulating actual parking time.

The method for simulating actual parking time is called *"randLateFee(const Vehicle& vec)".* It is defined in class Manager (Figure 28). When it is called, it first initializes two variables *"expectedTime", "actualTime"* and set them to registered parking time of the selected vehicle using "vec.getTime()" and 0 respectively. After that, the program uses *"random()"* function to randomize actual parking time of the vehicle with the range from 0 to 125% of registered one. Then an *"if-else"* condition runs to check if the vehicle exceed its registered parking time or not. If *"actualTime"* is smaller than *"expectedTime"*, the method simply ends. Otherwise, it will print overdue time, fee per overdue hour and total parking fee to the screen then ends.



```
void Manager::randLateFee(const Vehicle& vec)
{
    int expectedTime = vec.getTime();
    int actualTime = 0;
    srand(time(0));
    actualTime = rand() % expectedTime * 1.25;
    if (actualTime > expectedTime)
    {
        cout << "\nYou have parked for " << actualTime - expectedTime << " hours too long" << endl;
        cout << "You will be fined " << vec.getPrice() * 10 << "k VND for every additional hour (ten times your hourly rate)" << endl;
        cout << "Your total fee will be " << vec.getFee() + 50 * (actualTime - expectedTime) << "k VND" << endl;
    }
}
```

Figure 28: Program for simulating actual parking time.

IV. Discussion

Our program has operated exactly as the expectation. It is able to operate commands adding, removing, displaying, browsing and calculating correctly. However, there are still some problems that have not been completely addressed because of our limitation in terms of time and knowledge. Firstly, the program misses a command that allows the administrator to stop a current command and return to the home menu in case of there is a mistake . Secondly, there is no function to ask typers to provide a different input when they type incorrectly their ID, parking time. The current program just ended and send errors instantly if there are errors with the data type of inputs. Lastly, we have not develop a program for users, so this is still not a complete program that can effectively manage a parking lot.

V. Conclusion

The project has successfully met our team's targets as a simple back-end program for the purpose of effectively managing a parking lot in spite of some limitations. In the future, we intend to research more to improve the performance of this program, specifically exception handling, separating login for administrator and user, using different container, and getting information in real time, then apply it at VGU if possible.

VI. Task Division

ID	Task	In Charge	Start	End	State	Note
1	Idea	Lê Quốc Hoàng	20/11	20/11	Done	Parking Manager
2	Prototype 1	Cao Ngọc Nhật Huy	21/11	05/12	Done	Simplify the reference code
3	Prototype 2	Lê Quốc Hoàng	27/11	05/12	Done	Simplify the reference code and build on the prototype 1
4	Debugging 1	Lê Tuấn Minh Lê Quốc Hoàng	05/12	10/12	Done	Strings and polymorph issues
5	Protoype 3	Cao Ngọc Nhật Huy	10/12	17/12	Done	Build on the prototype 1 and 2; fix polymorphism
6	Debugging 2	Lê Tuấn Minh Cao Ngọc Nhật Huy	17/12	24/12	Done	Add randLateFee() and ID's uniqueness checker
7	Flowchart Drawing	Cao Ngọc Nhật Huy Lê Quốc Hoàng	26/12	28/12	Done	

8	Report Writing	Lê Tuấn Minh	26/12	28/12	Done	
9	Finish Report	All	28/12	29/12	Done	

VII. Reference

https://github.com/mcnugets/Car-parking-manager