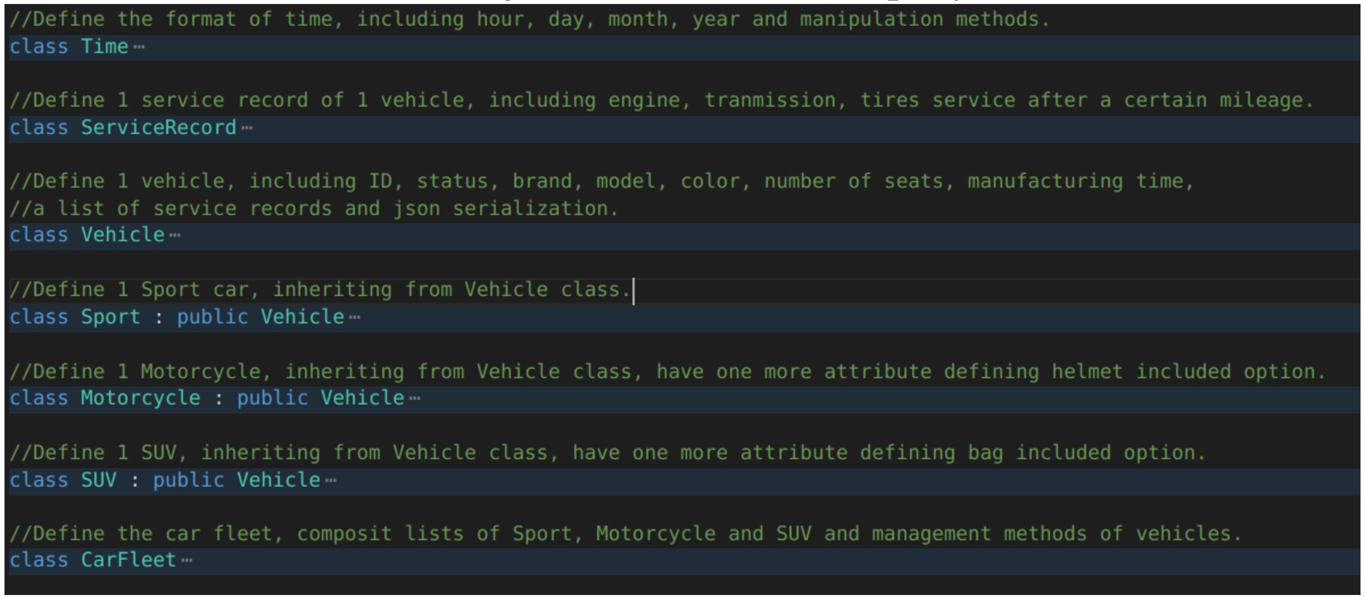
LAB Report

Course: Advanced Programming
Your Name: HỒ HOÀNG THIÊN LONG
Student ID: 1852161

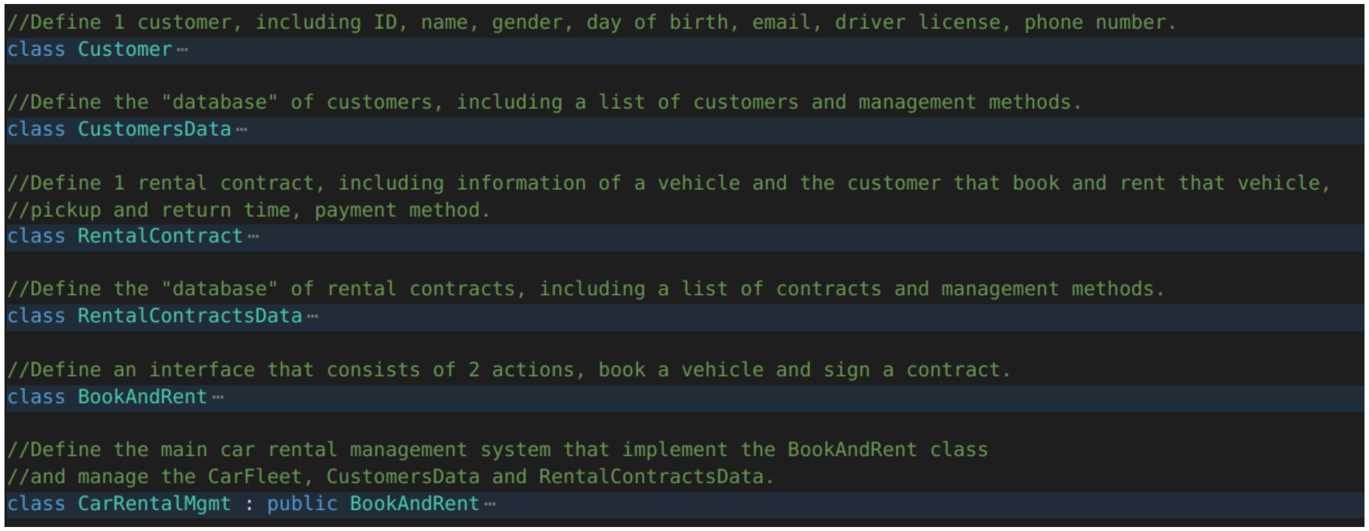
Note: Because the whole source code is very long, I just focus on the main ideas and parts that need to be explained. For more information about the source code, please check the git repository that I implemented https://github.com/goriummaximum/simple-car-rental-manager.git .

	Source code (an image captured from your IDE)	Demos Screenshot	Note
How to Think in Terms of Objects: A car rental company			Page 2
Overloading in OO: Car rental company revisited			Page 5
Inheritance and Composition			Page 8
Re-define your interfaces / abstract classes			Page 12
Operator Overloading			Page 16
Serializing the service history of a rental car			Page 18

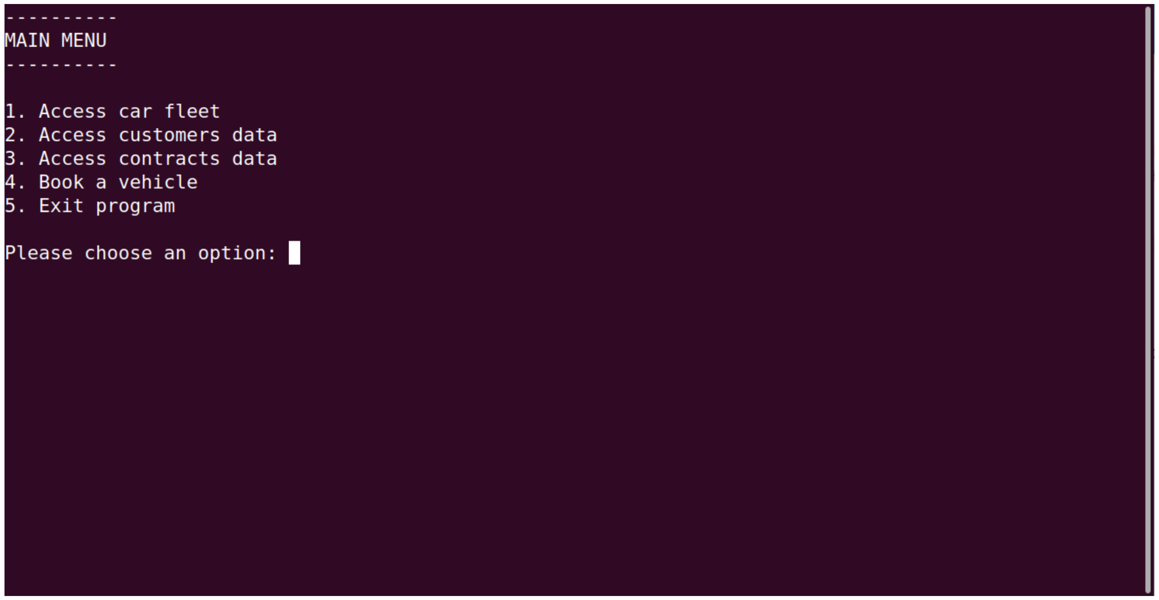
How to Think in Terms of Objects: A car rental company



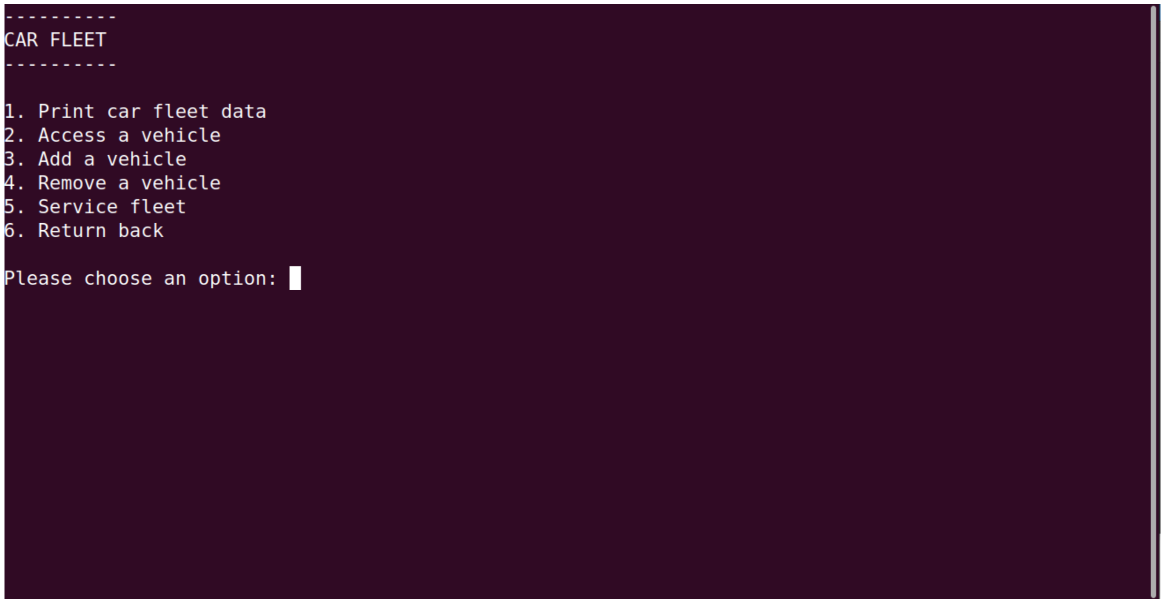
```
//Define the format of time, including hour, day, month, year and manipulation methods.
class Time⋯

//Define 1 service record of 1 vehicle, including engine, tranmission, tires service after a certain mileage.
class ServiceRecord⋯

//Define 1 vehicle, including ID, status, brand, model, color, number of seats, manufacturing time,
//a list of service records and json serialization.
class Vehicle⋯

//Define 1 Sport car, inheriting from Vehicle class.
class Sport : public Vehicle⋯

//Define 1 Motorcycle, inheriting from Vehicle class, have one more attribute defining helmet included option.
class Motorcycle : public Vehicle⋯

//Define 1 SUV, inheriting from Vehicle class, have one more attribute defining bag included option.
class SUV : public Vehicle⋯

//Define the car fleet, composit lists of Sport, Motorcycle and SUV and management methods of vehicles.
class CarFleet⋯
```



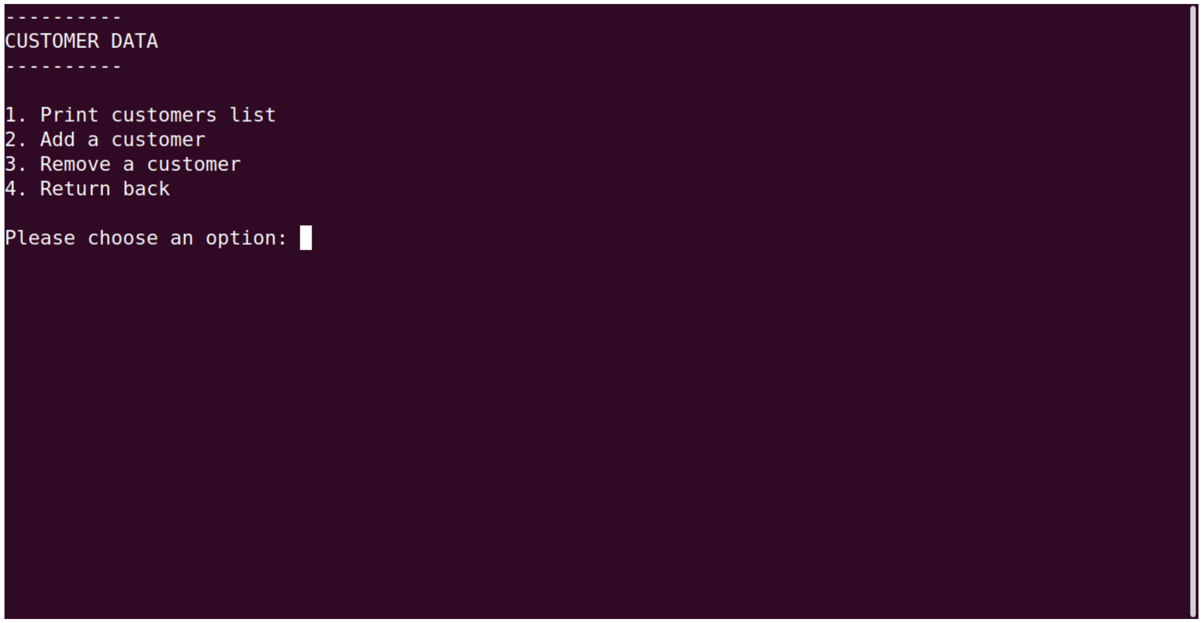
```
//Define 1 customer, including ID, name, gender, day of birth, email, driver license, phone number.
class Customer⋯

//Define the "database" of customers, including a list of customers and management methods.
class CustomersData⋯

//Define 1 rental contract, including information of a vehicle and the customer that book and rent that vehicle,
//pickup and return time, payment method.
class RentalContract⋯

//Define the "database" of rental contracts, including a list of contracts and management methods.
class RentalContractsData⋯

//Define an interface that consists of 2 actions, book a vehicle and sign a contract.
class BookAndRent⋯

//Define the main car rental management system that implement the BookAndRent class
//and manage the CarFleet, CustomersData and RentalContractsData.
class CarRentalMgmt : public BookAndRent⋯
```



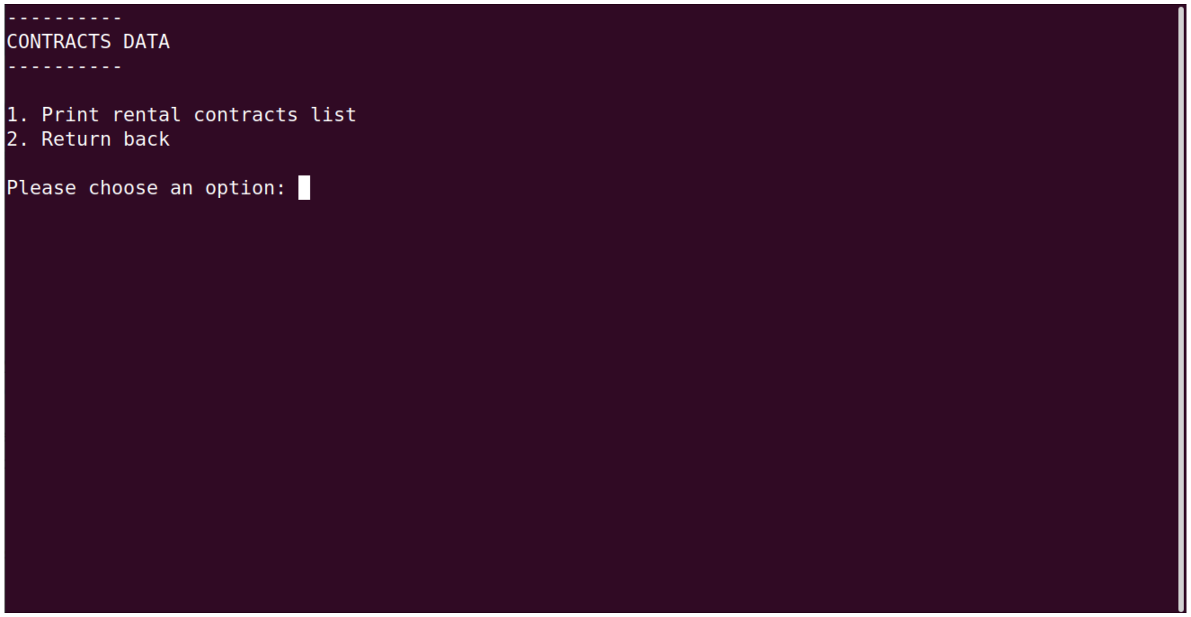
```
----------
MAIN MENU
----------

1. Access car fleet
2. Access customers data
3. Access contracts data
4. Book a vehicle
5. Exit program

Please choose an option: 
```



```
----------
CAR FLEET
----------

1. Print car fleet data
2. Access a vehicle
3. Add a vehicle
4. Remove a vehicle
5. Service fleet
6. Return back

Please choose an option: 
```



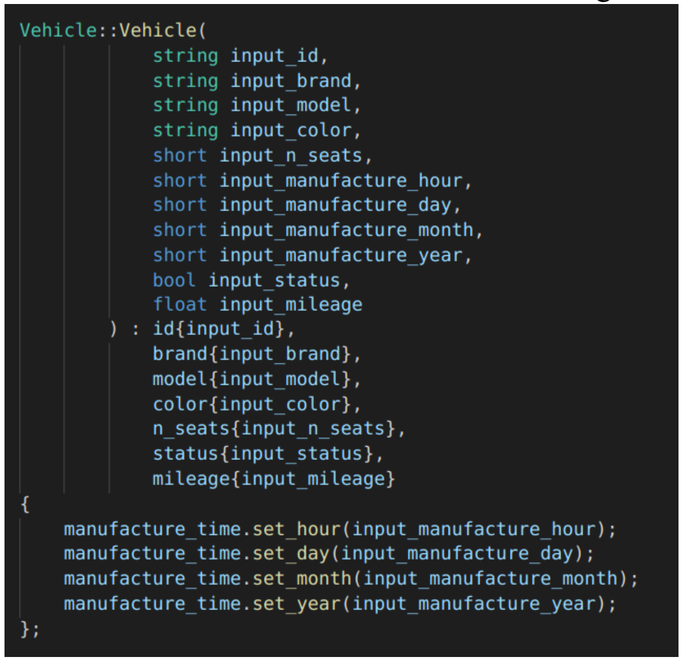
```
----------
CUSTOMER DATA
----------

1. Print customers list
2. Add a customer
3. Remove a customer
4. Return back

Please choose an option: 
```



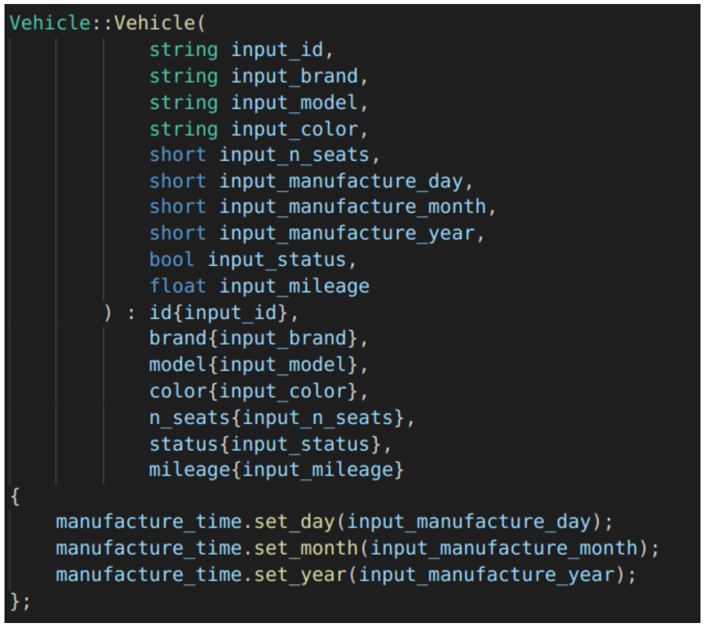
```
----------
CONTRACTS DATA
----------

1. Print rental contracts list
2. Return back

Please choose an option: 
```

Overloading in OO: Car rental company revisited

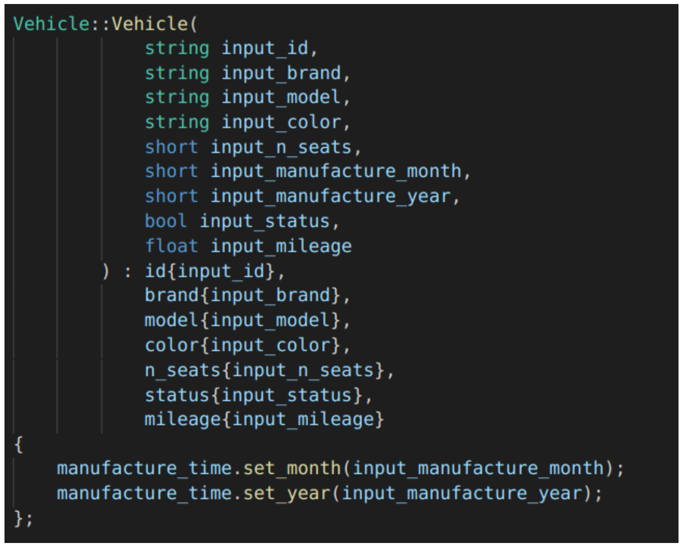
Overloading Vehicle with 4 constructors, defining the different specificity of manufacturing time. Therefore, child classes inheriting this class will have the same constructor overloading.



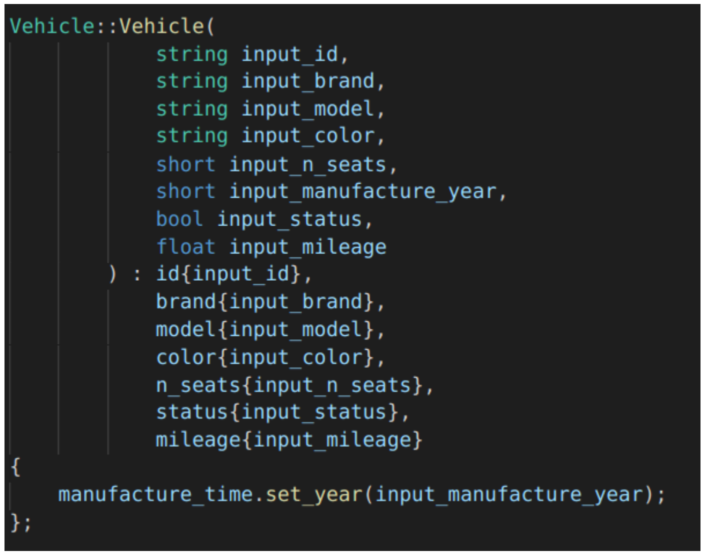
```
Vehicle::Vehicle(
            string input_id,
            string input_brand,
            string input_model,
            string input_color,
            short input_n_seats,
            short input_manufacture_hour,
            short input_manufacture_day,
            short input_manufacture_month,
            short input_manufacture_year,
            bool input_status,
            float input_mileage
        ) : id{input_id},
            brand{input_brand},
            model{input_model},
            color{input_color},
            n_seats{input_n_seats},
            status{input_status},
            mileage{input_mileage}
{
    manufacture_time.set_hour(input_manufacture_hour);
    manufacture_time.set_day(input_manufacture_day);
    manufacture_time.set_month(input_manufacture_month);
    manufacture_time.set_year(input_manufacture_year);
};
```



```
Vehicle::Vehicle(
            string input_id,
            string input_brand,
            string input_model,
            string input_color,
            short input_n_seats,
            short input_manufacture_day,
            short input_manufacture_month,
            short input_manufacture_year,
            bool input_status,
            float input_mileage
        ) : id{input_id},
            brand{input_brand},
            model{input_model},
            color{input_color},
            n_seats{input_n_seats},
            status{input_status},
            mileage{input_mileage}
{
    manufacture_time.set_day(input_manufacture_day);
    manufacture_time.set_month(input_manufacture_month);
    manufacture_time.set_year(input_manufacture_year);
};
```

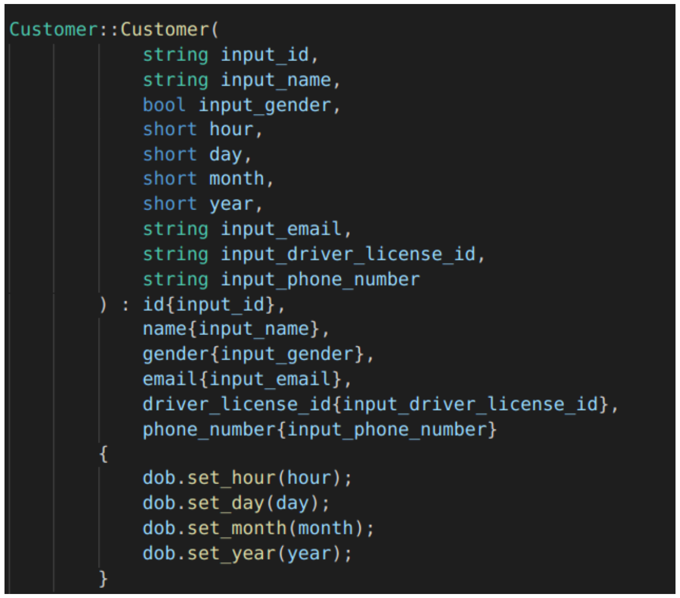


```
Vehicle::Vehicle(
            string input_id,
            string input_brand,
            string input_model,
            string input_color,
            short input_n_seats,
            short input_manufacture_month,
            short input_manufacture_year,
            bool input_status,
            float input_mileage
        ) : id{input_id},
            brand{input_brand},
            model{input_model},
            color{input_color},
            n_seats{input_n_seats},
            status{input_status},
            mileage{input_mileage}
{
    manufacture_time.set_month(input_manufacture_month);
    manufacture_time.set_year(input_manufacture_year);
};
```

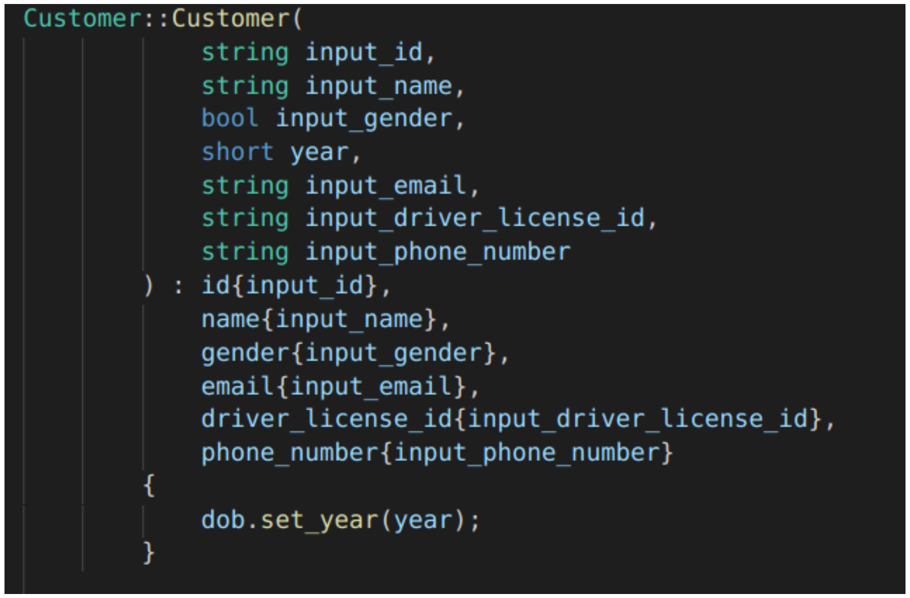


```
Vehicle::Vehicle(
            string input_id,
            string input_brand,
            string input_model,
            string input_color,
            short input_n_seats,
            short input_manufacture_year,
            bool input_status,
            float input_mileage
        ) : id{input_id},
            brand{input_brand},
            model{input_model},
            color{input_color},
            n_seats{input_n_seats},
            status{input_status},
            mileage{input_mileage}
{
    manufacture_time.set_year(input_manufacture_year);
};
```

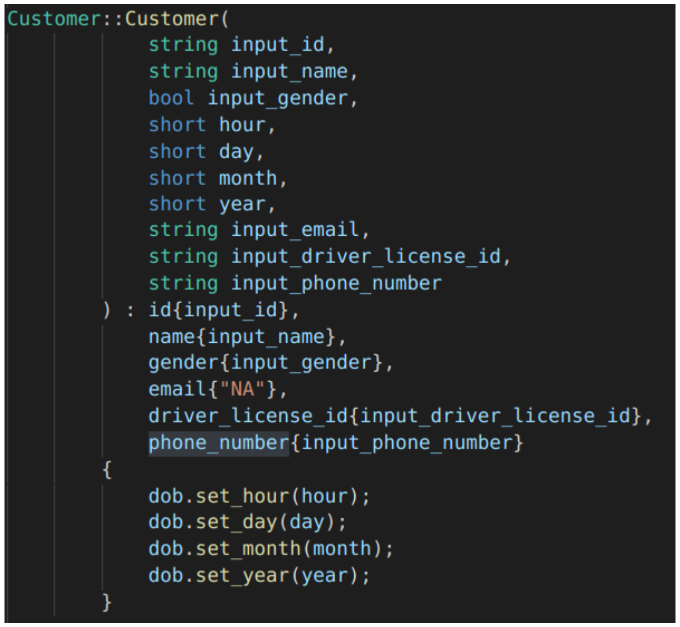
4 constructor overloadings of Customer, there are 2 variables that need overloading, day of birth and email. Day of birth is overloaded with 2 different specificity of time; Email is overloaded with 2 different cases, have or do not have. Therefore, by combining these 2 overloads, there are 2x2=4 overloadings.



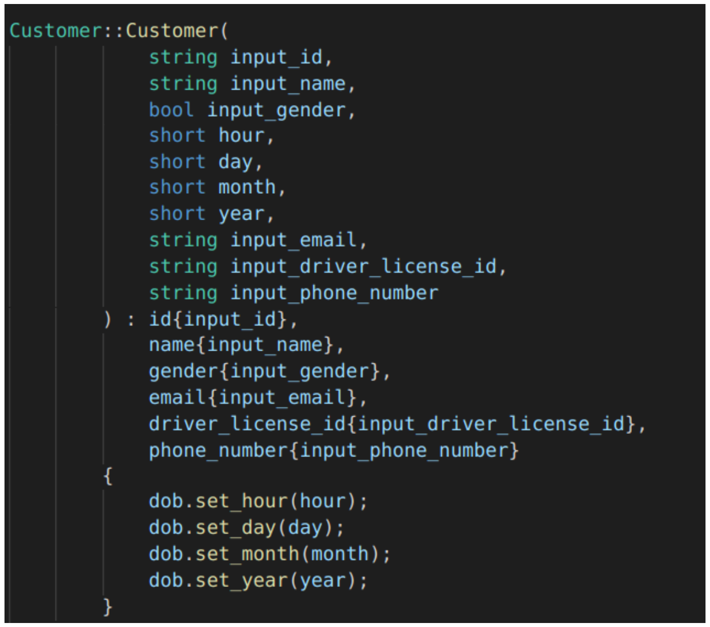
```cpp
Customer::Customer(
            string input_id,
            string input_name,
            bool input_gender,
            short hour,
            short day,
            short month,
            short year,
            string input_email,
            string input_driver_license_id,
            string input_phone_number
        ) : id{input_id},
            name{input_name},
            gender{input_gender},
            email{input_email},
            driver_license_id{input_driver_license_id},
            phone_number{input_phone_number}
        {
            dob.set_hour(hour);
            dob.set_day(day);
            dob.set_month(month);
            dob.set_year(year);
        }
```



```cpp
Customer::Customer(
            string input_id,
            string input_name,
            bool input_gender,
            short year,
            string input_email,
            string input_driver_license_id,
            string input_phone_number
        ) : id{input_id},
            name{input_name},
            gender{input_gender},
            email{input_email},
            driver_license_id{input_driver_license_id},
            phone_number{input_phone_number}
        {
            dob.set_year(year);
        }
```

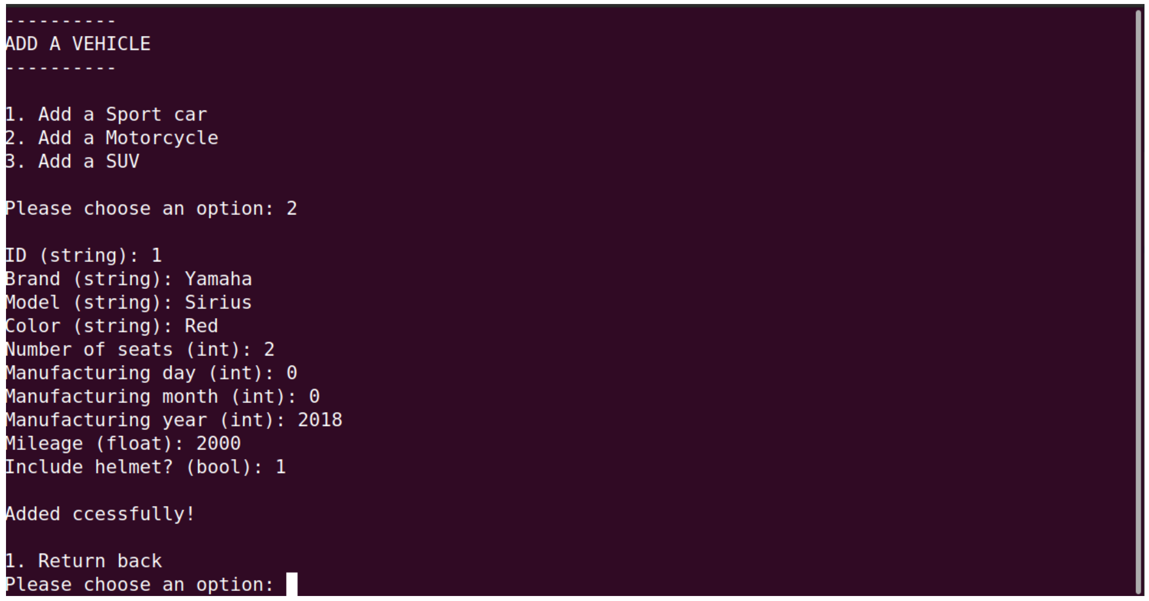


```cpp
Customer::Customer(
            string input_id,
            string input_name,
            bool input_gender,
            short hour,
            short day,
            short month,
            short year,
            string input_email,
            string input_driver_license_id,
            string input_phone_number
        ) : id{input_id},
            name{input_name},
            gender{input_gender},
            email{"NA"},
            driver_license_id{input_driver_license_id},
            phone_number{input_phone_number}
        {
            dob.set_hour(hour);
            dob.set_day(day);
            dob.set_month(month);
            dob.set_year(year);
        }
```



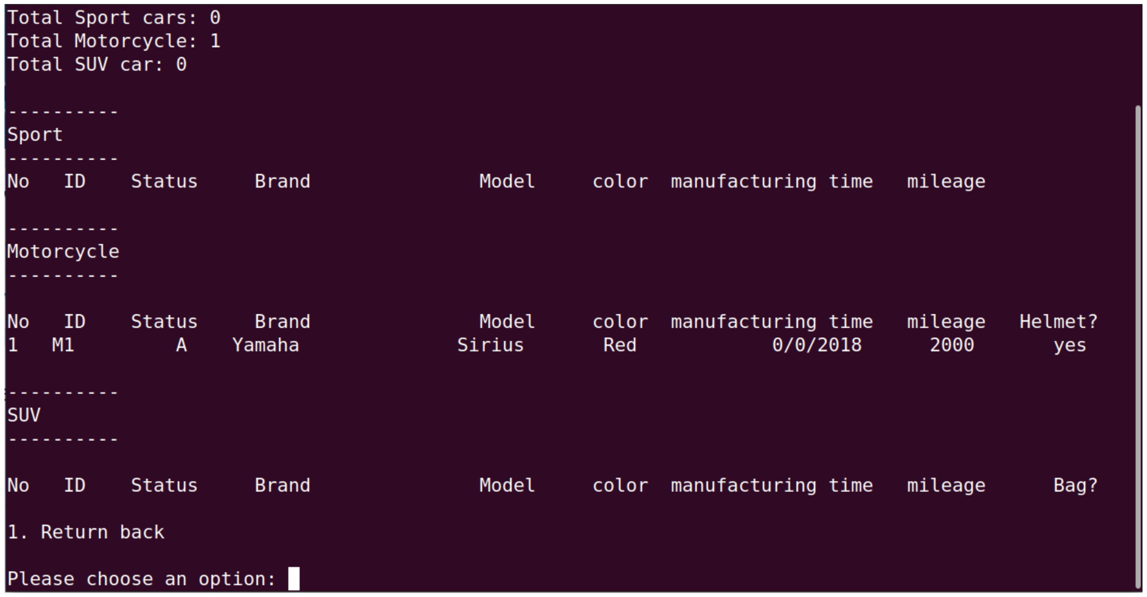
```cpp
Customer::Customer(
            string input_id,
            string input_name,
            bool input_gender,
            short hour,
            short day,
            short month,
            short year,
            string input_email,
            string input_driver_license_id,
            string input_phone_number
        ) : id{input_id},
            name{input_name},
            gender{input_gender},
            email{input_email},
            driver_license_id{input_driver_license_id},
            phone_number{input_phone_number}
        {
            dob.set_hour(hour);
            dob.set_day(day);
            dob.set_month(month);
            dob.set_year(year);
        }
```

Demo adding a vehicle, specifically a motorcycle.



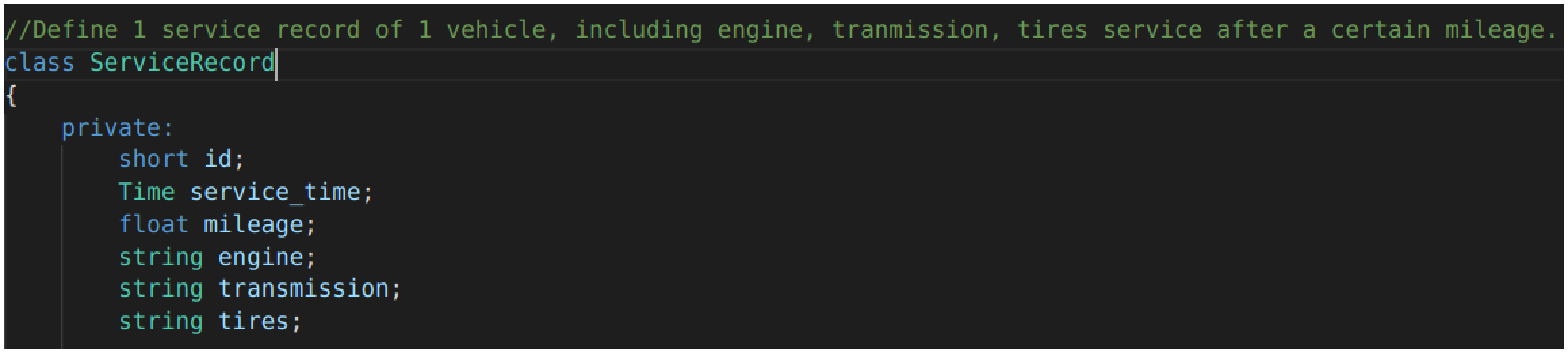
```
----------
ADD A VEHICLE
----------

1. Add a Sport car
2. Add a Motorcycle
3. Add a SUV

Please choose an option: 2

ID (string): 1
Brand (string): Yamaha
Model (string): Sirius
Color (string): Red
Number of seats (int): 2
Manufacturing day (int): 0
Manufacturing month (int): 0
Manufacturing year (int): 2018
Mileage (float): 2000
Include helmet? (bool): 1

Added ccessfully!

1. Return back
Please choose an option: 
```

After added successfully, we can check it by call print command. This motorcycle has an ID M1.



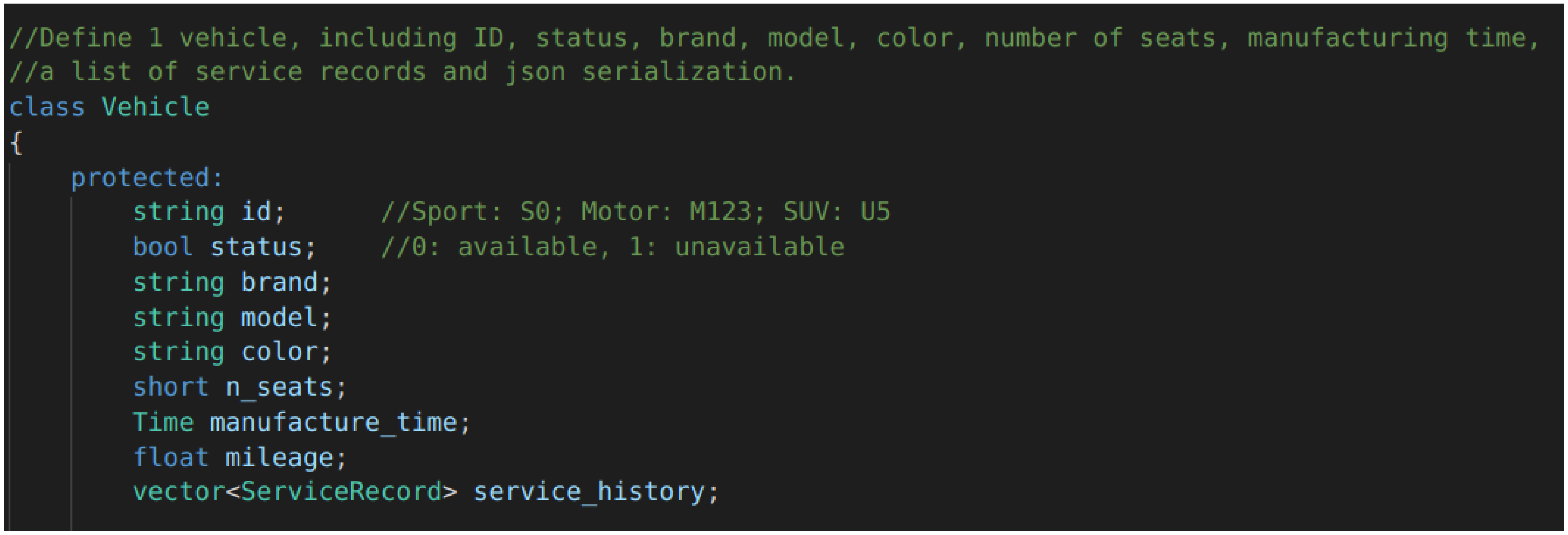
```
Total Sport cars: 0
Total Motorcycle: 1
Total SUV car: 0

----------
Sport
----------
No   ID    Status     Brand               Model     color  manufacturing time   mileage

----------
Motorcycle
----------

No   ID    Status     Brand               Model     color  manufacturing time   mileage   Helmet?
1   M1         A    Yamaha              Sirius       Red            0/0/2018      2000       yes

----------
SUV
----------

No   ID    Status     Brand               Model     color  manufacturing time   mileage      Bag?

1. Return back

Please choose an option: 
```

Inheritance and Composition



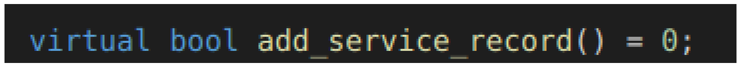
```
//Define 1 service record of 1 vehicle, including engine, tranmission, tires service after a certain mileage.
class ServiceRecord
{
    private:
        short id;
        Time service_time;
        float mileage;
        string engine;
        string transmission;
        string tires;
```

Vehicle composit a list of service record, call service_history.



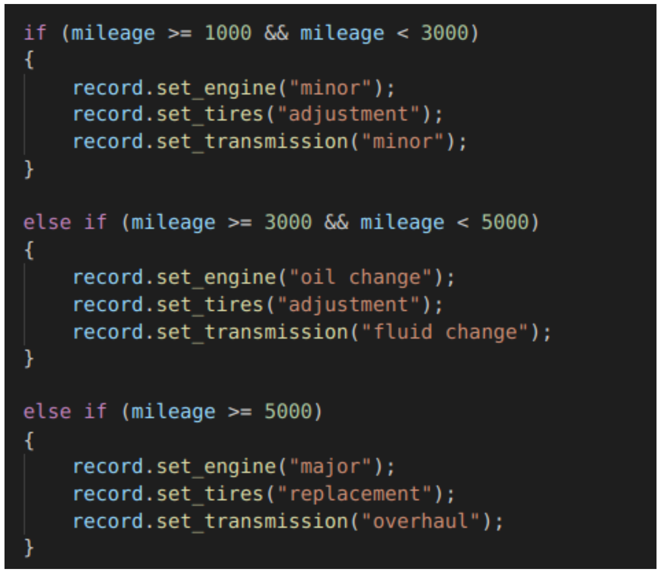
```
//Define 1 vehicle, including ID, status, brand, model, color, number of seats, manufacturing time,
//a list of service records and json serialization.
class Vehicle
{
    protected:
        string id;      //Sport: S0; Motor: M123; SUV: U5
        bool status;    //0: available, 1: unavailable
        string brand;
        string model;
        string color;
        short n_seats;
        Time manufacture_time;
        float mileage;
        vector<ServiceRecord> service_history;
```

Vehicle will have a virtual method add_service_record, which is implemented differently in each type of Vehicle based on mileage.



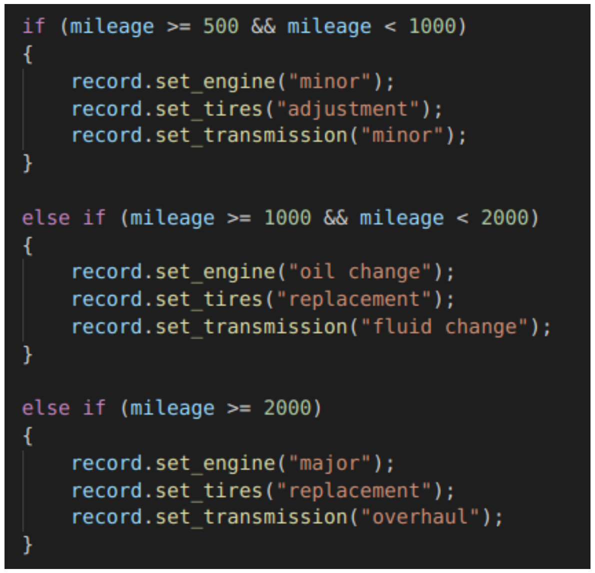
```
 virtual bool add_service_record() = 0;
```

Motorcycle



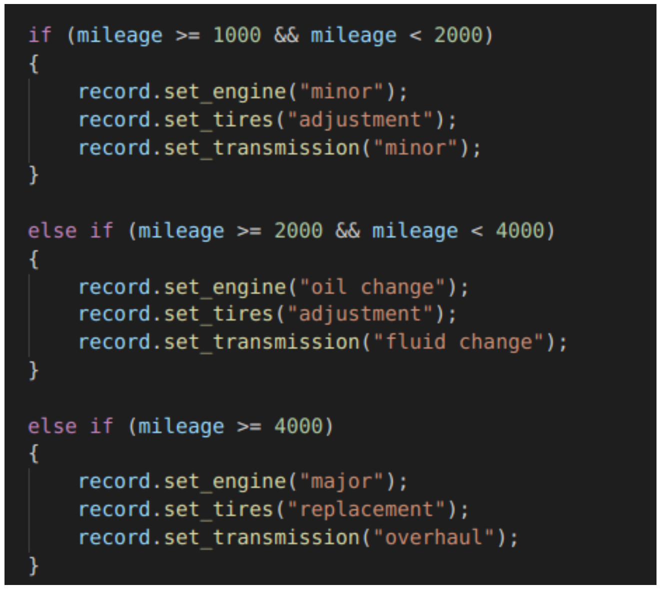
```
if (mileage >= 1000 && mileage < 3000)
{
    record.set_engine("minor");
    record.set_tires("adjustment");
    record.set_transmission("minor");
}

else if (mileage >= 3000 && mileage < 5000)
{
    record.set_engine("oil change");
    record.set_tires("adjustment");
    record.set_transmission("fluid change");
}

else if (mileage >= 5000)
{
    record.set_engine("major");
    record.set_tires("replacement");
    record.set_transmission("overhaul");
}
```

Sport



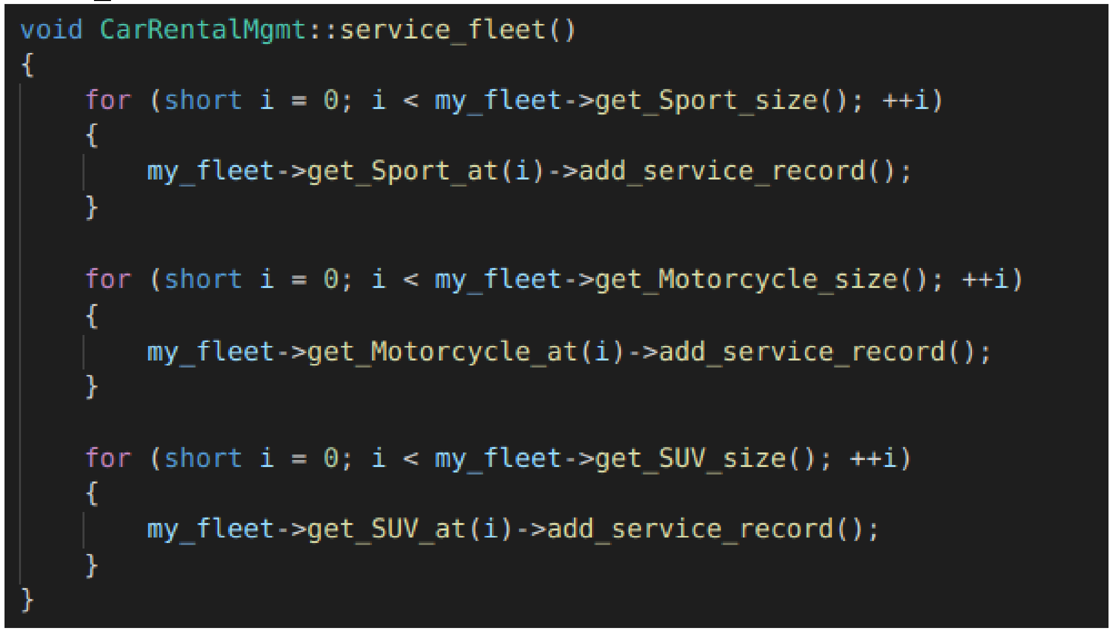
```
if (mileage >= 500 && mileage < 1000)
{
    record.set_engine("minor");
    record.set_tires("adjustment");
    record.set_transmission("minor");
}

else if (mileage >= 1000 && mileage < 2000)
{
    record.set_engine("oil change");
    record.set_tires("replacement");
    record.set_transmission("fluid change");
}

else if (mileage >= 2000)
{
    record.set_engine("major");
    record.set_tires("replacement");
    record.set_transmission("overhaul");
}
```

SUV



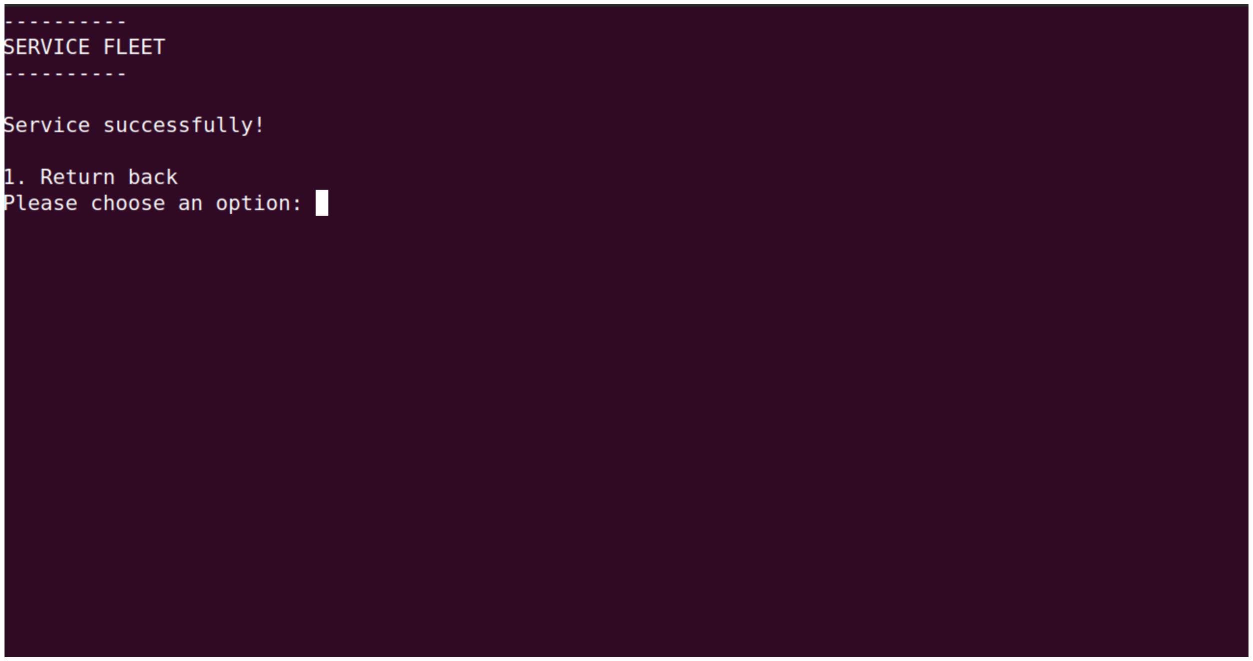
```
if (mileage >= 1000 && mileage < 2000)
{
    record.set_engine("minor");
    record.set_tires("adjustment");
    record.set_transmission("minor");
}

else if (mileage >= 2000 && mileage < 4000)
{
    record.set_engine("oil change");
    record.set_tires("adjustment");
    record.set_transmission("fluid change");
}

else if (mileage >= 4000)
{
    record.set_engine("major");
    record.set_tires("replacement");
    record.set_transmission("overhaul");
}
```

service_fleet

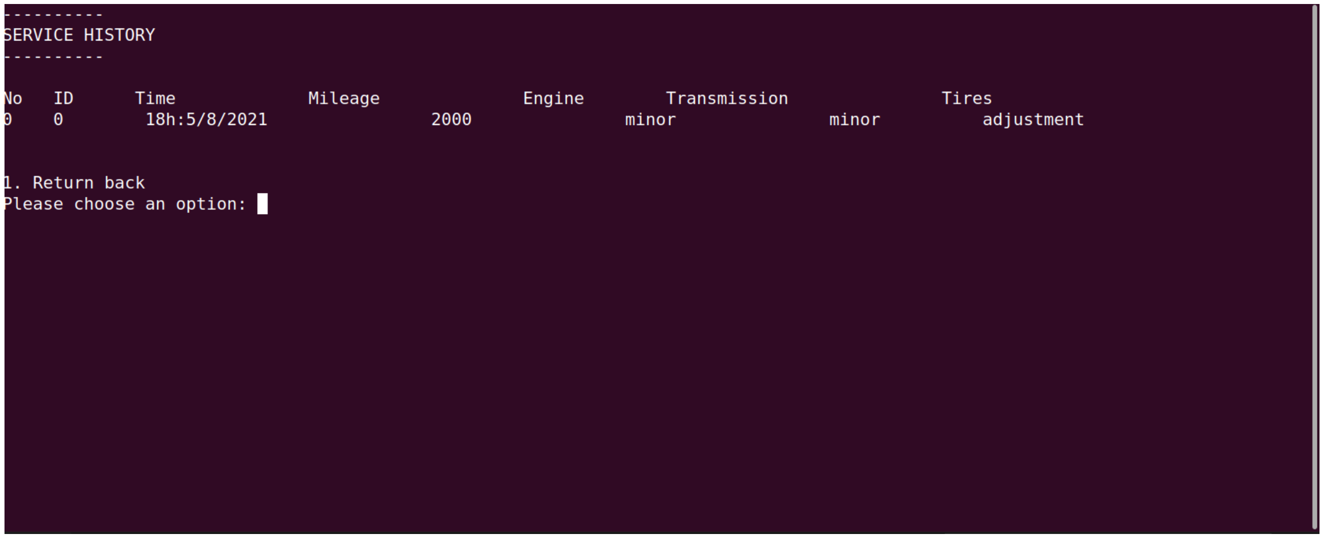


```cpp
void CarRentalMgmt::service_fleet()
{
    for (short i = 0; i < my_fleet->get_Sport_size(); ++i)
    {
        my_fleet->get_Sport_at(i)->add_service_record();
    }

    for (short i = 0; i < my_fleet->get_Motorcycle_size(); ++i)
    {
        my_fleet->get_Motorcycle_at(i)->add_service_record();
    }

    for (short i = 0; i < my_fleet->get_SUV_size(); ++i)
    {
        my_fleet->get_SUV_at(i)->add_service_record();
    }
}
```

Call service fleet command, then check the service history of the motorcycle with ID M1, we can see the service history has been done and recorded.



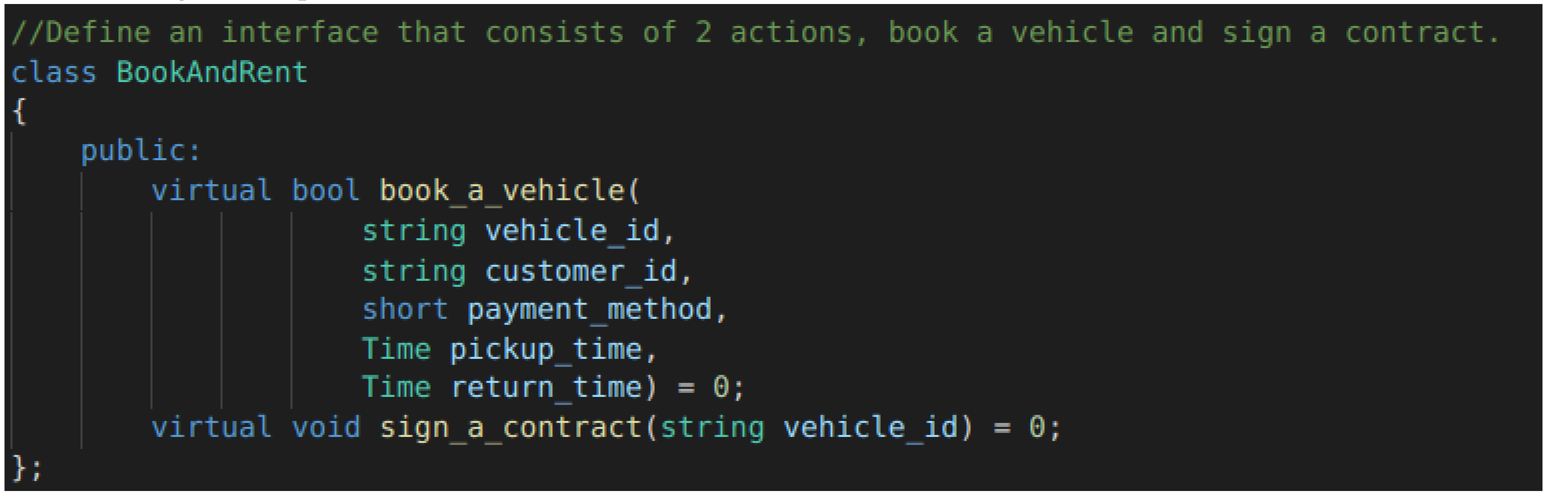
```
----------
SERVICE FLEET
----------

Service successfully!

1. Return back
Please choose an option:
```



```
----------
SERVICE HISTORY
----------

No   ID      Time              Mileage             Engine        Transmission              Tires
0    0        18h:5/8/2021              2000               minor               minor          adjustment


1. Return back
Please choose an option:
```

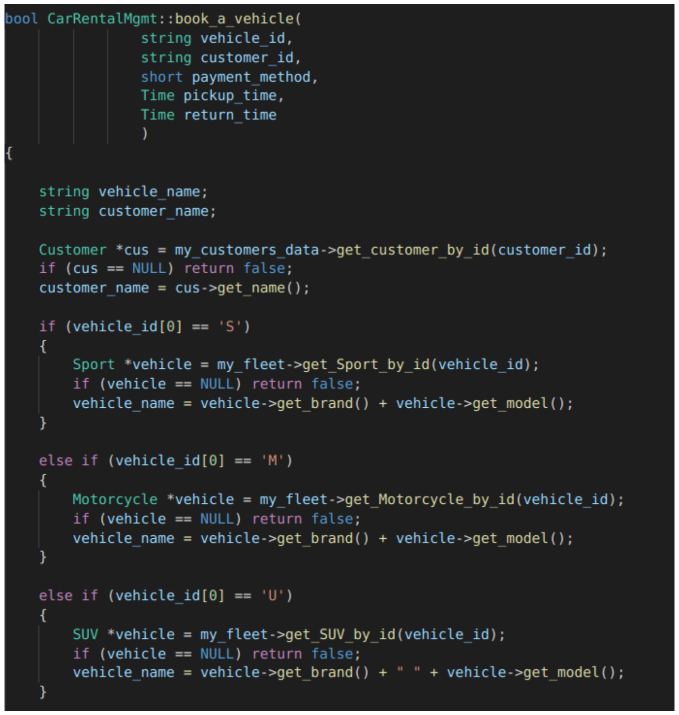
Re-define your interfaces / abstract classes

CarRentalMngt will implement the interface BookAndRent.

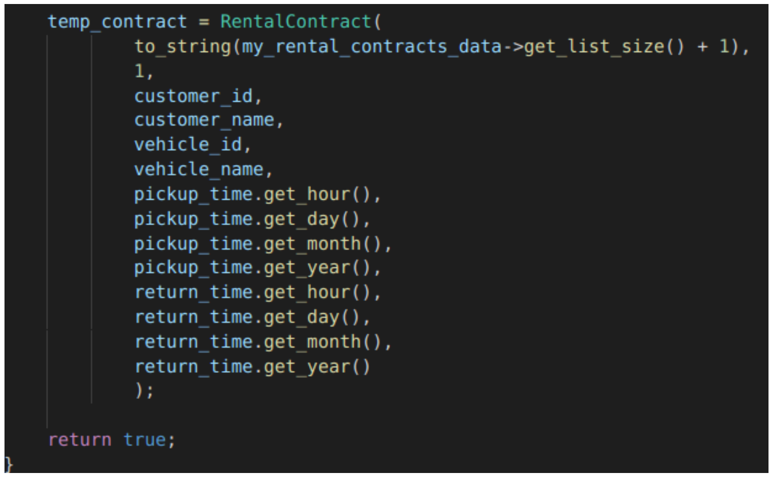


```cpp
//Define an interface that consists of 2 actions, book a vehicle and sign a contract.
class BookAndRent
{
    public:
        virtual bool book_a_vehicle(
                    string vehicle_id,
                    string customer_id,
                    short payment_method,
                    Time pickup_time,
                    Time return_time) = 0;
        virtual void sign_a_contract(string vehicle_id) = 0;
};
```

Book a vehicle will query the car fleet to get the ID and name of the vehicle, ID and name of the customer from CustomersData, set pick up time and return time and “fill in” the temporary contract.

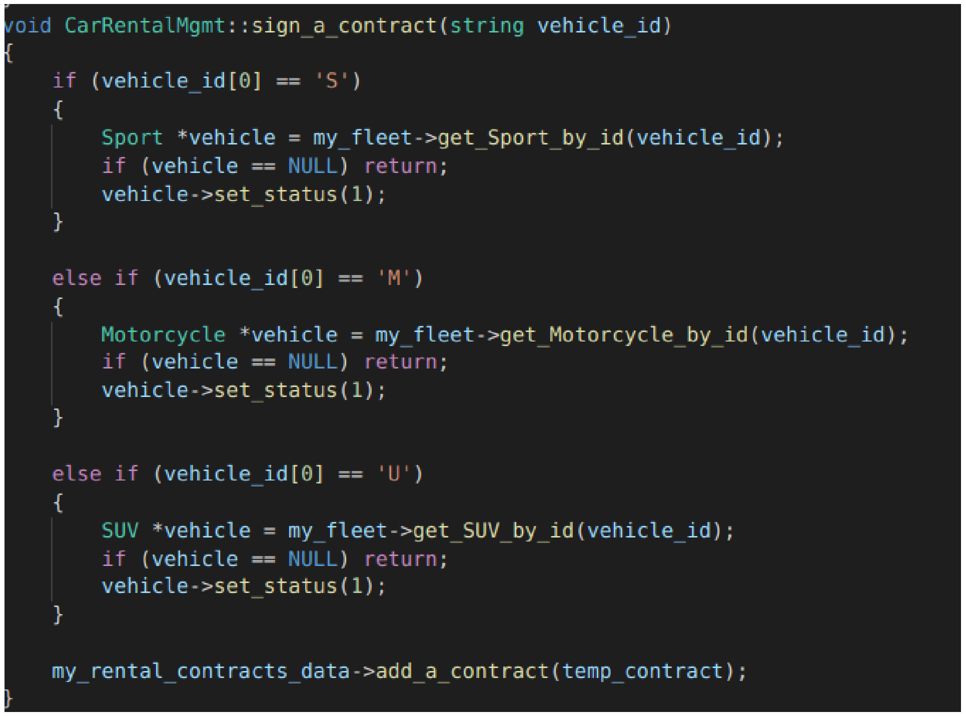


```cpp
bool CarRentalMgmt::book_a_vehicle(
                string vehicle_id,
                string customer_id,
                short payment_method,
                Time pickup_time,
                Time return_time
                )
{

    string vehicle_name;
    string customer_name;

    Customer *cus = my_customers_data->get_customer_by_id(customer_id);
    if (cus == NULL) return false;
    customer_name = cus->get_name();

    if (vehicle_id[0] == 'S')
    {
        Sport *vehicle = my_fleet->get_Sport_by_id(vehicle_id);
        if (vehicle == NULL) return false;
        vehicle_name = vehicle->get_brand() + vehicle->get_model();
    }

    else if (vehicle_id[0] == 'M')
    {
        Motorcycle *vehicle = my_fleet->get_Motorcycle_by_id(vehicle_id);
        if (vehicle == NULL) return false;
        vehicle_name = vehicle->get_brand() + vehicle->get_model();
    }

    else if (vehicle_id[0] == 'U')
    {
        SUV *vehicle = my_fleet->get_SUV_by_id(vehicle_id);
        if (vehicle == NULL) return false;
        vehicle_name = vehicle->get_brand() + " " + vehicle->get_model();
    }
```



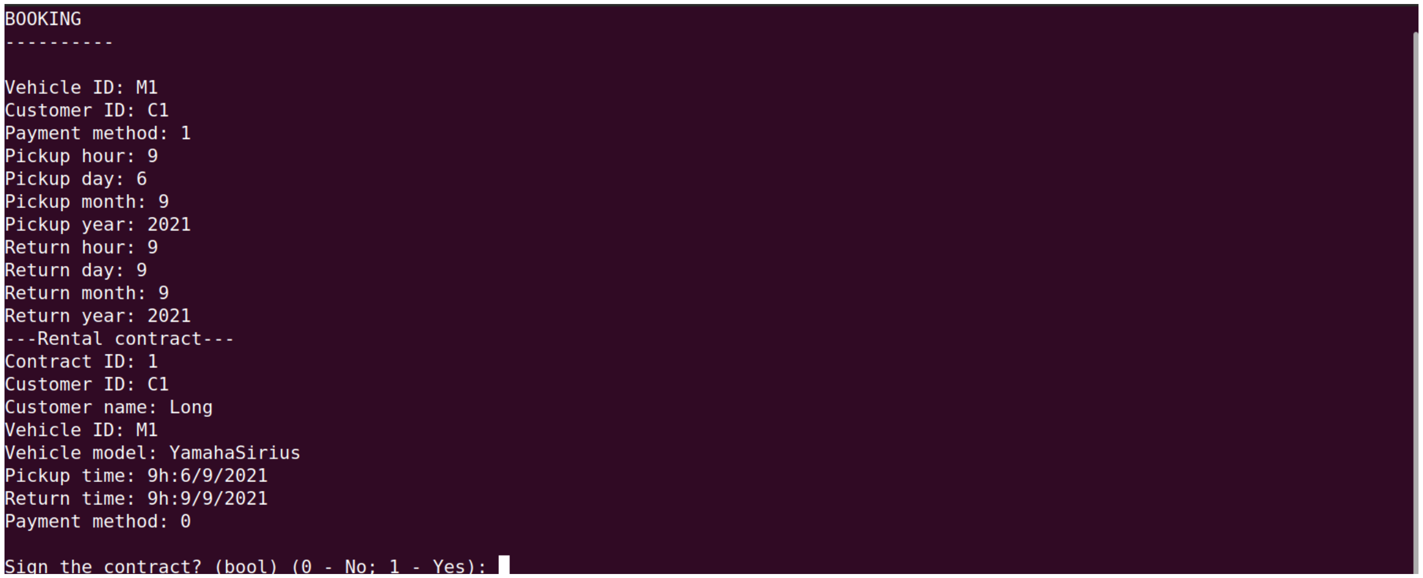
```cpp
    temp_contract = RentalContract(
            to_string(my_rental_contracts_data->get_list_size() + 1),
            1,
            customer_id,
            customer_name,
            vehicle_id,
            vehicle_name,
            pickup_time.get_hour(),
            pickup_time.get_day(),
            pickup_time.get_month(),
            pickup_time.get_year(),
            return_time.get_hour(),
            return_time.get_day(),
            return_time.get_month(),
            return_time.get_year()
            );

    return true;
}
```

If the customer sign the contract, the temporary contract will be saved to contracts data, and that vehicle status is set to unavailable.

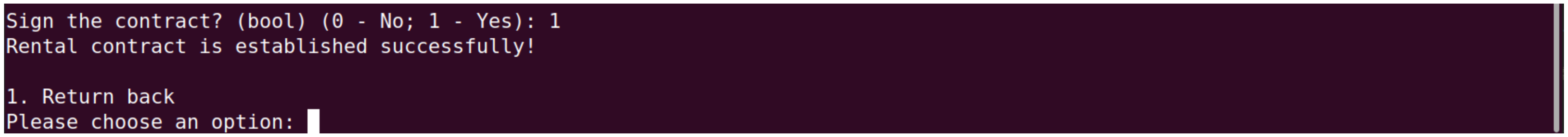


```cpp
void CarRentalMgmt::sign_a_contract(string vehicle_id)
{
    if (vehicle_id[0] == 'S')
    {
        Sport *vehicle = my_fleet->get_Sport_by_id(vehicle_id);
        if (vehicle == NULL) return;
        vehicle->set_status(1);
    }

    else if (vehicle_id[0] == 'M')
    {
        Motorcycle *vehicle = my_fleet->get_Motorcycle_by_id(vehicle_id);
        if (vehicle == NULL) return;
        vehicle->set_status(1);
    }

    else if (vehicle_id[0] == 'U')
    {
        SUV *vehicle = my_fleet->get_SUV_by_id(vehicle_id);
        if (vehicle == NULL) return;
        vehicle->set_status(1);
    }

    my_rental_contracts_data->add_a_contract(temp_contract);
}
```

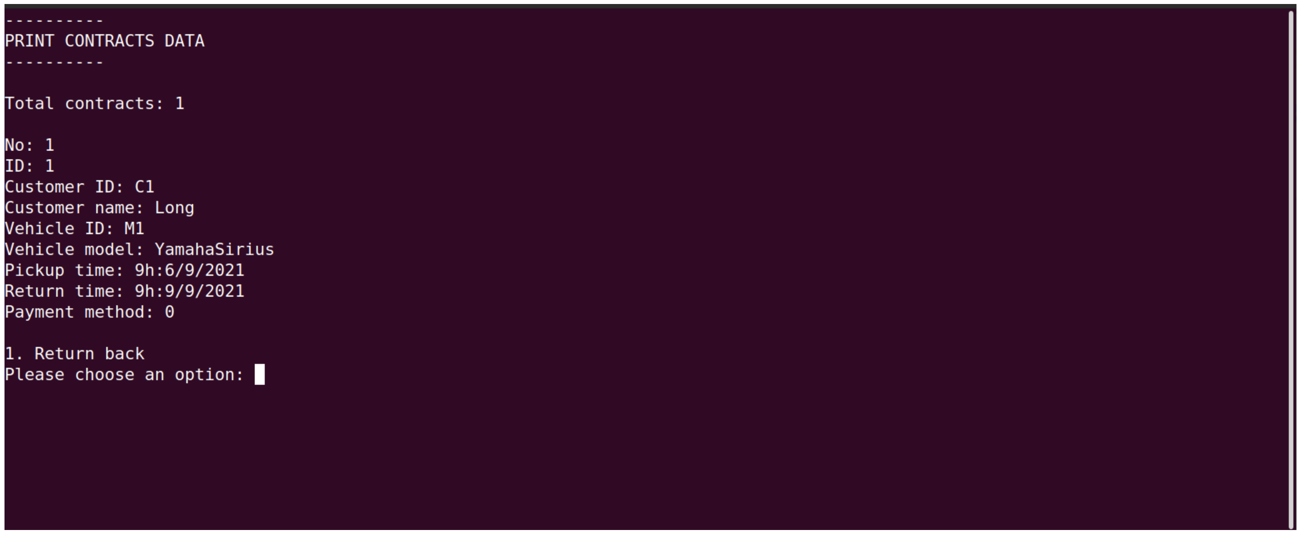
Add a customer name Long with ID C1, then book for him the motorcycle M1. We can see on the terminal that it's added successfully.



```
BOOKING
----------

Vehicle ID: M1
Customer ID: C1
Payment method: 1
Pickup hour: 9
Pickup day: 6
Pickup month: 9
Pickup year: 2021
Return hour: 9
Return day: 9
Return month: 9
Return year: 2021
---Rental contract---
Contract ID: 1
Customer ID: C1
Customer name: Long
Vehicle ID: M1
Vehicle model: YamahaSirius
Pickup time: 9h:6/9/2021
Return time: 9h:9/9/2021
Payment method: 0

Sign the contract? (bool) (0 - No; 1 - Yes):
```



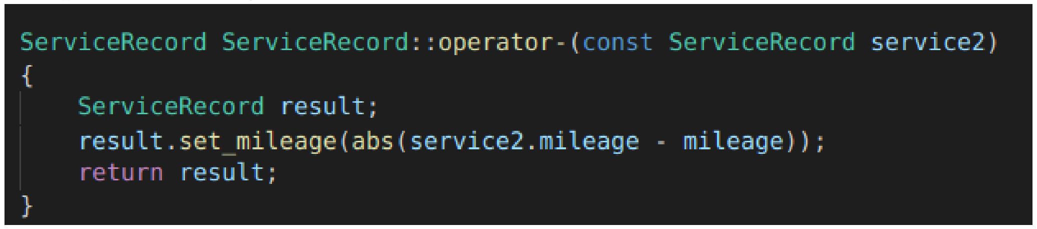
```
Sign the contract? (bool) (0 - No; 1 - Yes): 1
Rental contract is established successfully!

1. Return back
Please choose an option:
```



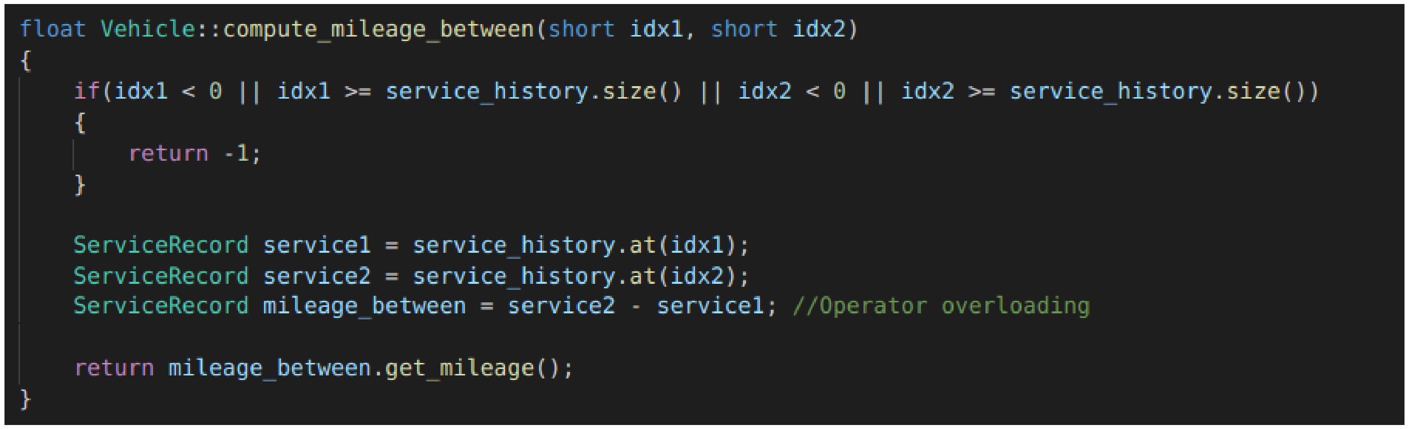
```
---------
PRINT CONTRACTS DATA
---------

Total contracts: 1

No: 1
ID: 1
Customer ID: C1
Customer name: Long
Vehicle ID: M1
Vehicle model: YamahaSirius
Pickup time: 9h:6/9/2021
Return time: 9h:9/9/2021
Payment method: 0

1. Return back
Please choose an option:
```

Operator Overloading

Compute the mileage difference between 2 services.

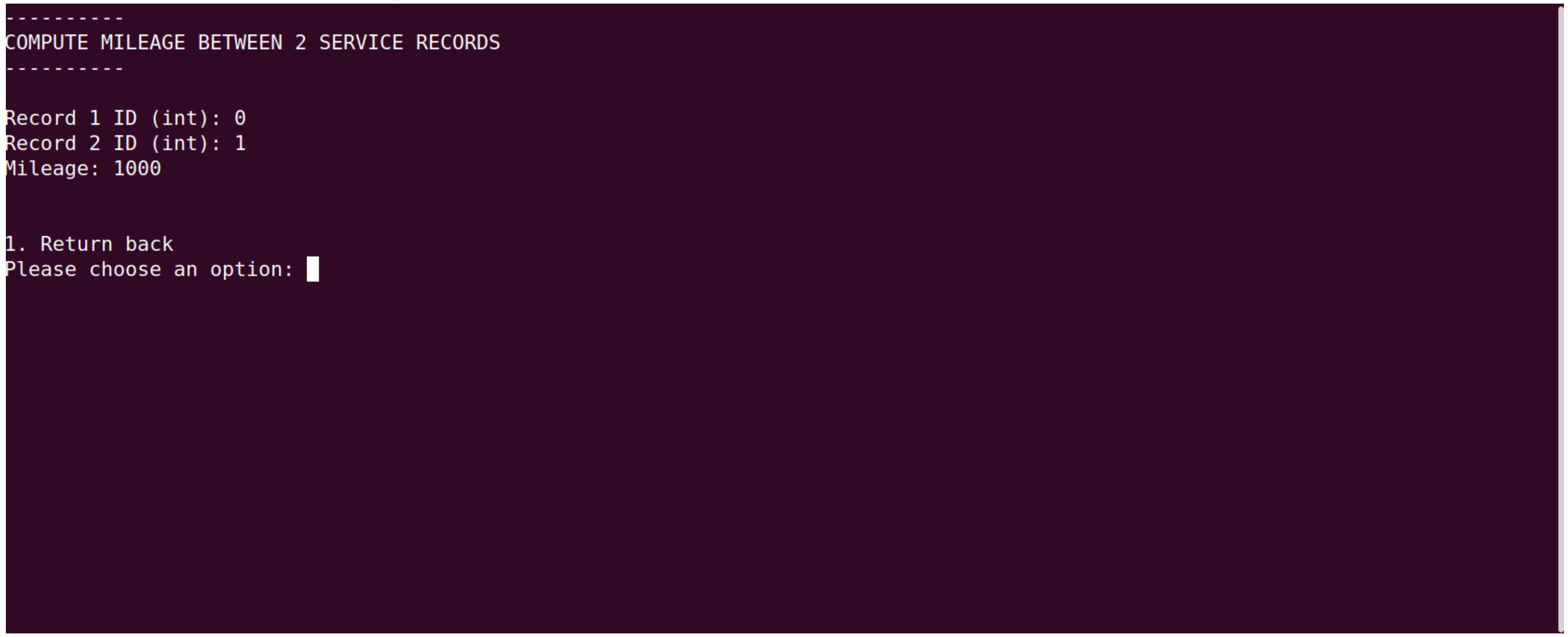


```cpp
ServiceRecord ServiceRecord::operator-(const ServiceRecord service2)
{
    ServiceRecord result;
    result.set_mileage(abs(service2.mileage - mileage));
    return result;
}
```



```cpp
float Vehicle::compute_mileage_between(short idx1, short idx2)
{
    if(idx1 < 0 || idx1 >= service_history.size() || idx2 < 0 || idx2 >= service_history.size())
    {
        return -1;
    }

    ServiceRecord service1 = service_history.at(idx1);
    ServiceRecord service2 = service_history.at(idx2);
    ServiceRecord mileage_between = service2 - service1; //Operator overloading

    return mileage_between.get_mileage();
}
```

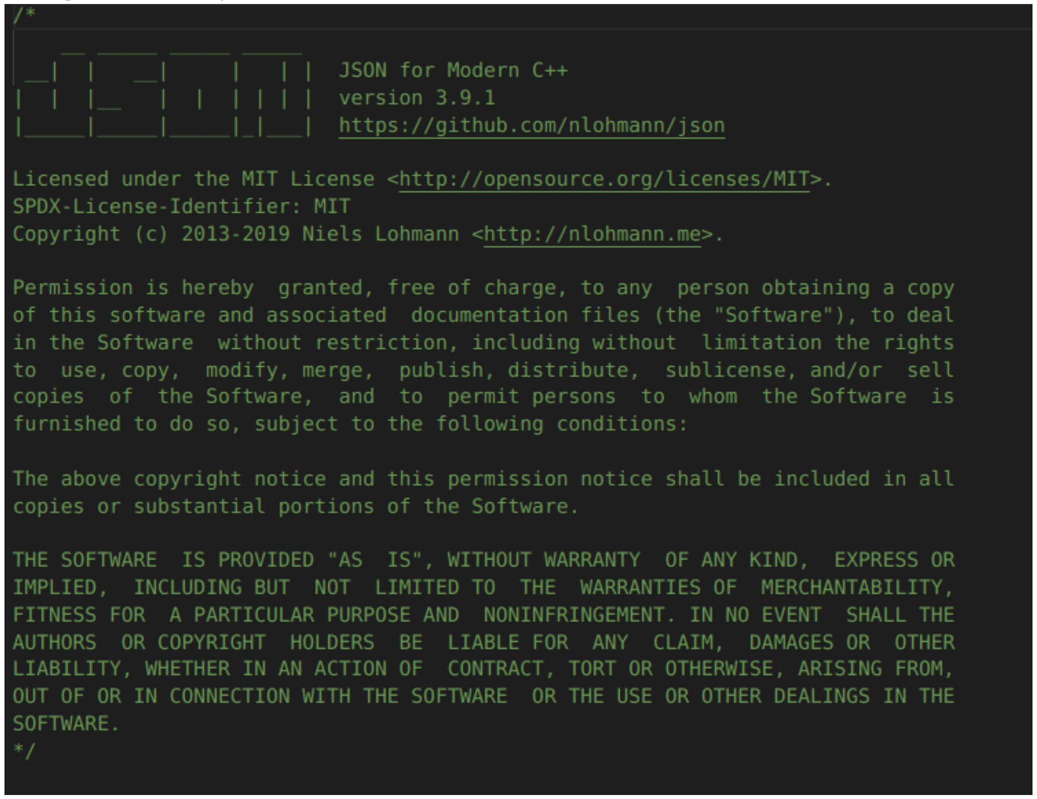
Assume that Long has returned the M1 vehicle and the mileage is added 1000 more, which is 3000 now, then we do another service and compute the difference mileage between the 2 services. The terminal return 1000.



```
----------
COMPUTE MILEAGE BETWEEN 2 SERVICE RECORDS
----------

Record 1 ID (int): 0
Record 2 ID (int): 1
Mileage: 1000


1. Return back
Please choose an option: 
```

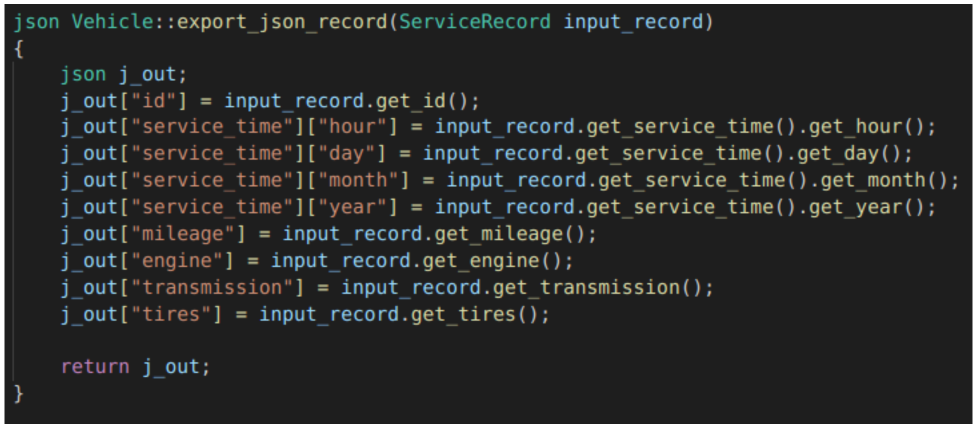
Serializing the service history of a rental car

Using Json library from Nlohmann.

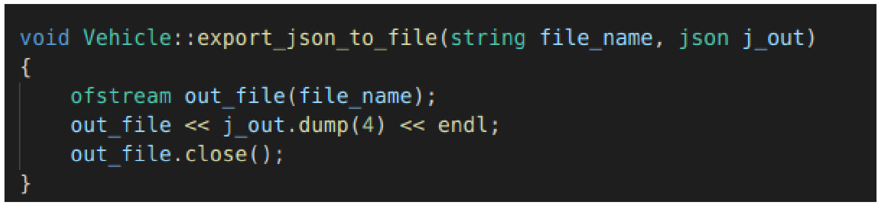


```
/*
    __ _____ _____ _____
 __|  |   __|     |   | |  JSON for Modern C++
|  |  |__   |  |  | | | |  version 3.9.1
|_____|_____|_____|_|___|  https://github.com/nlohmann/json

Licensed under the MIT License <http://opensource.org/licenses/MIT>.
SPDX-License-Identifier: MIT
Copyright (c) 2013-2019 Niels Lohmann <http://nlohmann.me>.

Permission is hereby  granted, free of charge, to any  person obtaining a copy
of this software and associated  documentation files (the "Software"), to deal
in the Software  without restriction, including without  limitation the rights
to  use, copy,  modify, merge,  publish, distribute,  sublicense, and/or  sell
copies  of  the Software,  and  to  permit persons  to  whom  the Software  is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE  IS PROVIDED "AS  IS", WITHOUT WARRANTY  OF ANY KIND,  EXPRESS OR
IMPLIED,  INCLUDING BUT  NOT  LIMITED TO  THE  WARRANTIES OF  MERCHANTABILITY,
FITNESS FOR  A PARTICULAR PURPOSE AND  NONINFRINGEMENT. IN NO EVENT  SHALL THE
AUTHORS  OR COPYRIGHT  HOLDERS  BE  LIABLE FOR  ANY  CLAIM,  DAMAGES OR  OTHER
LIABILITY, WHETHER IN AN ACTION OF  CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE  OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
*/
```

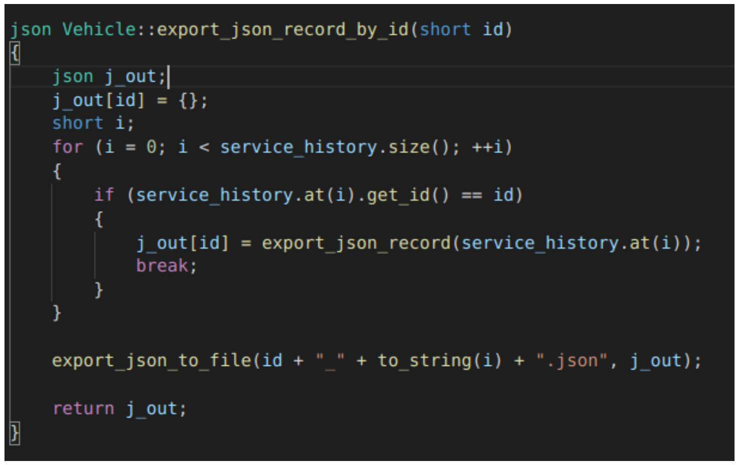
Support 2 options, export Json of a specific service record by ID, or export all the service history to .json file.



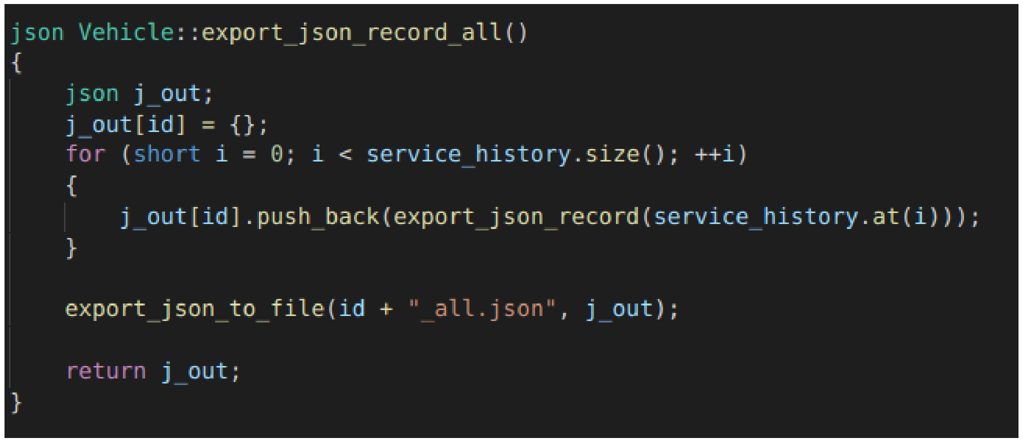
```cpp
json Vehicle::export_json_record(ServiceRecord input_record)
{
    json j_out;
    j_out["id"] = input_record.get_id();
    j_out["service_time"]["hour"] = input_record.get_service_time().get_hour();
    j_out["service_time"]["day"] = input_record.get_service_time().get_day();
    j_out["service_time"]["month"] = input_record.get_service_time().get_month();
    j_out["service_time"]["year"] = input_record.get_service_time().get_year();
    j_out["mileage"] = input_record.get_mileage();
    j_out["engine"] = input_record.get_engine();
    j_out["transmission"] = input_record.get_transmission();
    j_out["tires"] = input_record.get_tires();

    return j_out;
}
```



```cpp
void Vehicle::export_json_to_file(string file_name, json j_out)
{
    ofstream out_file(file_name);
    out_file << j_out.dump(4) << endl;
    out_file.close();
}
```

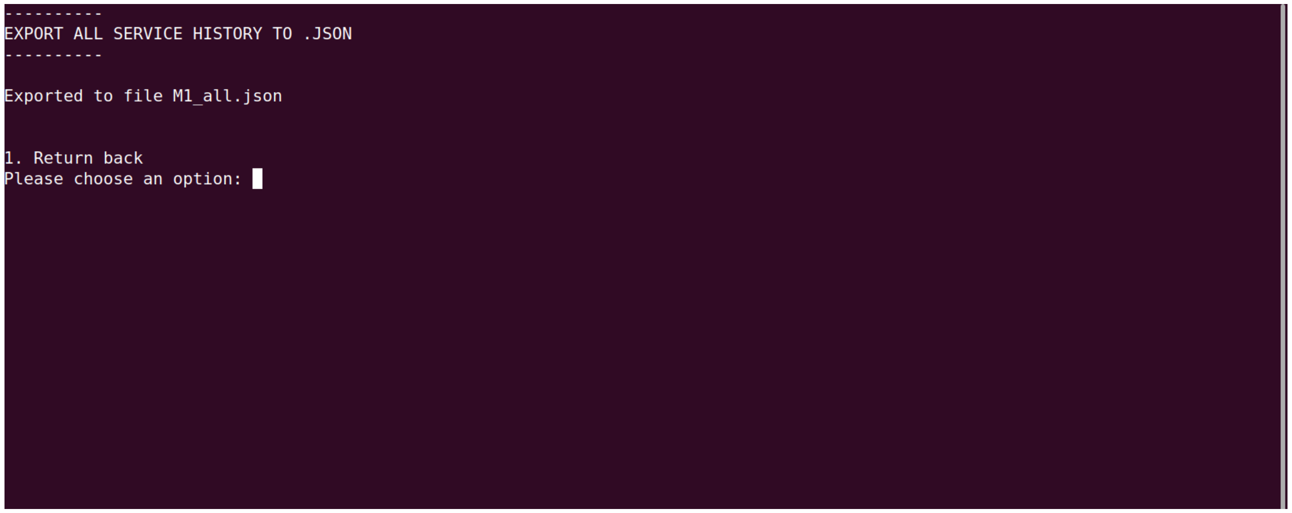


```cpp
json Vehicle::export_json_record_by_id(short id)
{
    json j_out;
    j_out[id] = {};
    short i;
    for (i = 0; i < service_history.size(); ++i)
    {
        if (service_history.at(i).get_id() == id)
        {
            j_out[id] = export_json_record(service_history.at(i));
            break;
        }
    }

    export_json_to_file(id + "_" + to_string(i) + ".json", j_out);

    return j_out;
}
```

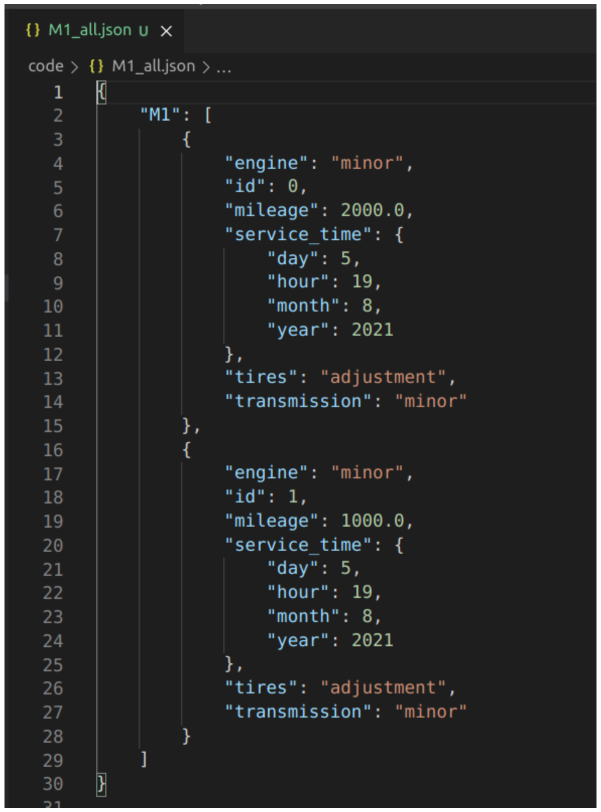


```cpp
json Vehicle::export_json_record_all()
{
    json j_out;
    j_out[id] = {};
    for (short i = 0; i < service_history.size(); ++i)
    {
        j_out[id].push_back(export_json_record(service_history.at(i)));
    }

    export_json_to_file(id + "_all.json", j_out);

    return j_out;
}
```

I choose the option export all, the terminal return exported successfully, then we can check with the M1_all.json storing the 2 records.



```
----------
EXPORT ALL SERVICE HISTORY TO .JSON
----------

Exported to file M1_all.json


1. Return back
Please choose an option: 
```



```
{
    "M1": [
        {
            "engine": "minor",
            "id": 0,
            "mileage": 2000.0,
            "service_time": {
                "day": 5,
                "hour": 19,
                "month": 8,
                "year": 2021
            },
            "tires": "adjustment",
            "transmission": "minor"
        },
        {
            "engine": "minor",
            "id": 1,
            "mileage": 1000.0,
            "service_time": {
                "day": 5,
                "hour": 19,
                "month": 8,
                "year": 2021
            },
            "tires": "adjustment",
            "transmission": "minor"
        }
    ]
}
```