Ques 1: (practice before mid 2)

```cpp
#ifndef PARKINGSYSTEM_H
#define PARKINGSYSTEM_H

#include <iostream>
#include <string>
using namespace std;

class Vehicle {
public:
    Vehicle(const string& registration, const string& manufacturer, const string&
model)
        : registrationNumber(registration), manufacturer(manufacturer),
model(model) {}

    virtual ~Vehicle() {} // Virtual destructor for polymorphic behavior

    virtual void display() const {
        cout << "Registration Number: " << registrationNumber << ", Manufacturer:
" << manufacturer
            << ", Model: " << model << endl;
    }

protected:
    string registrationNumber;
    string manufacturer;
    string model;
};

class Car : public Vehicle {
public:
    Car(const string& registration, const string& manufacturer, const string&
model, int doors)
        : Vehicle(registration, manufacturer, model), numDoors(doors) {}

    void display() const override {
        Vehicle::display();
        cout << ", Number of Doors: " << numDoors << endl;
    }

private:
    int numDoors;
};
```

```
class Bike : public Vehicle {
public:
    Bike(const string& registration, const string& manufacturer, const string&
model)
        : Vehicle(registration, manufacturer, model) {}
};

class ParkingLot {
public:
    ParkingLot(int size);
    ~ParkingLot();

    bool park(Vehicle* vehicle);
    Vehicle** getParkedVehicles();
    int getNum_of_slots() const;

private:
    int numof_slots;
    Vehicle** parkedVehicles;
};

#endif // PARKINGSYSTEM_H
```

CPP

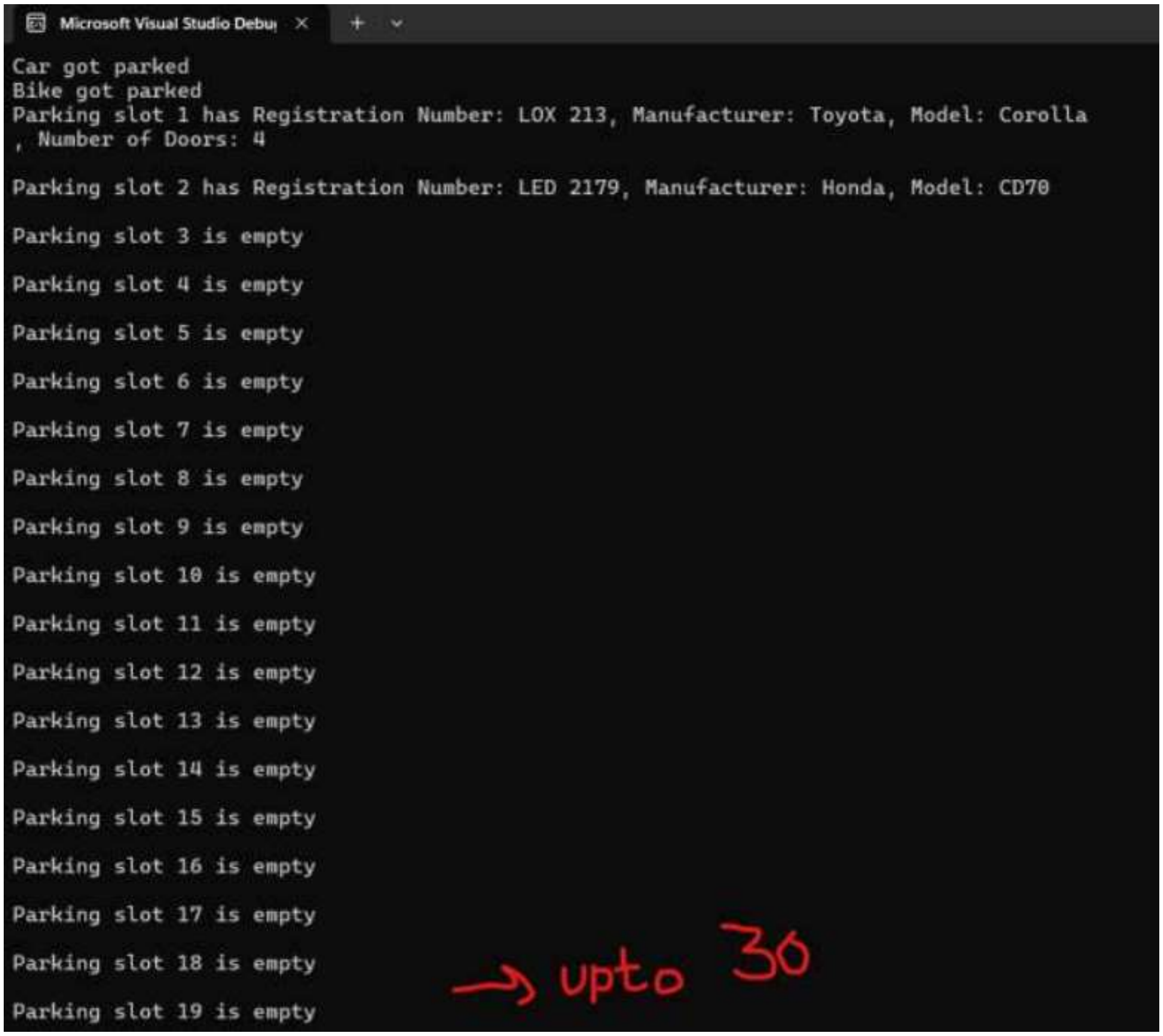
```
#include "ParkingSystem.h"

ParkingLot::ParkingLot(int size) : numof_slots(size) {
    parkedVehicles = new Vehicle * [numof_slots];
    for (int i = 0; i < numof_slots; ++i) {
        parkedVehicles[i] = nullptr;
    }
}

ParkingLot::~ParkingLot() {
    for (int i = 0; i < numof_slots; ++i) {
        delete parkedVehicles[i];
    }
    delete[] parkedVehicles;
}
```

```cpp
bool ParkingLot::park(Vehicle* vehicle) {
    for (int i = 0; i < numof_slots; ++i) {
        if (!parkedVehicles[i]) {
            parkedVehicles[i] = vehicle;
            return true;
        }
    }
    return false; // Parking lot is full
}

Vehicle** ParkingLot::getParkedVehicles() {
    return parkedVehicles;
}

int ParkingLot::getNum_of_slots() const {
    return numof_slots;
}
```

Driver

```cpp
#include "ParkingSystem.h"
using namespace std;

int main() {
    ParkingLot* fastParking = new ParkingLot(30);
    Car* corolla1 = new Car("LOX 213", "Toyota", "Corolla", 4);
    Bike* honda1 = new Bike("LED 2179", "Honda", "CD70");

    if (fastParking->park(corolla1))
        cout << "Car got parked" << endl;
    if (fastParking->park(honda1))
        cout << "Bike got parked" << endl;

    int num = fastParking->getNum_of_slots();
    Vehicle** vehicles = fastParking->getParkedVehicles();

    for (int i = 0; i < num; i++) {
        if (vehicles[i] != nullptr)
            cout << "Parking slot " << i + 1 << " has ";
        else
            cout << "Parking slot " << i + 1 << " is empty" << endl;
        if (vehicles[i] != nullptr)
            vehicles[i]->display();

        cout << endl;
    }

    delete fastParking;
    delete corolla1;
    delete honda1;

    return 0;
}
```

Output



```
Car got parked
Bike got parked
Parking slot 1 has Registration Number: LOX 213, Manufacturer: Toyota, Model: Corolla
, Number of Doors: 4

Parking slot 2 has Registration Number: LED 2179, Manufacturer: Honda, Model: CD70

Parking slot 3 is empty

Parking slot 4 is empty

Parking slot 5 is empty

Parking slot 6 is empty

Parking slot 7 is empty

Parking slot 8 is empty

Parking slot 9 is empty

Parking slot 10 is empty

Parking slot 11 is empty

Parking slot 12 is empty

Parking slot 13 is empty

Parking slot 14 is empty

Parking slot 15 is empty

Parking slot 16 is empty

Parking slot 17 is empty

Parking slot 18 is empty

Parking slot 19 is empty
```

$\rightarrow$ upto 30