➢ Ques 3

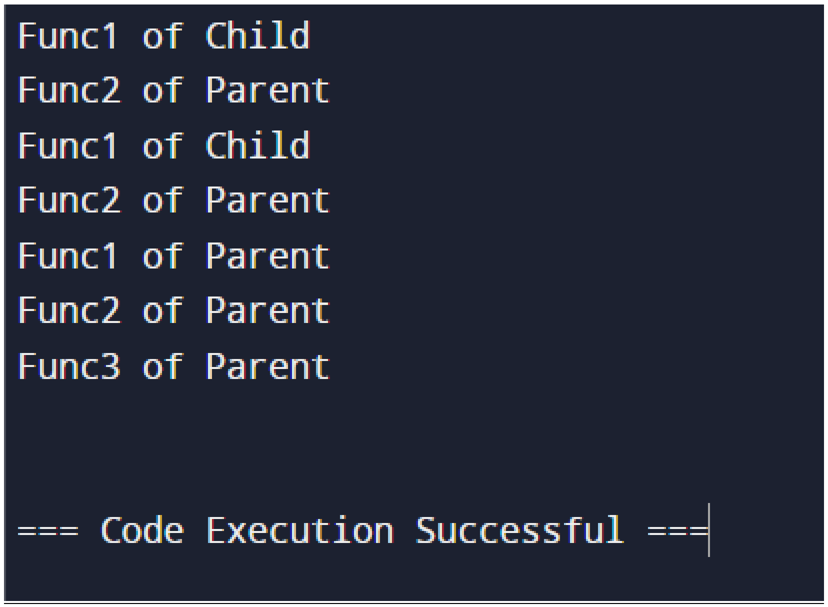
```cpp
#include <iostream>

class Department {
    int deptNumber;
    std::string name;

public:
    void setDepartment(int dep, std::string name) {
        deptNumber = dep;
        this->name = name;
    }
};

class Employee {
    int id;
    Department d;

public:
    void setEmployee(int dept, std::string depName, int empId) {
        id = empId;
        d.setDepartment(dept, depName);
    }

    int getID() const {
        return id;
    }
};

int main() {
    Employee employees[10];

    for (int i = 0; i < 10; ++i) {
        employees[i].setEmployee(1, "Administration", i + 1);
    }

    // Display employee details
    for (int i = 0; i < 10; ++i) {
        std::cout << "Employee ID: " << employees[i].getID() << std::endl;
    }
    return 0;
}
```

Ques 2

```
#include <iostream>

class Parent {
public:
    virtual void Func1() {
        std::cout << "Func1 of Parent" << std::endl;
    }

    void Func2() {
        std::cout << "Func2 of Parent" << std::endl;
    }

    void Func3() {
        std::cout << "Func3 of Parent" << std::endl;
    }

    void Func4() {
        Func1();
        Func2();
    }
};

class Child : public Parent {
public:
    void Func1() {
        std::cout << "Func1 of Child" << std::endl;
    }

    void Func2() {
        std::cout << "Func2 of Child" << std::endl;
    }

    Virtual void Func3() {
        std::cout << "Func3 of Child" << std::endl;
    }
```

```cpp
};

int main() {
    Parent* m1 = new Child;
    m1->Func1();
    m1->Func2();
    m1->Func4();

    Parent m2 = *(new Child);
    m2.Func1();
    m2.Func2();
    m2.Func3();

    delete m1; // Freeing the memory allocated by new
    return 0;
}
```

Output

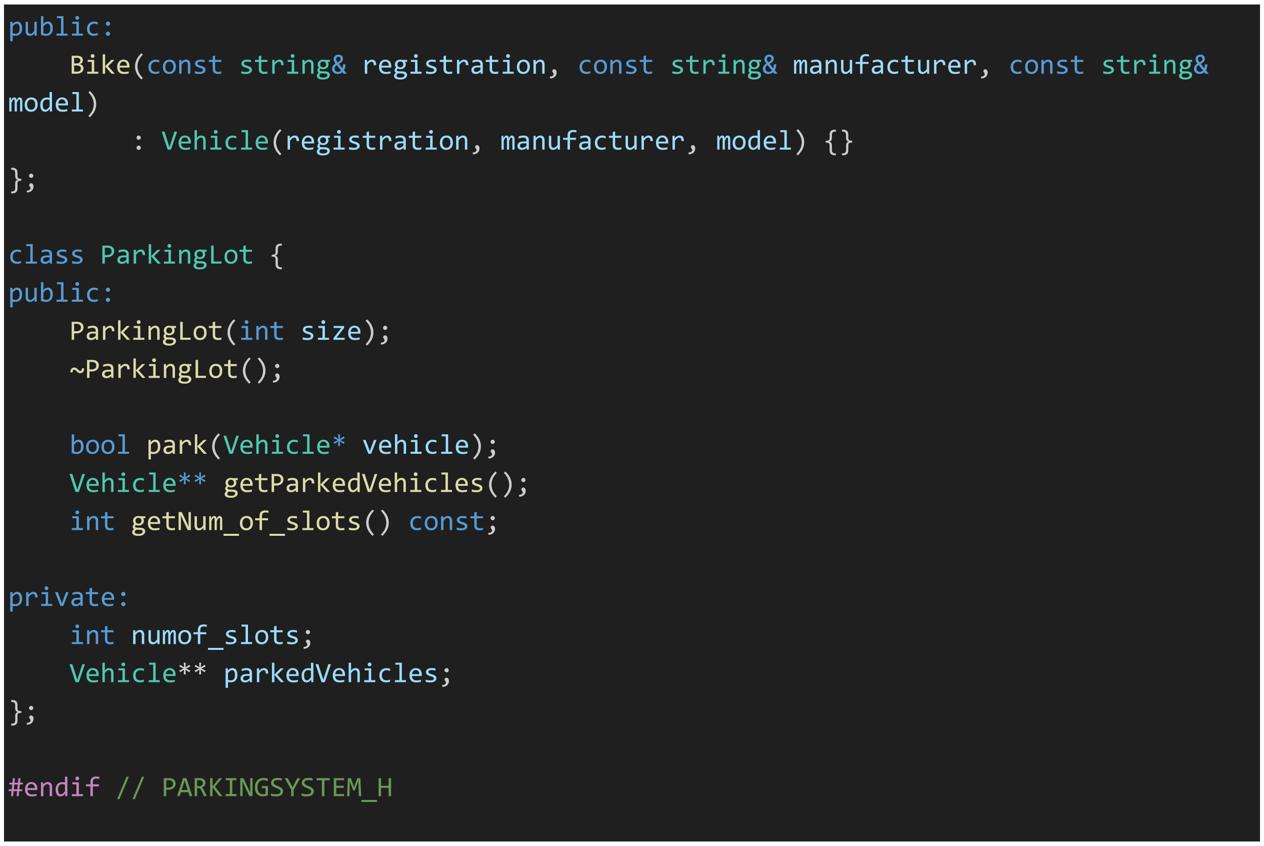


```
Func1 of Child
Func2 of Parent
Func1 of Child
Func2 of Parent
Func1 of Parent
Func2 of Parent
Func3 of Parent


=== Code Execution Successful ===
```

- **Ques 1**

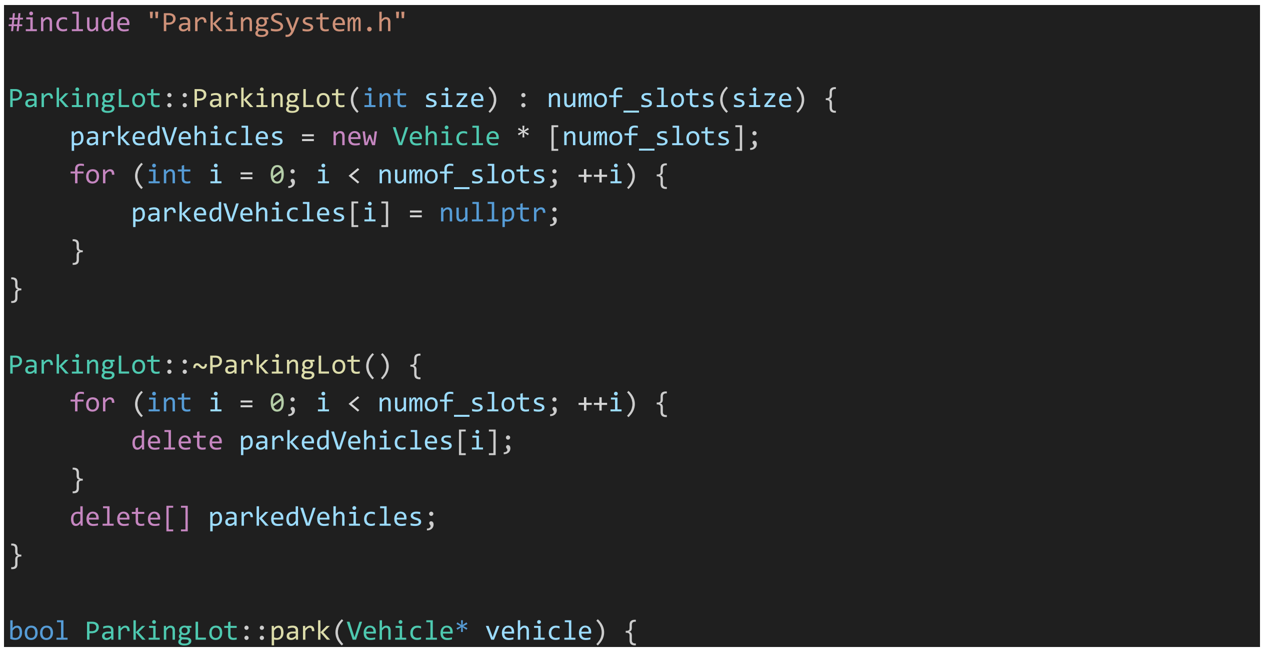
  Header File

```
#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



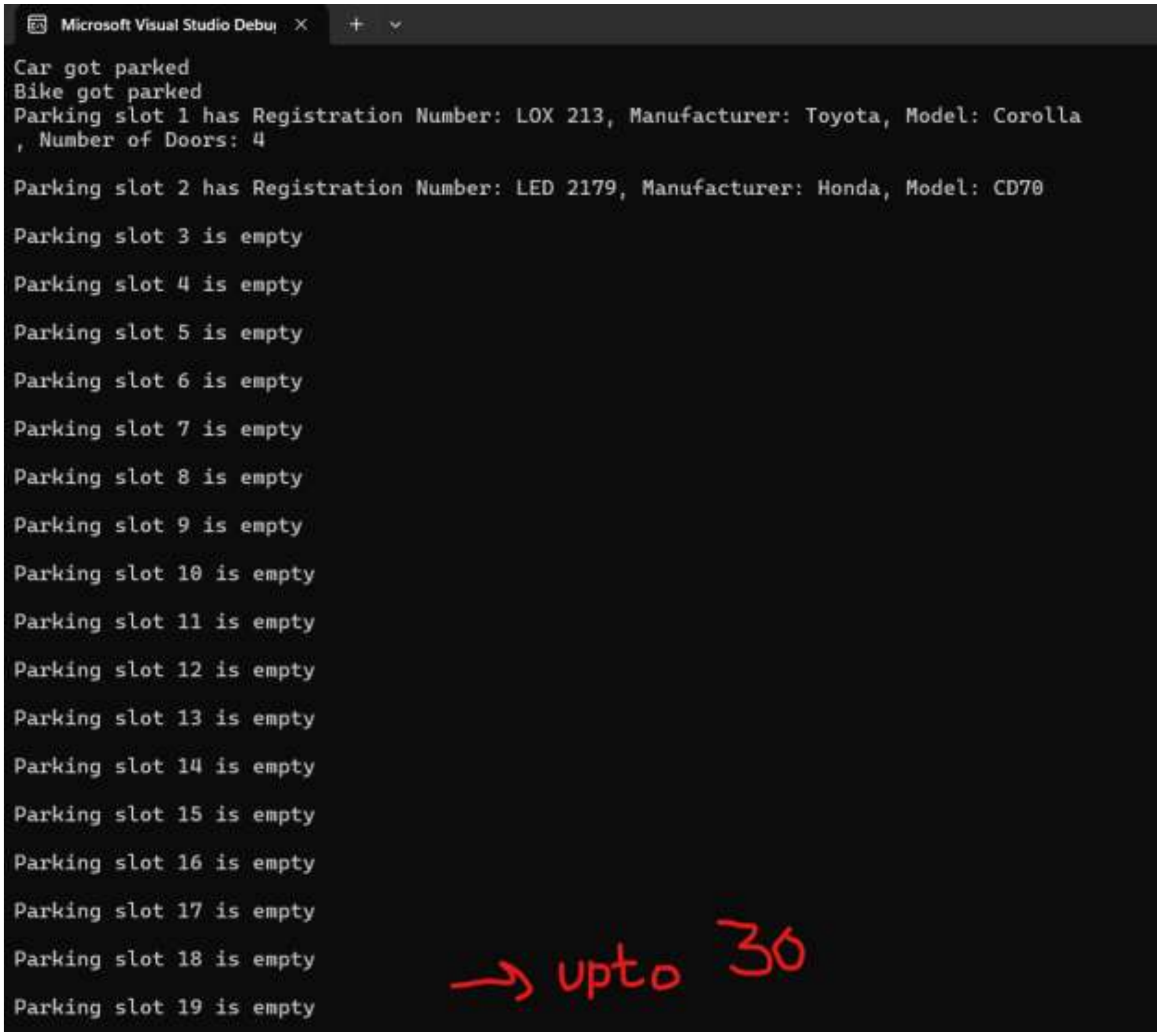
```
#include "ParkingSystem.h"

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;
}

bool ParkingLot::park(Vehicle* vehicle) {
```

```cpp
    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"

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;
}

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;
}
```

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
```