石 家 庄 铁 道 大 学

**实 验 报 告**

**实习二 栈和队列应用**

题 目： 停车场管理

班 级： 信1901 –4

姓 名： 闫竞存

学 号：20194127

日 期：2020.10.29

**1. 实验题目**

仅仅认识到栈和队列是两种特殊的线性表是远远不够的，本次实习的目的在于使读者深入了解栈和队列的特征，以便在实际问题背景下灵活运用它们，同时还将巩固这两种结构的构造方法。

[问题描述]

设停车场内只有一个的停放 n 辆汽车的狭长通道，且只有一个大门可供汽车进出。汽车在停车场内按车辆到达时间的先后顺序，依次由北向南排列（大门在最南端，最先到达的第一辆车停放在车场的最北端），若车场内已停满 n 辆汽车，则后来的汽车只能在门外的便道上等候，一旦有车开走，则排在便道上的第一辆车即可开入；当停车场内某辆车要离开时，在它之后开入的车辆必须先退出车场为它让路，待该辆车开出大门外，其它车辆再按原次序进入车场，每辆停放在车场的车在它离开停车场时必须按它停留的时间长短交纳费用。试为停车场编制按上述要求进行管理的模拟程序。

[基本要求]

以栈模拟停车场，以队列模拟车场外的便道，按照从终端读入的输入数据序列进行模拟管理。每一组输入数据包括三个数据项：汽车“到达”或“离去”信息、汽车牌照号码及到或离去的时刻，对每一组输入数据进行操作后的输出数据为：若是车辆到达，则输出汽车在停车场内或便道上的停车位置；若是车离去；则输出汽车在停车场内停留的时间和应交纳的的费用（在便道上停留时间不收费）。栈以顺序结构实现，队列以链表实现。

**2. 需求分析**

存储在停车场的栈和从中转栈，两个栈共享空间。

汽车可有不同种类，则它们的占地面积不同，收费标准也不同，如 1 辆客车和1.5 辆小汽车的占地面积相同，1 辆十轮卡车占地面积相当于 3 辆小汽车的占地面积。

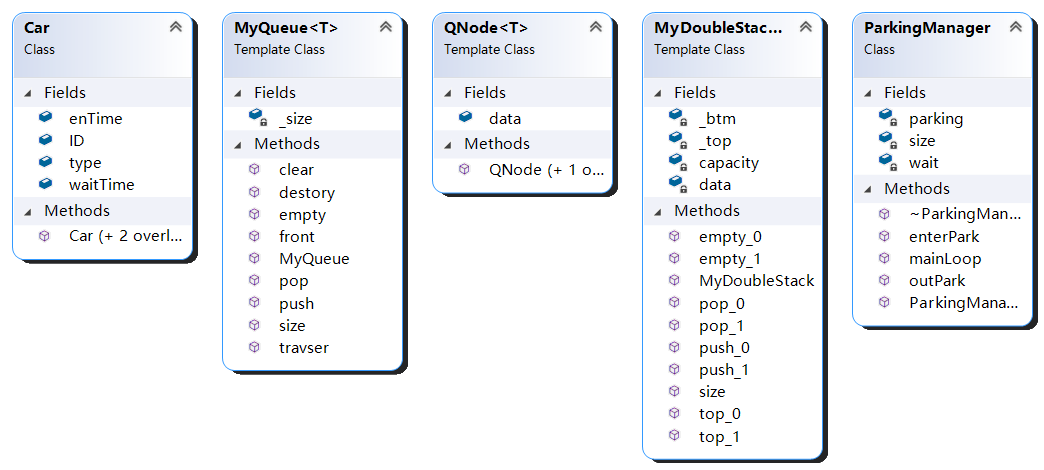
汽车可以直接从便道上开走，此时派在它前面的汽车要先开走让路，然后再依次排到队尾。

停放在便道上的汽车也收费，收费标准比停放在停车场的车低

每一个位置的收费情况都是根据车辆的类型和车辆停在其位置上的时间决定的。

**3. 概要设计**

程序的类图如下：



|  |  |  |
| --- | --- | --- |
| 类名 | 说明 | 备注 |
| Car | 车 | 普通类 |
| QNode<T> | 队列的节点 | 模板类 |
| MyQueue<T> | 队列（底层以链表实现） | 模板类 |
| MyDoubleStack<T> | 栈（双栈） | 模板类 |
| ParkingManager | 车站管理系统 | 普通类 |

**4. 关键算法的详细设计**

设定：

|  |  |  |  |
| --- | --- | --- | --- |
| 车辆类型 | 类型代号 | 占地面积 | 收费系数 |
| 小轿车 | 1 | 1 | 1 |
| 客车 | 2 | 1.5 | 1.5 |
| 十轮卡车 | 3 | 3 | 3 |

停车场使用规则：

* 车场不存在“对齐”情况，即只要有位置，就可以停车，不管容量剩余是不是整数。
* 每当有车从车库中出来，在等待队列中的第一辆车会根据停车场中剩余位置的情况来选择继续等候还是进入停车场。

收费规则：

* 收费金额 = 收费系数 \* [在停车场中单位时间的收费金额 \* （当前时间 – 进入停车场的时间） + 在等候队列中单位时间的收费金额 \* （进入停车场的时间- 进入等候队列的时间） ]

**部分源码：**

停车：

// 停车函数

void ParkingManager::enterPark() {

    string id;

    int time;

    int type;

    int enterSuc = 0;

    do {

        cin >> id >> type >> time;

        if (type < 1 || type > 3 || time < 0) {

            cout << "输入数据有误，请重新输入" << endl;

            enterSuc = 1;

        } else break;

    } while (enterSuc);

    if (this->size - OCCUPY[type] < 0) { // 检查车库有没有满

        cout << "车已停满，车辆进入等候区等候..." << endl;

        this->wait.push(new Car(time, id, type));

        return;

    }

    this->parking.push\_0(new Car(id, type, time));

    cout << "车号 " << id << " 进入 " << this->parking.size() << " 号车位, 时间为" << time << endl;

    this->size -= OCCUPY[type];

}

出停车场:

/\*

    考虑三种情况:

    1. 一直在停车场

    2. 原来在便道等候，后来进入了停车场

    3. 一直在便道等候

\*/

void ParkingManager::outPark() {

    string id;

    int time;

    int type;

    cin >> id >> type >> time;

    int flag = 0;

    while (!this->parking.empty\_0()) {

        if (this->parking.top\_0()->ID != id) {

            //这辆车不是要出去的车

            this->parking.push\_1(this->parking.top\_0());

            this->parking.pop\_0();

        } else {

            // 这辆车是要出去的车

            cout << "车 " << this->parking.top\_0()->ID

                << " 出库，时长为 " <<

                (time - this->parking.top\_0()->enTime) +

                (this->parking.top\_0()->enTime - this->parking.top\_0()->waitTime)

                << " 应付金额为 "

                << OCCUPY[type] \* (PARK\_PER\_TIME \* (time - this->parking.top\_0()->enTime) +

                    WAIT\_PER\_TIME \* (this->parking.top\_0()->enTime - this->parking.top\_0()->waitTime))

                << endl;

            Car\* todel = this->parking.top\_0();

            this->parking.pop\_0();

            //车站的容量加上刚刚出去的这辆车的容量

            this->size += OCCUPY[todel->type];

            //delete 刚出栈的车

            delete todel;

            flag = 1;

            break;

        }

    }

    if (flag) {

        // 在车库中找到了这辆车，这辆车已经出去了的情况下

        while (!this->parking.empty\_1()) {

            // 将刚刚让路去的车弄进去

            this->parking.push\_0(this->parking.top\_1());

            this->parking.pop\_1();

        }

        // 如果在外面等待的第一辆车符合要求，

        // 也就是车库中可以容纳这辆车

        // 就把在外面等待的1辆车放进车库

        if (!this->wait.empty()) { // 如果外面等待队列不空

            //  如果车库中还有空间

            if (this->size - OCCUPY[this->wait.front()->type] >= 0) {

                this->size -= OCCUPY[this->wait.front()->type];

                cout << "在等待队列中的车 " << this->wait.front()->ID << " 进入车库，时间为 " << time << endl;

                this->wait.front()->enTime = time;

                //设置等待队列最前面的车的入库时间

                this->parking.push\_0(this->wait.front());

                this->wait.pop();

            }

        }

    } else {

        // 在车库中没有找到这辆车，从等候区寻找这辆车

        flag = 0;

        // int findit = 0;

        for (int i = 0; i < this->wait.size(); i++) {

            if (this->wait.front()->ID == id) {

                // 找到了这辆车，这时候front位置就是我们要找的车辆

                cout << "车号 " << this->wait.front()->ID

                    << " 出库，时长为 " << time - this->wait.front()->waitTime

                    << " 应付金额为 " << OCCUPY[type] \* WAIT\_PER\_TIME \* (time - this->wait.front()->waitTime) << endl;

                Car\* todel = this->wait.front();

                this->wait.pop();

                //出停车场的车，delete掉

                delete todel;

                flag = 1;

                break;

            } else {

                // 当前这辆车不是要找的对象， 则将这辆车插入到队列尾部（就算以后找到了也是要插到队列的尾部的）

                this->wait.push(this->wait.front());

                this->wait.pop();

            }

        }

    }

    if (!flag) cout << "车库和等候区中没有这辆车" << endl;

}

**5. 遇到的问题以及解决方案**

问题：Car\* 数据类型在pop函数中delete的时候发生的浅拷贝问题。

解决：

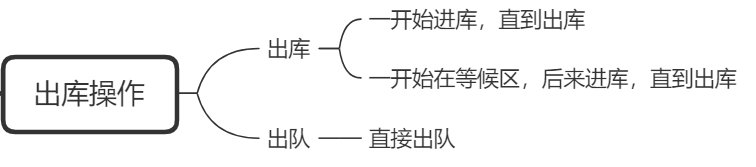
在计算应付金额的时候应该考虑三种情况：

1. 一直在停车场

2. 原来在便道等候，后来进入了停车场

- 这种情况下，应该对车进入车库的时间进行记录所以应该为Car类加上属性waitTime表示进入等候区的时间，因此，当可以直接进入车库的时候waitTime为0

3. 一直在便道等候



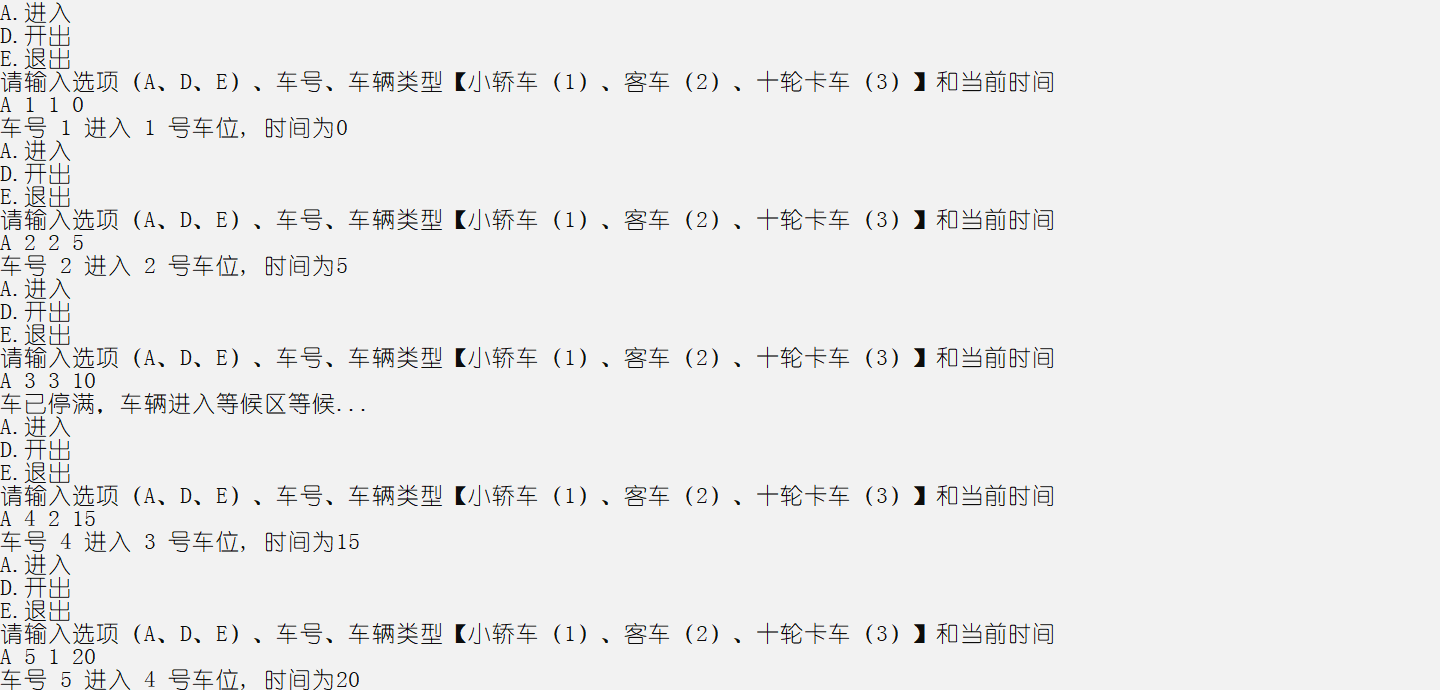
**6. 用户使用说明**

运行程序。

在每一次提示输入的时候输入相应的选项、车号、车辆类型和进入停车场的时间

A表示进入车站，D表示出车站，E表示退出程序

举例描述：



**7. 测试结果**

题目测试数据：

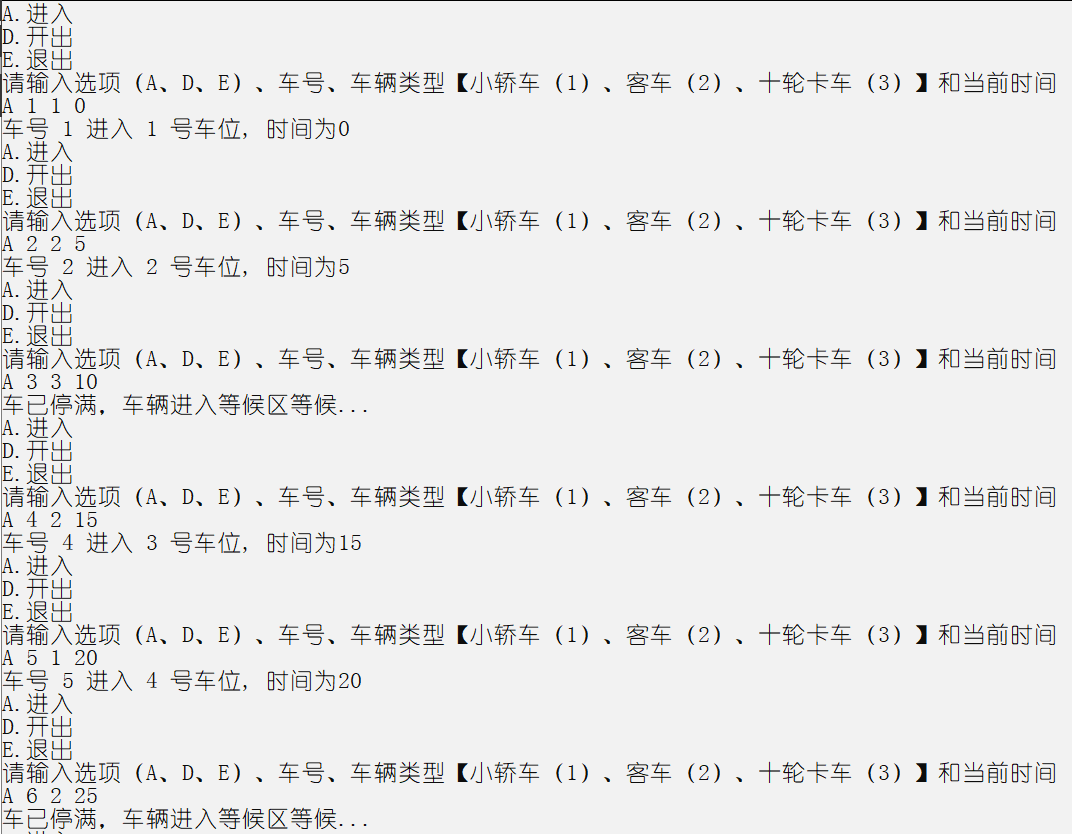
测试数据1

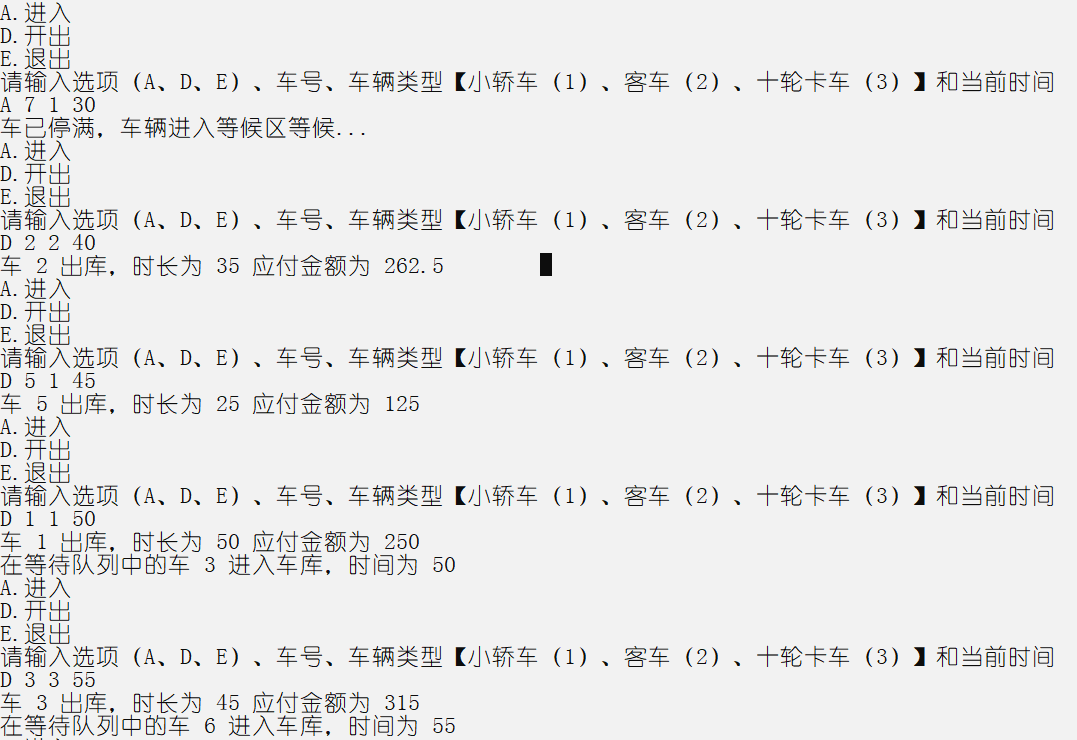
|  |
| --- |
| A 1 1 0  A 2 2 5  A 3 3 10  A 4 2 15  A 5 1 20  A 6 2 25  A 7 1 30  D 2 2 40  D 5 1 45  D 1 1 50  D 3 3 55  D 4 2 60  D 7 1 65  D 6 1 70 |

程序流程：



程序执行截图：







测试数据2。

设 n=2,输入数据为：（‘A’，1，5），（‘A’，2，10），（‘D’，1，15），（‘A’，3， 20），（‘A’，4，25），（‘A’，5，30），（‘D’，2，35），（‘D’，4，40），（‘E’，0，0）。其中，‘A’表示到达；‘D’表示离去，‘E’表示输入结束。



