数据结构实验报告

课程名称：数据结构	班级：计科 151	
姓名： 刘禾子	学号：1500170082	指导教师：刘长云
实验序号：二		实验成绩：

一、实验名称

栈和队列及其应用

二、实验目的及要求

1、熟悉栈的原理和实现方式；
2、能够利用顺序表快速封装栈；
3、熟悉队列的原理和实现方式。

三、实验环境

Visual C++

四、实验内容

【问题描述】

设停车场是一个可停放 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，在它之后进入的车辆必须先退出车场为它让路，待该辆车开出大门外，其他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，以队列模拟停车场外的便道，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，汽车牌照号码以及到达或离去的时刻。对每一组输入数据进行操作后的输出信息为：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；若是车辆离去，则输出汽车在停车场内停留的时间和应交纳的费用（在便道上停留的时间不收费）。栈以顺序结构实现，队列以链表结构实现。

【测试数据】

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),(’ E’ ,0,0)，其中：’ A’ 表示到达；’ D’ 表示离去；’ E’ 表示输入结束。

五、算法描述及实验步骤

算法描述：

设置一个栈用来当作停放车辆的停车场空间，另外还需设一个栈，临时停放为给要离去的车辆让路而从停车场退出来的汽车，也用顺序存储结构实现。输入数据

按到达或离去的时刻有序。栈中的每个元素表示一辆汽车，也包含两个数据项：汽车的牌照号和进入停车场的时刻。
具体定义的结构类型有：
CarNode(每辆车的信息)，QNode(队列节点)，Queue(便道队列)，CarStack(栈)

程序源码：

```
#include<iostream>
using namespace std;
#define MAX_SIZE 2//停车场能够容纳的车的数量
#define FARE 5//表示停车场的收费为每小时FARE元
int CountForStack = 0;// 此全局变量用来计数堆栈现有的车数
int CountForQueue = 0;// 此全局变量用来计数队列现有的车数
typedef struct//这个节点用来保存每辆车的信息
{
    char Condition;//用来表示“到达”或者“离开”的状态," "表示还没有到达，也没有离开
    int ArriveTime;//用来记录到达时间，默认为-1，说明还没有到达
    int LeaveTime;// 用来记录离开时间，默认为-1，说明还没有离开
    int License;// 记录车牌号
}CarNode;
typedef struct //栈的定义
{
    CarNode *base;//栈底指针,指向0
    CarNode *top;//栈顶指针，如果指向0，说明栈为空
    int stacksize;//栈的容量大小
}CarStack;
typedef struct QNode//队列节点的定义
{
    char Condition;//用来表示“到达”或者“离开”的状态," "表示还没有到达，也没有离开
    int ArriveTime;//用来记录到达时间，默认为-1，说明还没有到达
    int LeaveTime;// 用来记录离开时间，默认为-1，说明还没有离开
    int License;// 记录车牌号
    QNode *next;//指向下一个节点的指针
}QNode;
typedef struct// 队列的定义
{
    QNode *front;//队头指针
    QNode *rear;//队尾指针
}Queue;

bool InitStack(CarStack &S)//此函数用来初始化栈
{
    S.base = (CarNode *)malloc(MAX_SIZE*sizeof(CarNode));
    if (!S.base)
    {
```

```
        cout << "内存分配失败！" << endl;
        return false;//说明内存分配失败，返回false
    }
    S.top = S.base;
    S.stacksize = MAX_SIZE;
    return true;
}
bool InitQueue(Queue &Q)//此函数用来初始化队列
{
    Q.front = (QNode *)malloc(sizeof(QNode));
    if (!Q.front)
    {
        cout << "内存分配失败！" << endl;
        return false;
    }
    Q.rear = Q.front;
    Q.front->next = 0;//下一个节点指空
    return true;
}
bool EnQueue(Queue &Q, QNode &qnode)//此函数用来入队一个节点
{
    QNode *p = (QNode *)malloc(sizeof(QNode));
    if (!p)
    {
        cout << "内存分配失败！" << endl;
        return false;
    }
    p->ArriveTime = qnode.ArriveTime;
    p->Condition = qnode.Condition;
    p->LeaveTime = qnode.LeaveTime;
    p->License = qnode.License;
    p->next = 0;
    Q.rear->next = p;
    Q.rear = p;
    return true;
}
bool DeQueue(Queue &Q, QNode &t)//此函数用来出队
{
    if (Q.front == Q.rear)
    {
        cout << "队列为空！" << endl;
        return false;
    }
    QNode *p = Q.front->next;
```

```
    t.ArriveTime = p->ArriveTime;
    t.Condition = p->Condition;
    t.LeaveTime = p->LeaveTime;
    t.License = p->License;
    Q.front->next = p->next;
    if (Q.rear == p)//如果P是指向最后一个出队的元素
        Q.rear = Q.front;
    free(p);
    return true;
}
void InitCarNode(CarNode &C, char condition, int arrivetime, int leavetime, int
license)//本函数用来初始化一个CarNode 节点
{
    C.ArriveTime = arrivetime;
    C.Condition = condition;
    C.LeaveTime = leavetime;
    C.License = license;
}
bool Push(CarStack &S, CarNode &car)//此函数用来入栈一个CarNode 节点
{
    if (S.top - S.base >= S.stacksize)
    {
        cout << "此栈已满，不能压入新的信息" << endl;
        return false;
    }
    (*S.top).ArriveTime = car.ArriveTime;
    (*S.top).Condition = car.Condition;
    (*S.top).LeaveTime = car.LeaveTime;
    (*S.top).License = car.License;
    ++S.top;//栈顶指针上移
    return true;
}
bool Pop(CarStack &S, CarNode &t)//此函数用来弹出栈内元素
{
    if (S.top == S.base)
    {
        cout << "栈空，不能执行出栈操作！" << endl;
        return false;
    }
    --S.top;//栈顶指针下移
    t.ArriveTime = (*S.top).ArriveTime;
    t.Condition = (*S.top).Condition;
    t.LeaveTime = (*S.top).LeaveTime;
    t.License = (*S.top).License;
```

```
    return true;
}
bool IsStackFull(CarStack &S)//此函数用来判断堆栈是否已满
{
    if (S.top - S.base >= S.stacksize)
        return true;
    else
        return false;
}
bool IsStackEmputy(CarStack &S)//此函数用来判断堆栈是否为空
{
    if (S.top == S.base)
        return true;
    else
        return false;
}
bool IsQueueEmputy(Queue &Q)//此函数用来判断队列是否为空
{
    if (Q.front == Q.rear)
        return true;
    else
        return false;
}
bool SearchInStack(CarStack&S, int a)//a表示要查找的车牌号,如果在停车场里面，就返回true
{
    bool tag = false;
    if (!IsStackEmputy(S))
    {
        CarNode *p = S.top - 1;
        while (p != S.base)
        {
            if ((*p).License == a)
                tag = true;
            --p;
        }
        if ((*p).License == a)
            tag = true;
    }
    return tag;
}
bool SearchInQueue(Queue &Q, int a)//a表示要查找的车牌号,如果在通道里面，就返回true
{
    bool tag = false;
    if (!IsQueueEmputy(Q))//如果队列非空
```

```
    {
        QNode *p = Q.front->next;
        while (p != Q.rear)
        {
            if ((*p).License == a)
                tag = true;
        }//退出此while循环时p指向最后一个元素
        if ((*p).License == a)
            tag = true;
    }
    return tag;
}
void InCar(CarStack &S, Queue &Q, int a1, int a2)//此函数用来表示进入车辆，参数a1用来表示到达时间，参数a2表示车牌号码
{
    if (SearchInStack(S, a2))
    {
        cout << "车号" << a2 << "已经存在于停车场内，输入有误" << endl;
        return;
    }
    if (SearchInQueue(Q, a2))
    {
        cout << "车号" << a2 << "已经存在于通道内，输入有误" << endl;
        return;
    }

    if (IsStackFull(S))//如果堆栈已满，说明停车场已满，需要停车在通道
    {
        QNode qnode;
        qnode.ArriveTime = -1;//在通道时间不收费，所以不计时
        qnode.Condition = 'A';
        qnode.LeaveTime = -1;//定义为-1，说明还没有开始离开
        qnode.License = a2;
        EnQueue(Q, qnode);//停在通道上
        ++CountForQueue;
        cout << "车号：" << qnode.License << "停在通道的第" << CountForQueue << "号位置" << endl;
    }
    else
    {
        CarNode carnode;
        carnode.ArriveTime = a1;
        carnode.Condition = 'A';
        carnode.LeaveTime = -1;
```

```
            carnode.License = a2;
            Push(S, carnode);
            ++CountForStack;
            cout << "车号：" << carnode.License << "到达时间 " << carnode.ArriveTime << "
停在停车场的第" << CountForStack << "号位置" << endl;
        }
}
void OutCar(CarStack &S, Queue &Q, int a1, int a2)//此函数用来出车，参数a1用来表示离开
时间，参数a2表示车牌号码
{
    if (SearchInQueue(Q, a2))
    {
        cout << "车号" << a2 << "存在于通道内，还未进入停车场，不能离开" << endl;
        return;
    }
    if (!SearchInStack(S, a2))
    {
        cout << "车号" << a2 << "不在停车场内，输入有误" << endl;
        return;
    }
    CarStack tempstack;
    InitStack(tempstack);//建立并且初始化用于暂存出车时让车的堆栈
    bool tag1 = false;//标志这个停车场出车以前是否已满，默认为没有满
    tag1 = IsStackFull(S);
    bool tag2 = true;//标志通道是否有汽车在等待，默认为通道为空
    tag2 = IsQueueEmputy(Q);
    CarNode temp;//用来保存暂时取出的汽车
    bool tag3 = false;//用来标志是否是离开时间小于到达时间而导致离开失败，true表示离开
失败
    while (1)//让车离开
    {
        Pop(S, temp);
        if (temp.License == a2)
        {
            if (a1<temp.ArriveTime)

            {
                cout << "输入有误，离开时间不能小于到达时间，离开失败" << endl;
                tag3 = true;
                Push(tempstack, temp);
            }
            else
                cout << "车号" << a2 << "现在离开停车场，所用的时间为" << a1 -
temp.ArriveTime << "收费为" << (a1 - temp.ArriveTime)*FARE << endl;
```

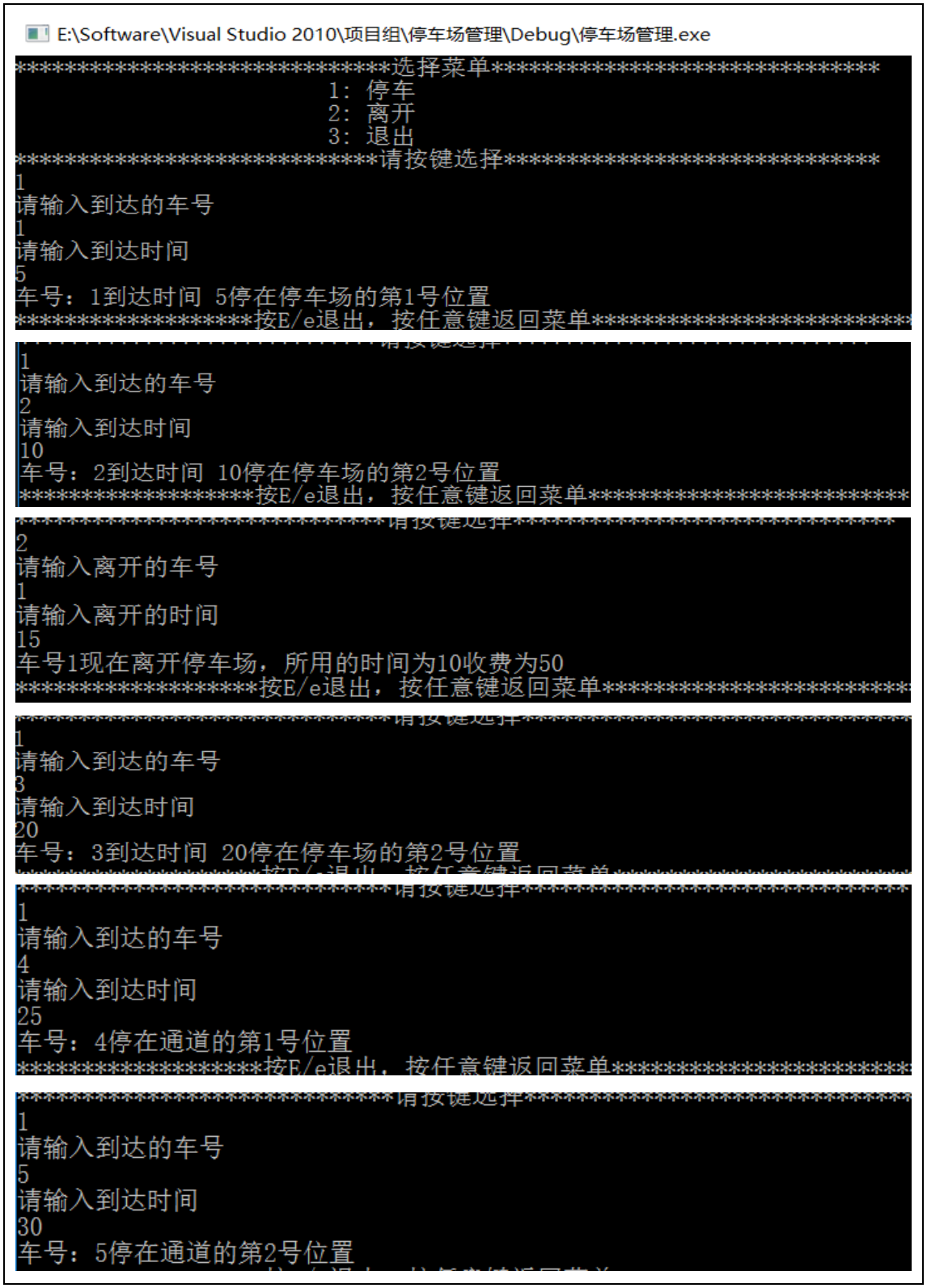
```
            break;
        }
        else
            Push(tempstack, temp);//进入暂存栈
    }
    while (!IsStackEmputy(tempstack))//倒出的车再次进入停车场
    {
        Pop(tempstack, temp);
        Push(S, temp);
    }
    QNode tempqnode;//用来暂时保存从通道出来的汽车
    if (tag1 == true && tag2 == false && tag3 == false)//如果出车前停车场已满，并且通道不为空,并且离开没有失败
    {
        DeQueue(Q, tempqnode);
        --CountForQueue;
        temp.ArriveTime = a1;
        temp.Condition = tempqnode.Condition;
        temp.LeaveTime = tempqnode.LeaveTime;
        temp.License = tempqnode.License;
        Push(S, temp);
    }
    if (tag2 == true && tag3 == false)// 如果停车场没有满，并且离开成功
        --CountForStack;
}
void showmenu(CarStack &S, Queue &Q)
{
    cout << "*****************************选择菜单*****************************" << endl;
    cout << "                          1: 停车" << endl;
    cout << "                          2: 离开" << endl;
    cout << "                          3: 退出" << endl;
    cout << "*****************************请按键选择*****************************" << endl;
    int tag;
    cin >> tag;
    while (tag != 1 && tag != 2 && tag != 3)
        cin >> tag;
    int a1;
    unsigned int a2;
    switch (tag)
    {
    case 1:
        cout << "请输入到达的车号" << endl;
```

```
            cin >> a1;
            cout << "请输入到达时间" << endl;
            cin >> a2;
            InCar(S, Q, a2, a1);
            break;

        case 2:
            cout << "请输入离开的车号" << endl;
            cin >> a1;
            cout << "请输入离开的时间" << endl;
            cin >> a2;
            OutCar(S, Q, a2, a1);
            break;
        case 3:
            return;
            break;
        }
        char ch;
        cout << "*******************按E/e退出，按任意键返回菜单*************************"
<< endl;
        cin >> ch;
        if (ch != 'E'&&ch != 'e')
            showmenu(S, Q);
}
void main()
{
        CarStack carstack;
        InitStack(carstack);// 建立并且初始化用于停车场的堆栈
        Queue carqueue;
        InitQueue(carqueue);//建立并且初始化用于通道的队列
        showmenu(carstack, carqueue);
}
```

六、调试过程及实验结果

调试过程中出现暂存栈中的车辆离开后，回退时再次进入栈内，下次访问时出现无法正常离栈，在调试暂时出来的车辆回退栈的时候出现问题，重新用 push 函数压入停车场从而解决。


E:\Software\Visual Studio 2010\项目组\停车场管理\Debug\停车场管理.exe
**********选择菜单**********
1：停车
2：离开
3：退出
**********请按键选择**********
1
请输入到达的车号
1
请输入到达时间
5
车号：1到达时间 5停在停车场的第1号位置
**********按E/e退出，按任意键返回菜单**********
1
请输入到达的车号
2
请输入到达时间
10
车号：2到达时间 10停在停车场的第2号位置
**********按E/e退出，按任意键返回菜单**********
2
请输入离开的车号
1
请输入离开的时间
15
车号1现在离开停车场，所用的时间为10收费为50
**********按E/e退出，按任意键返回菜单**********
1
请输入到达的车号
3
请输入到达时间
20
车号：3到达时间 20停在停车场的第2号位置
1
请输入到达的车号
4
请输入到达时间
25
车号：4停在通道的第1号位置
**********按E/e退出，按任意键返回菜单**********
1
请输入到达的车号
5
请输入到达时间
30
车号：5停在通道的第2号位置

七、总结

1.在出栈过程中应注意先让栈顶指针变化再出栈
2.深刻理解队列的操作，从队头出从队尾入