栈、队列与递归算法设计

1. 需求分析

栈（stack）的典型操作是入栈（push）和出栈（pop），前者将新元素压入栈中，后者弹出栈顶元素。栈只提供对栈顶元素的访问操作，由 top 完成。push、pop 和 top 三者构成栈的最小功能接口。为方便使用，栈还应提供若干基本操作。这些操作可按使用型（或称只读访问型）操作和加工型操作进行分类。

实验要求如下：

1）建立一个顺序栈。

2）建立一个循环顺序队列。

3）分别实现栈和队列的基本操作。

4）商品货架可以看成一个栈,栈顶商品的生产日期最早,栈底商品的生产日期最晚。上货时,需要倒货架,以保证生产日期较晚的商品在货架较下的位置。针对一种特定商品,实现上述管理过程。用栈模拟货架和周转空间。

5）设停车场内是一个可停放 n 辆汽车的狭长区域,且只有一个大门可供汽车进出。在停车场内,汽

车按到达时间的先后顺序,依次由北向南排列(假设

大门在最南端,最先到达的那辆车停放在车场的 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,待该车开出大门外,其他车再按原次序返回停车场。每辆停放在车场的车离开停车场时,必须 按其停留时间的长短交费。试为停车场编制按上述要求进行管理的模拟程序。以顺序栈模拟停车场,以链队列模拟便道,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 3 项:是“到达”还是“离去”、汽车牌照号码及“到达”或“离去”的时刻。对每一 组输入数据进行操作后的输出数据为:如果是到达的车辆,则输出其在停车场内或便道上的停车位置; 如果是离去的车辆,则输出其在停车场内的停留时间和应交的费用(在便道上停留的时间不收费)。 栈以顺序结构实现,队列以链表实现。需另设一个栈,临时停放为给要离去的汽车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。栈中每个元素表示一辆汽车,包含两个数据项:汽车的牌照号码和进入停车场的时刻。

(1) 输入的形式和输入值的范围：系统定义的基本类型。

(2) 输出的形式：系统定义的基本类型。

(3) 程序所能达到的功能：以栈的数据结构储存数据、以队列的数据结构存储数据 、用栈实现商品货架管理、用栈和队列综合实现停车场管理系统。

(4) 测试数据：依次输入 1、2、3、4、5，然后对栈进行 top&pop 操作，查看输出结果。

2. 概要设计

(1)栈类的基本操作：

void InitStack(Stack &S,int num = 10)

操作结果：构造了一个栈，栈的默认空间为 10。

void DestoryStack(Stack &S)

初始条件：栈已存在

操作结果：销毁该栈。

void ClearStack(Stack &S)

初始条件：栈已存在

操作结果：清空该栈。

int GetTop(Stack &S,Node &n)

初始条件：栈已存在且非空（为空则返回 0）

操作结果：得到栈顶元素。

int IsEmpty(Stack S)

初始条件：栈已存在

操作结果：判断栈是否为空，为空返回 1，否则返回 0

int Pop(Stack &S,Node &n)

初始条件：栈已存在且非空（为空则返回 0）

操作结果：出栈，从栈中删除栈顶元素，并将值赋给传入的变量引用。

void Push(Stack &S,int num1)

初始条件：栈已存在

操作结果：入栈一个元素，并返回修改后的栈 。

void PrintStack(Stack S)

初始条件：栈已存在

操作结果：依次输出栈内的数据。

(2)用栈来实现货物管理

void PushGoods(Stack &S)

初始条件：栈已存在且栈未满（栈满则在控制台显示）

操作结果：将栈倒空，让货物入栈，之后再将之前货物倒回。

void SellGoods(Stack &S)

初始条件：栈已存在且栈不为空（栈空则在控制台显示）

操作结果：卖出货物，将栈顶取出。

main()

初始条件：无。

操作结果：实现菜单以及各项功能。

(3)链式队列类的基本操作：

void InitQueeue(Queeue &Q)

操作结果：构造了一个空的循环链式队列。

int GetlenthQueeue(Queeue &q)

初始条件：队列已存在

操作结果：得到队列长度。

int IsEmptyQueeue(Queeue &Q)

初始条件：队列已存在

操作结果：判断队列是否为空，为空返回 1，否则返回 0

void EnQueeue(Queeue &Q,string num,int data)

初始条件：队列已存在

操作结果：入队一个元素，并返回修改后的队列 。

int DeQueeue(Queeue &Q,string &num,int &data)

初始条件：队列已存在且队列不为空（为空则返回 0）

操作结果：出队一个元素，并返回修改后的队列并将值赋给传入的变量引用。

void PrintQueeue(Queeue Q)

初始条件：队列已存在

操作结果：依次输出队列内的数据。

(4)用栈和队列来实现停车场管理

void arrive(Stack &parkinglot,Queeue &road)

初始条件：模拟停车场的栈已存在且栈未满（栈满则在控制台显示）

操作结果：若栈未满，入栈，停车记录到达时间；若栈已满，让车辆

在模拟过道的队列中等候。

void leave(Stack &parkinglot,Queeue &road,Stack &waitline)

初始条件:模拟停车场的栈已存在且栈不为空(栈空则在控制台显示)

操作结果：将停车场内车辆驶出，如果该车辆后方有车辆，依次让道（进入另一个栈），该车辆离开并缴费后，让道车辆回到停车场中，此时如果有车辆在过道等候，则进入停车场。

main()

初始条件：无。

操作结果：实现菜单以及各项功能。

3. 详细设计

（1）栈类基本内容———头文件

```
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
using namespace std;


typedef struct {

    int num1;
    int num2;
    int num3;
}Node;
```

```
typedef struct{


    Node *top;
    Node *base;
    int size_num;
}Stack;


void InitStack(Stack &S,int num = 10)//新建栈
{
    S.base=(Node *)malloc(num*sizeof(Node));
    S.top=S.base-1;//先挖坑再种萝卜
    S.size_num = num;
}

void DestoryStack(Stack &S)//撤销栈
{
    free(S.base);
}

void ClearStack(Stack &S)
{
    S.top=S.base-1;
}


int GetTop(Stack &S,Node &n)
{

    if(S.top==S.base-1)
    {

        return 0;
    }

    n = *S.top;
    return 1;
}

int IsEmpty(Stack S)
```

```
{

    if(S.top-S.base == -1)
        return 1;



    else return 0;
}
int IsFull(Stack S)
{
    if(S.top-S.base>=S.size_num-1)
    {

        return 1;
    }
    else return 0;
}

void Push(Stack &S,int num1)
{

    S.top++;
    S.top->num1=num1;




}


int Pop(Stack &S,Node &n)
{
    if(S.top-S.base == -1)
    {

        return 0;
    }



    n.num1 = S.top->num1;
```

```
    S.top--;
    return 1;


}
```

```
void PrintStack(Stack S)
{
    Node* p = S.base;
    if(p==S.top+1)
    {cout<<endl<<"货架内无货物！"<<endl;return;}
    int i=0;
    while(p!=(S.top+1))
    {
        cout<<"货架从下到上里第"<<++i<<"个货物编号为："
"<<p->num1<<endl;
        p++;
    }
}
```

（2）模拟商品货架———可执行 cpp 文件

```
#include "Stack.h"
#include <iostream>
using namespace std;



//货架管理
//加货时将货架倒出，最新的货物加到最低，再将货物依次倒入货架
//取货时从货架顶上取，确保取到的为最早生产的
//用栈模拟

void PushGoods(Stack &S)
{

    Stack SS;
    InitStack(SS,1000);
```

```
     if(IsFull(S)==1)
    {
        cout<<"货架已满！"<<endl<<endl;
        return;
    }



    while(true)
    {   Node n1;
        if(IsEmpty(S)==1)break;
        Pop(S, n1);
        Push(SS, n1.num1);

    }

          int num;
cout<<"请输入你要添加的货物编号："<<endl;
    cin>>num;
    cout<<endl<<endl;
    Push(S,num);



    while(true)
    { Node n2;
        if(IsEmpty(SS)==1)break;
        Pop(SS, n2);
        Push(S, n2.num1);

    }

}



void SellGoods(Stack &S)
{
    Node n;
    if(IsEmpty(S) == 1){
        cout<<endl<< "货物已取完，货架内无货物！"<<endl<<endl;
        return;
    }
```

```
    Pop(S, n);
    cout<<endl<<"成功取出编号为"<<n.num1<<"的货物"<<endl;


}


int main() {
    Stack S;
    InitStack(S);
    while(true)
    {
        cout<<"=======">>"货架管理系统"<<"=======">>endl;
        cout<<"           1.加货"<<endl<<endl;
        cout<<"           2.取货"<<endl<<endl;
        cout<<"           3.查看"<<endl<<endl;
        cout<<"=======">>"=======">>"============">>endl;
        int i;
        cin>>i;
        cout<<endl;
        switch(i){

            case 1:PushGoods(S);break;
            case 2:   SellGoods(S);break;
            case 3:PrintStack(S);break;
            case 0: return 0;
            default: continue;


        }


    }


}
```

（3）停车场管理中栈表类基本内容（稍作改进）

```
#include <iostream>
#include <stdlib.h>
```

```
#include <stdio.h>
using namespace std;


typedef struct {

    string num1;
    int num2;

}Node;




typedef struct{


    Node *top;
    Node *base;
    int size_num;
}Stack;


void InitStack(Stack &S,int num = 5)//新建栈
{
    S.base=(Node *)malloc(num*sizeof(Node));
    S.top=S.base-1;//先挖坑再种萝卜
    S.size_num = num;
}

void DestoryStack(Stack &S)//撤销栈
{
    free(S.base);
}

void ClearStack(Stack &S)
{
    S.top=S.base-1;
}


int IsEmptyStack(Stack &S)
{
```

```
    if(S.top==S.base-1)return 1;
    else return 0;
}
int IsFullStack(Stack &S)
{
    if(S.top-S.base>=S.size_num-1)return 1;
    else return 0;
}



int GetTop(Stack &S,string &str)
{

    if(S.top==S.base-1)
    {

        return 0;
    }

    str = (*S.top).num1;
    return 1;
}


int Push(Stack &S,string num1,int num2)
{
    if(S.top-S.base>=S.size_num-1)
    {

        return 0;
    }
    S.top++;
    (*S.top).num1=num1;
    (*S.top).num2=num2;


    return 1;

}


int Pop(Stack &S,string &str,int &data)
{
```

```
    if(S.top-S.base == -1)
    {

        return 0;
    }


    //n.num1 = 1;
    //n.num2 = 2;
    //n.num2 = 3;
    str = (*S.top).num1;
    data = (*S.top).num2;


    S.top--;
    return 1;


}


void PrintStack(Stack S)
{
    Node* p = S.base;
    if(p==S.top+1)
    {cout<<endl<<"停车场内无车辆！"<<endl;return;}
    int i=0;
    while(p!=(S.top+1))
    {
        cout<<"停车场里第"<<++i<<"辆车的车牌号为: "<<p->num1<<endl;
        p++;
    }
}


int charge(Stack S,string id)
{
    Node* p = S.base;
    bool i = false;

    while(p!=(S.top+1))
    {
        if(p->num1==id)
        {i=true;break;}
```

```
        p++;
    }
    if(i==true)return 1;
    else return 0;

}
```

（4）停车场管理中链式循环队列头文件

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



typedef struct QNode{

    string num;
    int data;
    QNode *next;
}QNode,*Queeue;

QNode *head,*rear;

void InitQueeue(Queeue &Q){
    head = (QNode *)malloc(sizeof(QNode));
    Q = head;
    rear = head;
    head->next = head;

}

int GetlenthQueeue(Queeue &q)
{


    QNode *p = head;
```

```
    int num=0;
    while(p->next!=head){
        p=p->next;
        num++;
    }
    return num;
}
int IsEmptyQueeue(Queeue &Q)
{
    if(head->next == head)return 1;
    return 0;
}

void EnQueeue(Queeue &Q,string num,int data){
    QNode *newnode = (QNode *)malloc(sizeof(QNode));


    rear->next = newnode;
    newnode->next = head;
    rear = newnode;//采用尾插法入队
    newnode->num = num;
    newnode->data = data;

}

int DeQueeue(Queeue &Q,string &num,int &data){


    if(head->next==head)return 0;//队列为空


    QNode *oldnode = head->next;

    if(oldnode == rear)rear=head;

    head->next = oldnode->next;
    num = oldnode->num;
    data = oldnode->data;
    free(oldnode);
    return 1;

}

void PrintQueeue(Queeue Q)
```

```
{



        QNode* p = head;

    if(p->next==head)
    {cout<<endl<<"过道内无车辆等候！"<<endl;return;}
    int i=0;
        while(p->next!=head)
        {
        cout<<"过道里等候的第"<<++i<<"辆车的车牌号为："<<p->next->num<<endl;
          p=p->next;
    }

}
```

（5）停车场管理的实现

```
#include <iostream>
using namespace std;
#include "Queeue-Link.h"
#include "Stack.h"
#define FEE 10  //每小时停车费用十元

//停车场问题
//假设停车场内空间为5，过道空间无限
//停车场内部用栈表示，过道空间用队列表示
//负责暂时停放让行车辆的假设也在过道，但与等候车辆不属于同一过道，用栈表示，假设空间也为10




void arrive(Stack &parkinglot,Queeue &road){

    string id;
    int data;

    cout<<"请输入到来车辆的车牌号：";
    cin>>id;
    cout<<endl<<"请输入车辆到来时间：";
```

```
    cin>>data;
    cout<<endl;

    if(IsFullStack(parkinglot)==1)
    {
       EnQueeue(road,id,data);
        cout<<"停车场已满，暂时在外排队等候"<<endl<<endl;
    }

    else{
        if(IsFullStack(parkinglot)==0)
        {
            Push(parkinglot,id,data);
            cout<<"开始计费"<<endl<<endl;
        }

    }

}

void leave(Stack &parkinglot,Queeue &road,Stack &waitline){


    if(IsEmptyStack(parkinglot))
       {
           cout<<endl<<"没有车可以离开，停车场为空!"<<endl;
           return;
       }



    string id;
    int data;

    cout<<"请输入要离开车辆的车牌号：";
    cin>>id;
    int charge_id = charge(parkinglot, id);
    if(charge_id==0)
    {
        cout<<endl<<"停车场里没有此车！请确认！"<<endl<<endl;
        return;
    }

    cout<<endl<<"请输入车辆离开时间：";
```

```
    cin>>data;
    cout<<endl;


    while(true)
    {
        string id2;
        int data2;
        Pop(parkinglot, id2, data2);
        if(id2==id)//车辆已出停车场，缴费
        {
            int fee;
            fee = FEE*(data - data2);
            cout<<"请缴费"<<fee<<"元"<<endl;
            break;
        }
        else{
            Push(waitline, id2, data2);//后面车辆依次退出停车场，让行
        }
    }

    while(!IsEmptyStack(waitline))//暂时离开的车辆依次进入停车场
    {
        string id3;
        int data3;

        Pop(waitline, id3, data3);
        Push(parkinglot, id3, data3);
    }
  //由于每次函数执行总是仅仅出来一辆车，所以过道队列中如果有车辆，仅仅进来一辆即可
    if(!IsEmptyQueeue(road))
    {
        string id4;
        int data4;
        DeQueeue(road, id4, data4);
        Push(parkinglot, id4, data4);
    }



}

void start(Stack &parkinglot,Queeue &road,Stack &waitline)
```

```
{

    while(true)
    {
        cout<<"=======停车场管理系统"<<"=======<<endl;
        cout<<"           1.泊车"<<endl<<endl;
        cout<<"           2.离开"<<endl<<endl;
        cout<<"           3.查看"<<endl<<endl;
        cout<<"=======<<"=======<<"============"<<endl;
        int i;
        cin>>i;
        switch (i) {
            case 1:
                arrive(parkinglot, road);
                break;
            case 2:leave(parkinglot, road, waitline);
                break;

            case 3:PrintStack(parkinglot);PrintQueeue(road);break;
               default:continue;

        }


    }


}


int main()
{
    Queeue road;
    Stack parkinglot;
    Stack waitline;


    InitStack(parkinglot,3);
    InitStack(waitline,3);
    InitQueeue(road);

    start(parkinglot,road,waitline);
```

```
    return 0;
}
```

4. 调试分析

对栈和队列进行基本测试，测试完毕，可正常使用。(测试结果一并在测试结果中显示)

<1>调试过程中遇到的问题：

货架管理一开始调试时遇到的问题时，无论怎么输入，取出货物的时候编号都为 0，开始百思不得解，后来经检查发现，原来把栈的两种模式——先挖再栽和先栽再挖给搞混了了，导致不正常的输出。

停车场问题开始测试发现没什么问题，但在使用过程中发现有时候数据会乱，经检查发现，忘了判断栈空，补加后功能恢复正常。

<2>时间和空间分析；

查询功能所用时间为遍历栈和队列所耗时间，删除功能也为遍历栈和队列所耗时间，均为 O(n)。

栈和队列占用空间不定，为每个结点所占用空间之和。

<3>改进设想：

链式队列的好处是可以有无限的空间，如果用顺序表循环队列就可

能出现队列满的情况，当然，这种情况在有些时候是必须的，但本题中，过道空间可以视为无限，所以最后改成了链式队列。

<4>经验、心得和体会：

经过上次设计链表，这次明显定义结构体的时候熟练了很多，而且也巩固了很多上次实验没用到的C语言知识，受益颇深。

5. 使用说明

1、对于货架管理：

1）输入 1，向货架内增添货物，货物置于货架底部。

2）输入 2，从货架顶部取出货物。

3）输入 3，展示货架内部所有货物。

2、对于停车场管理：

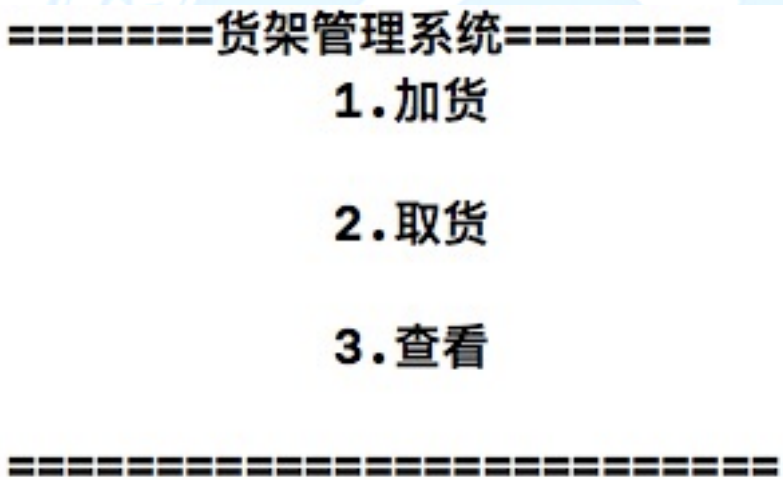
1）输入 1，停车场停入车辆，开始计费，若停车场已满则在外等候。

2）输入 2，停车场内车辆离开并缴费，过道内等候车辆进入停车场。

3）输入 3，展示停车场和过道内所有车辆。

6. 测试程序的运行结果

1、货架管理：



（1）向货架内依次增添货物：

```
============================
1

请输入你要添加的货物编号:
1111


=======货架管理系统=======
                1.加货

                2.取货

                3.查看

============================
1

请输入你要添加的货物编号:
2222


=======货架管理系统=======
                1.加货

                2.取货

                3.查看

============================
```

（2）当放入四件后，依次显示

```
==============================
1

请输入你要添加的货物编号:
4444


=======货架管理系统=======
                1.加货

                2.取货

                3.查看

==============================
3

货架从下到上里第1个货物编号为： 4444
货架从下到上里第2个货物编号为： 3333
货架从下到上里第3个货物编号为： 2222
货架从下到上里第4个货物编号为： 1111
```

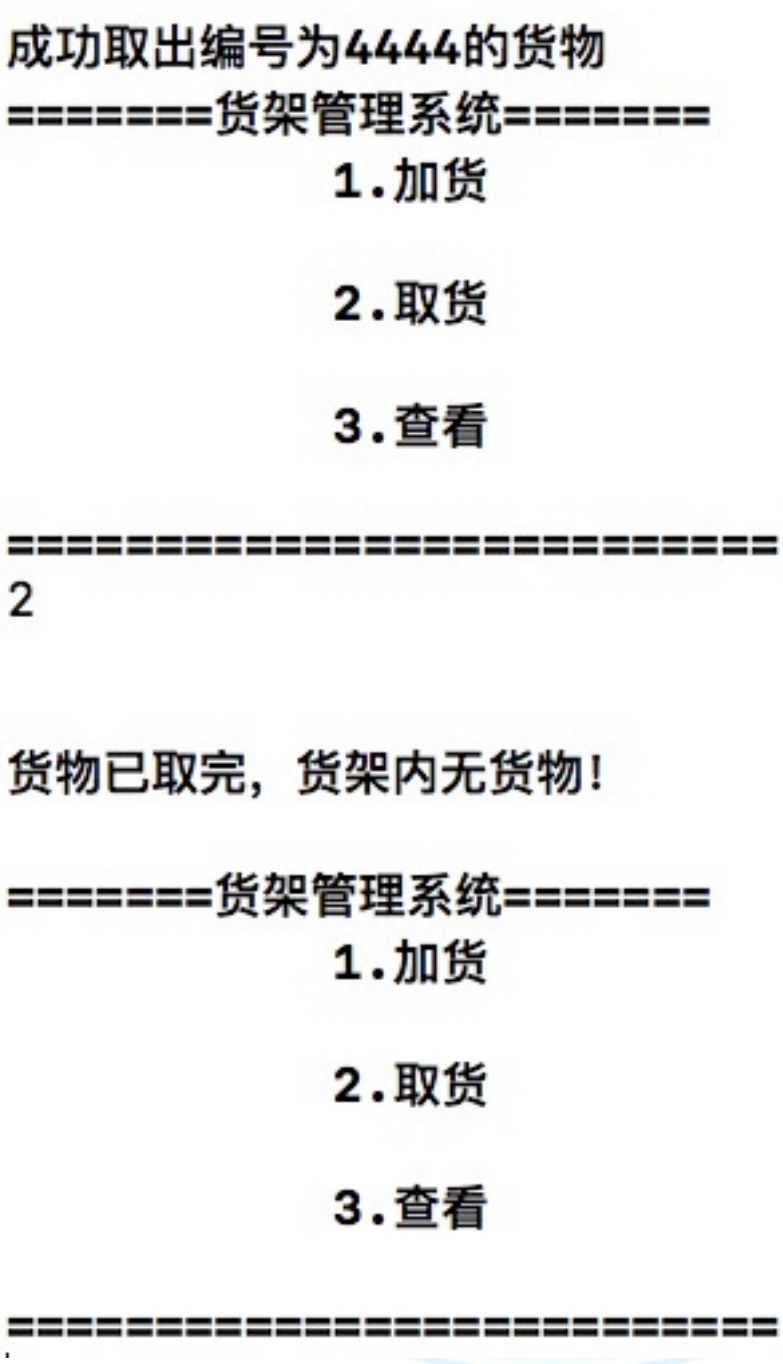
（3）取出货架顶部的货物

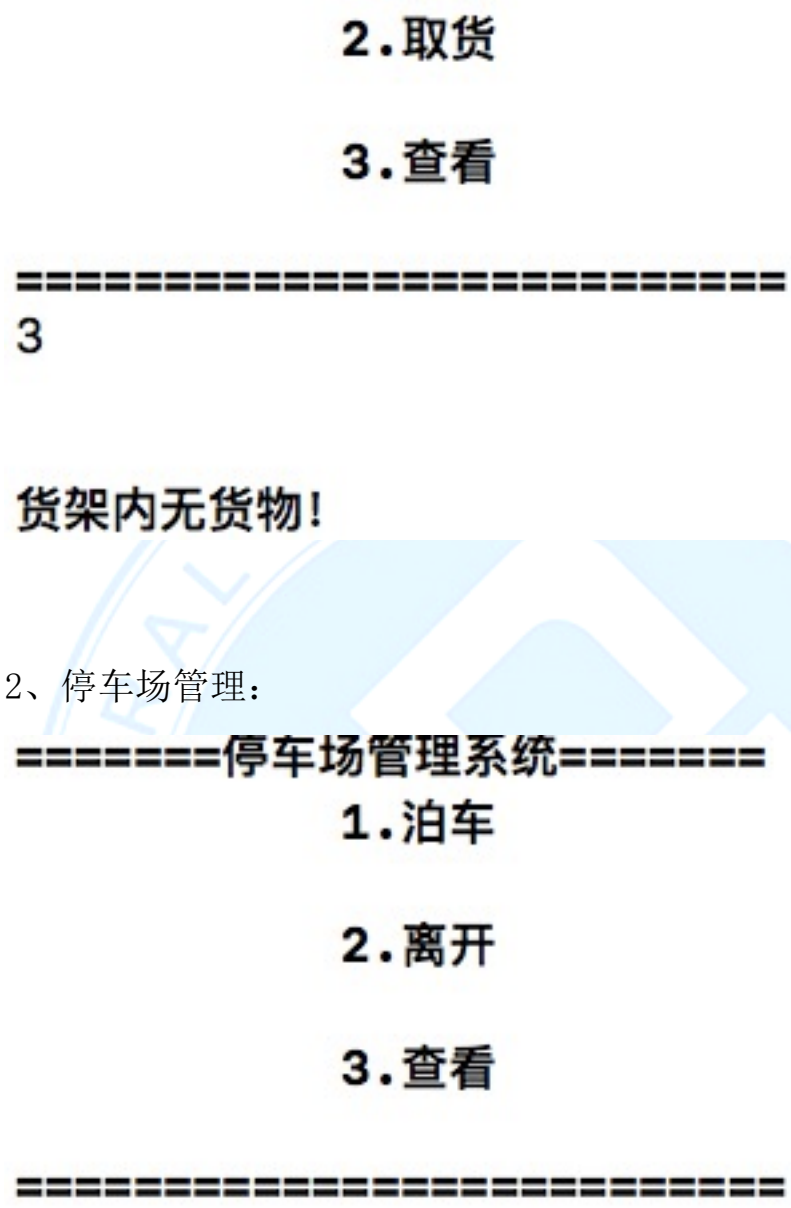
```
=======货架管理系统=======
                1.加货

                2.取货

                3.查看

============================
2


成功取出编号为1111的货物
```

（4）显示现在货架内的商品

```
=======货架管理系统=======
                1.加货

                2.取货

                3.查看

============================
3

货架从下到上里第1个货物编号为： 4444
货架从下到上里第2个货物编号为： 3333
货架从下到上里第3个货物编号为： 2222
```

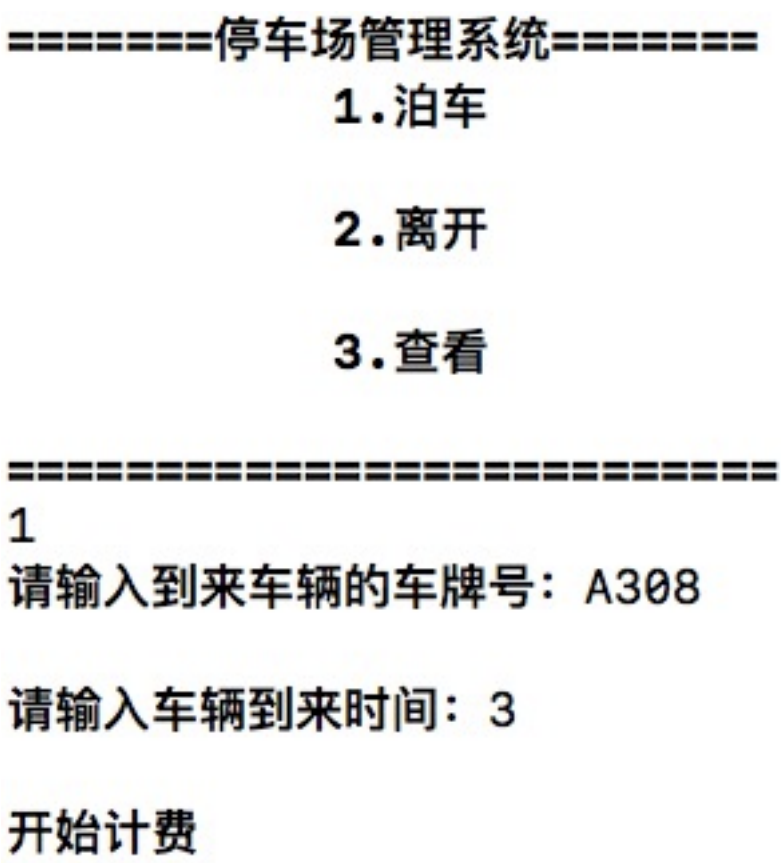
（5）当货架为空时取货和查看





2、停车场管理：

（1）测试数据默认停车场内可以停放 3 辆车，先依次停入三辆车



（2）第四辆车到来时显示在外面等候

开始计费

========停车场管理系统========

1.泊车

2.离开

3.查看

==============================

1

请输入到来车辆的车牌号：D504

请输入车辆到来时间：7

停车场已满，暂时在外排队等候

（3）第一辆车离开停车场

```
=======停车场管理系统=======
                1.泊车

                2.离开

                3.查看

============================
2
请输入要离开车辆的车牌号：A308

请输入车辆离开时间：15

请缴费120元
```

（4）显示停车场信息

```
=======停车场管理系统=======
                1.泊车

                2.离开

                3.查看

============================
3
停车场里第1辆车的车牌号为： B207
停车场里第2辆车的车牌号为： C111
停车场里第3辆车的车牌号为： D504

过道内无车辆等候!
```

（5）再来一辆车后，在过道等候，显示停车场信息

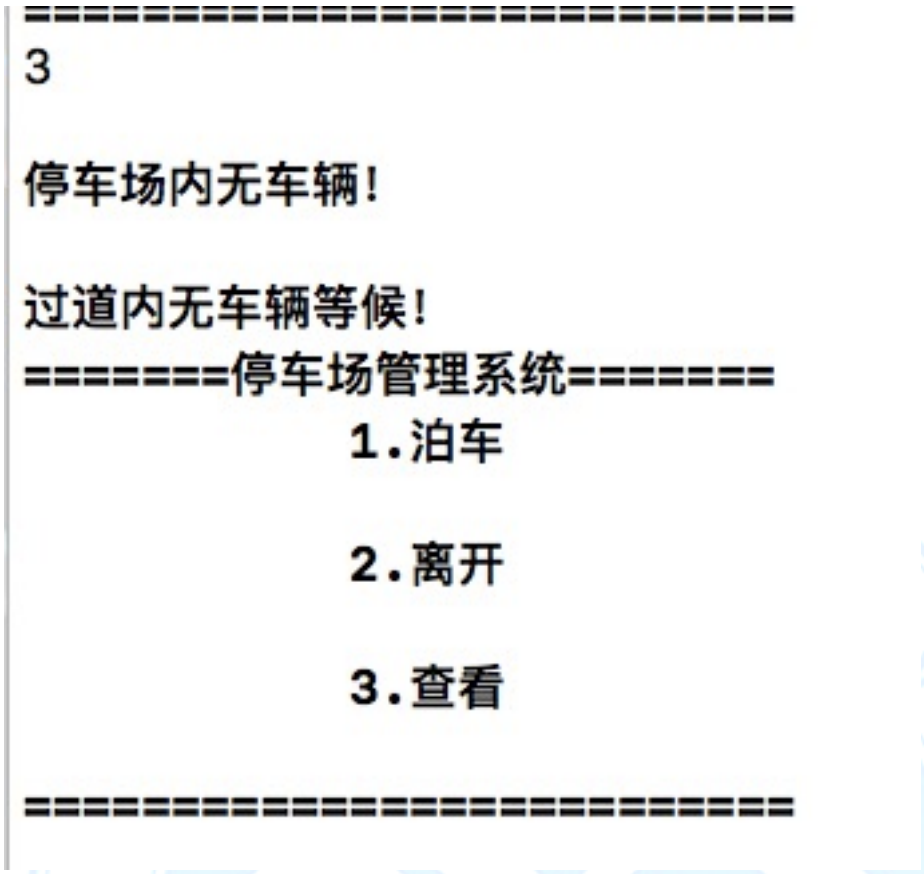
```
============================
1
请输入到来车辆的车牌号：E320

请输入车辆到来时间：16

停车场已满，暂时在外排队等候

=======停车场管理系统=======
                1.泊车

                2.离开

                3.查看

============================
3
停车场里第1辆车的车牌号为： B207
停车场里第2辆车的车牌号为： C111
停车场里第3辆车的车牌号为： D504
过道里等候的第1辆车的车牌号为： E320
```

（6）若车牌号输入错误，显示

```
=======停车场管理系统=======
                1.泊车

                2.离开

                3.查看

============================
2
请输入要离开车辆的车牌号：qqq

停车场里没有此车！请确认！
```

（7）若停车场为空，显示

```
=======停车场管理系统=======
                1.泊车

                2.离开

                3.查看

============================
2

没有车可以离开，停车场为空！
```



7. 心得体会

这一次做的实验比上一次就庞大的多，但是还不算困难，所以完成的也比较快，后期加入菜单等操作温习了 C 语言课程设计时实用的一些方法，栈和队列的应用十分广泛，这次可以说是很好的巩固了课堂上学习的内容，受益颇深。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包,存放在 bin 子目录。文件清单如下：

Stack.h……基本栈的定义及基本操作（货架管理），头文件

Stack 2.h……基本栈的定义及基本操作（停车场），头文件

Queeue-Link.h……链式队列的定义及基本操作，头文件

Goods.cpp……货物管理实验算法实现

ParkingLot.cpp ……停车场管理算法实现