第1.3 题：停车场管理

【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度交纳费用。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“离开”信息、汽车牌照号码以及到达或离去的时刻。对每一个数据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或便道上的停车位置；若是车辆离开，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)，(‘D’,2,35)，(‘D’,4,40)，(‘E’,0,0)。其中：’A’表示到达（Arrival）；’D’表示离去（Departure）；’E’表示输入结束（End）。

【实现提示】

需要另外设一个栈，临时停放为给要离开的汽车让路而从停车场退出的汽车，也用顺序存储结构实现。输入数据按照到达或离开的时刻有序。栈中每一个元素表示一辆汽车，包括两个数据项：汽车的拍照号码和进入停车场的时刻。

实验报告第1.3题：停车场管理

实验报告

魏思哲

题目：编制一个实现任意长的整型进行加法运算的演示程序

一、需求分析

1. 本程序中，可以通过用户提交的状态信息，包括，车辆离开还是到达、车辆牌号、时间发生的时间，来模拟停车场的缴费系统
2. 演示程序以用户和计算机对话的方式进行，即在计算机终端上显示提示信息之后，用户在键盘上输入相应的信息；命令执行完毕后，按回车键，用户会得到相应的车辆信息，例如停放位置、或者在停车场内的停留时间。
3. 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)，('D',2,35)，('D',4,40)，('E',0,0)。其中：'A'表示到达（Arrival）；'D'表示离去（Departure）；'E'表示输入结束（End）。

二、概要设计

以栈park模拟停车场，以队列waitline模拟停车场外的便车道，另外设一个栈transferline，来模拟停车场外的临时停车点。

其中，park、transferline 用顺序结构实现、waitline 用链接结构实现。

每个栈（队）内含有构造函数、析构函数、出栈（队）、入栈（队）函数、以及返回栈尾（队首）函数。

此外，还设计了一个结构体，用于存放车辆信息。

三、详细设计

```
struct car_inf                    这是描述车辆信息的结构体
{
   int car_num;                   第一个参数为车牌号码
   int eve_time;                  第二个参数为车辆进入停车场的时间
};

class park                        这是停车场采用的栈的顺序结构的实现
{
      private:
            car_inf *elem;
            int maxsize;
      public:
            int top_p;
```

```
        park(int initSize=10);                  构造函数
        ~park();                                析构函数
        bool isEmpty() const;                   判断是否为空函数
        void push(const car_inf &x);            进栈函数
        car_inf pop(){return elem[top_p--];};   出栈函数

};

park::park(int initSize)
{
    elem=new car_inf[initSize];
    maxsize=initSize;
    top_p=-1;
}

park::~park()
{
    delete[] elem;
}

bool park::isEmpty() const
{
    return top_p==-1;
}

void park::push(const car_inf &x)
{
    elem[++top_p]=x;
}

struct node
{
    car_inf data;
    node *next;
    node(const car_inf &x, node *N=NULL)
    {
        data=x;
        next=N;
    }
    node():next(NULL){}
    ~node(){}
};

class waitline                                  等候的快车道的队的链接实现
```

```
{
    public:
        node *front,*rear;
        waitline();
        ~waitline();
        bool isEmpty() const;
        void enQuene(const car_inf &x);
        car_inf deQuene();
        car_inf getHead()const;
};

waitline::waitline()
{
    front=rear=NULL;
}

waitline::~waitline()
{
    node *tmp;
    while(front!=NULL)
    {
        tmp=front;
        front=front->next;
        delete tmp;
    }
}

bool waitline::isEmpty()const
{
    return front==NULL;
}

void waitline::enQuene(const car_inf &x)
{
    if(rear==NULL)
    {
        front=rear=new node(x);
    }
    else
        rear=rear->next=new node(x);
}

car_inf waitline::deQuene()
{
```

```
    node *tmp=front;
    car_inf value=front->data;
    front=front->next;
    if(front==NULL) rear = NULL;
    delete tmp;
    return value;
}

car_inf waitline::getHead()const
{
    return front->data;
}

class transfer_line
{
    private:
        car_inf *elem;
        int maxsize;
    public:
        int top_p;                                  后面要用到这个指针，所以用公有的了
        transfer_line(int initSize=10);
        ~transfer_line();
        bool isEmpty() const;
        void push(const car_inf &x);
        car_inf pop(){return elem[top_p--];};
};

transfer_line::transfer_line(int initSize)
{
    elem=new car_inf[initSize];
    maxsize=initSize;
    top_p=-1;
}

transfer_line::~transfer_line()
{
    delete[] elem;
}

bool transfer_line::isEmpty() const
{
    return top_p==-1;
}
```

```
void transfer_line::push(const car_inf &x)
{
    elem[++top_p]=x;
}

int main()
{
    bool flag=true;//判断是否继续要求用户输入车辆信息，执行循环结构
    char car_condition;          //记录车辆到达或者离开或者结束时间
    car_inf arrcar;
    int car_number,happentime;    //记录车牌号信息与相应的事件发生时间
    int parksize=2;   //停车场的车位数
    park ori_park(parksize);
    waitline ori_waitline;
    transfer_line ori_transferline(parksize);
    int amont_of_waitline=0;//记录便车道还有几辆车
    while(flag)
    {
        cout<<"请输入此时的状态信息"<<endl;
        cin>>car_condition;
        cout<<"请输入车牌照号码"<<endl;
        cin>>car_number;
        cout<<"请输入此时的时间"<<endl;
        cin>>happentime;
        if(car_condition=='A')
        {
            arrcar.car_num=car_number;
            arrcar.eve_time=happentime;
            if(ori_park.top_p==parksize-1)   //停车场已满
            {
                amont_of_waitline++;
                ori_waitline.enQuene(arrcar);
                cout<<"Sorry,the parking space is full,your car will be waiting at the driveway, the number is:"<<amont_of_waitline<<endl;//此处记得有个缺点，还没写便车道上面的停车顺序
            }
            else
            {
                ori_park.push(arrcar);
                cout<<"Your car has been parked in the park, and the position of it is: "<<ori_park.top_p+1<<endl;
            }
        }
        else if (car_condition=='D')
```

```
{
    car_inf leaving_car;
    while(true)
    {
        car_inf trans_car;
        trans_car=ori_park.pop();
        if(trans_car.car_num==car_number)
        {
            leaving_car=trans_car;
            break;
        }
        else
            ori_transferline.push(trans_car);
    }
    while(ori_transferline.top_p!=-1)
    {
        ori_park.push(ori_transferline.pop());
    }
    if(ori_waitline.isEmpty()!=1)
    {
        car_inf #include <iostream>
```


四、调试分析

1. 算法的时间复杂度分析

每次车辆到达时，会用到进栈或者入队操作，时间复杂度均为 O(1)。

每次车辆离开时，首先会进行一轮查找，找到车辆的位置，然后出栈，其他的车入栈，在从临时停放点出栈，在进入停车场，整个过程的时间复杂度为 O(n).

因此，函数的时间复杂度为 O（n），n 为停车场能够放置的最多车辆数。

2. 算法的空间复杂度分析

最开始创建两个栈，大小均为 n，而队列要根据车辆数来判断其大小。

故算法的空间复杂度为 O（n）。

五、用户手册

1. 本程序使用的 Code::Blocks 10.05 IDE，程序以项目的方式组织，（如图 1）

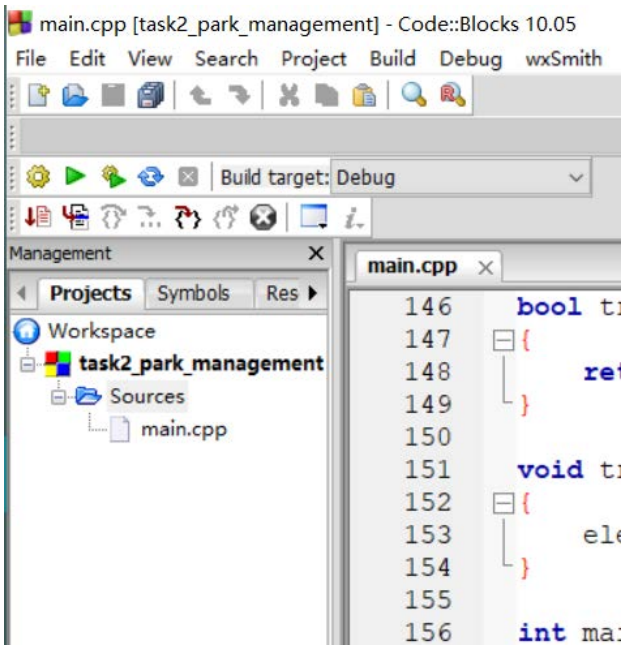

图 1

2. 依次点击菜单“Build”->“Build and run”，显示文本方式的用户界面，（如图 2）

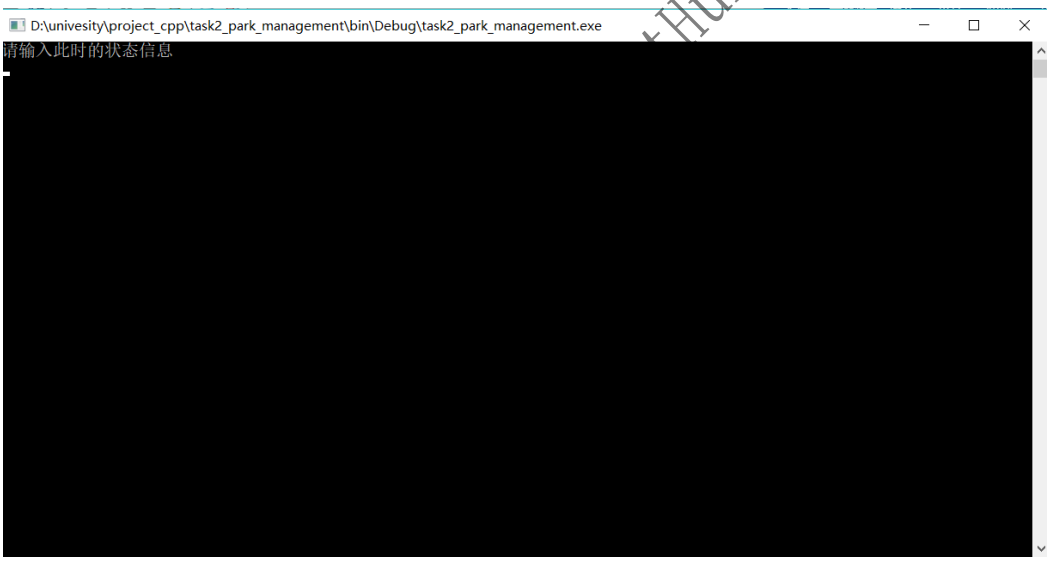

图 2

3.在图 2 的界面中按照说明键入状态信息，按回车，键入第二个信息，以此类推，直到输入的最后一个信息为 E。（如图 3）

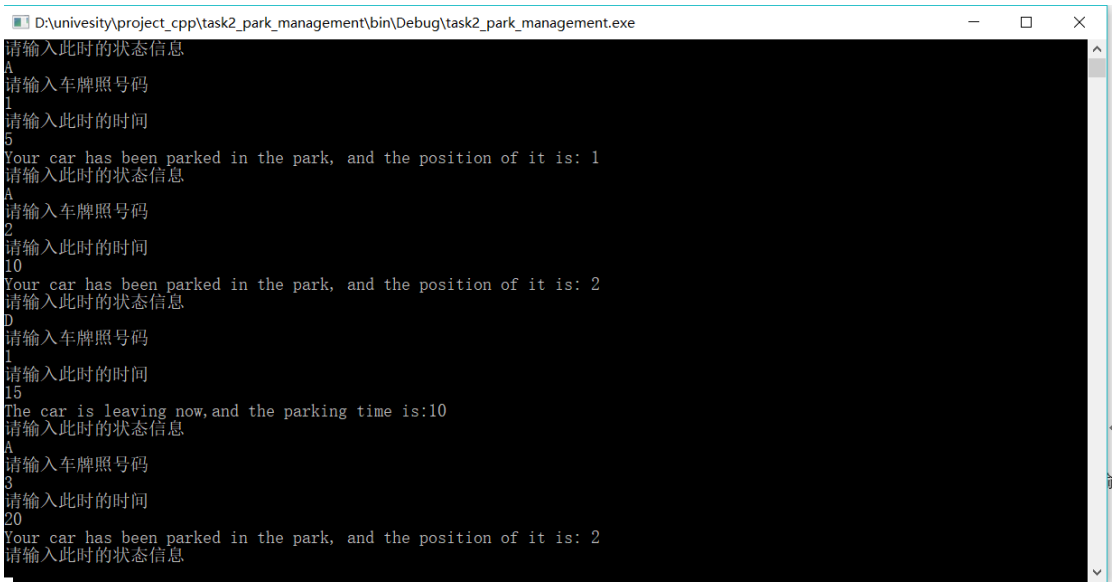

图 3