|  |  |  |  |
| --- | --- | --- | --- |
| **学生学号** | 175064 | **实验课成绩** |  |

*河北工业大学*

**学 生 实 验 报 告 书**

|  |  |
| --- | --- |
| **实验课程名称** | 数据结构 |
| **开 课 学 院** | 国际教育学院 |
| **指导教师姓名** | 宛艳萍 |
| **学 生 姓 名** | 张心宇 |
| **学生专业班级** | 中法计172 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2018 | -- | 2019 | 学年 | 第 | 二 | 学期 |

# 实验内容

【问题描述】

设停车场是一个可以停放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），若车场内已经停满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场为它让路，待该辆车开出大门外，其他车辆再按次序进入车场，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、汽车牌照号码以及到达或离去的时刻。对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’，1，5）， （‘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。

# 实验目的

1. 深入了解栈和队列的特性，掌握栈和队列的存储方法。
2. 掌握栈和队列的基本操作，如初始化、入栈（队列）、出栈（队列）等，并能在实际问题背景下灵活运用。

# 程序清单

#include<stdio.h>  
#include<stdlib.h>  
  
// 汽车信息  
typedef struct{  
 char status; // 记录车'A' 或 ‘D'.  
 int id;  
 int time;  
}Car;  
  
// 栈结构  
typedef struct{  
 int capacity;  
 int top;  
 Car \* array;  
}Stack;  
  
// 队列节点  
typedef struct node{  
 Car car;  
 struct node \* next;  
}node;  
  
// 队列结构  
typedef struct {  
 node \* front;  
 node \* rear;  
 int num; // 记录队列长度  
}Queue;  
  
  
// 初始化车辆信息  
Car\* create\_car(char s,int id, int time){  
 Car \* car = (Car \*)malloc(sizeof(Car));  
 car->id = id;  
 car->time = time;  
 car->status = s;  
 return car;  
}  
  
  
Stack\* create\_stack(int n){  
 Stack \* s = (Stack\*)malloc(sizeof(Stack));  
 if(!s)exit(-1);  
 s->array = (Car\*)malloc(sizeof(Car) \* n);  
 if(!s->array)exit(-1);  
 s->capacity = n;  
 s->top = 1;  
 return s;  
}  
  
  
// 初始化呆头结点的队列  
Queue\* create\_queue(){  
 Queue \* q = (Queue\*)malloc(sizeof(Queue));  
 if(!q)exit(-1);  
 q->front=q->rear=(node\*)malloc(sizeof(node)); // 创建头结点  
 if(!q->front)exit(-1);  
 q->front->next = NULL;  
 q->num = 0;  
 return q;  
}  
  
  
// 1 空， 0 非空  
int stack\_is\_empty(Stack \*s){  
 if(s->top==1)return 1;  
 return 0;  
}  
int queue\_is\_empty(Queue \*q){  
 return q->front->next==NULL;  
}  
  
// 1 满， 0 不满  
int stack\_is\_full(Stack \*s){  
 if(s->top==s->capacity+1)return 1;  
 return 0;  
}  
  
int push(Car \*c, Stack \*s){  
 if(stack\_is\_full(s))exit(-1);  
 s->array[s->top++] = \*c;  
 return 0;  
}  
  
Car\* pop(Stack \*s){  
 if(stack\_is\_empty(s))exit(-1);  
 s->top--;  
 return &s->array[s->top];  
}  
  
// 入队  
int enqueue(Queue \*q, Car \*c){  
 node \* p = (node\*)malloc(sizeof(node));  
 if(!p)exit(-1);  
 p->car=\*c;  
 p->next=NULL;  
 q->rear->next=p;  
 q->rear=p;  
 q->num++;  
 return 0;  
}  
  
// 出队  
Car dequeue(Queue \*q){  
 if(queue\_is\_empty(q))exit(-1); // 如果保证操作队列绝对不空， 这条语句必要吗？  
 Car c = q->front->next->car; // 如果这里使用指针，free以后会不会丢失内存  
 node \* p = q->front;  
 q->front=q->front->next;  
 free(p);  
 q->num--;  
 return c; // 如果是返回指针， 函数调用后， 指针所指向的内存释放。  
}  
  
// 车辆离开车库  
Car out\_car(Car \*c, Stack \*s, Stack \*tmp){  
 Car \* car, result;  
 while((car=pop(s))->id!=c->id){  
 push(car, tmp);  
 }  
 result = \*car;  
 while(!stack\_is\_empty(tmp)){  
 push(pop(tmp), s);  
 }  
 return result;  
}  
  
// 停车时间  
int parking\_time(Car a, Car\*d){  
 return (d->time)-(a.time);  
}  
  
// 收费, n:每分钟费用  
int charge(int parking\_time, int n){  
 return n\*parking\_time;  
}  
  
// 停车场函数  
int parking(){  
 // 初始化两个栈  
 int n; // 停车场大小  
 printf("请输入停车场大小：");  
 scanf("%d", &n);  
 Stack \* park = create\_stack(n); // 停车场栈  
 Stack \* tmp = create\_stack(n-1); // 倒车时的暂存栈  
  
 // 初始化便道  
 Queue \* road = create\_queue();  
  
 Car \* car = NULL;  
 int flag = 1; // 控制循环  
 int i = 1; // 记录循环次数  
 while(flag){  
 // 初始化车辆信息  
 if(!car){  
 char status;  
 int id;  
 int time;  
 printf("请输入车辆%d信息：", i);  
 i++;  
 scanf(" %c,%d,%d", &status, &id, &time);  
 car = create\_car(status, id, time);  
 }  
  
 switch(car->status){  
 case 'A': if(!stack\_is\_full(park)){  
 push(car, park);  
 printf("%d号车辆位于停车场的%d号位。\n", car->id, park->top-1); // 这样是不是破坏了封装性  
 }else{  
 enqueue(road, car);  
 printf("%d号车辆位于便道的%d号位\n", car->id, road->num); // 长整型  
 }  
 car = NULL; break;  
 case 'D': if(!queue\_is\_empty(road)){  
 Car c = dequeue(road);  
 c.status = 'A';  
 c.time = car->time;  
  
 int time = parking\_time(out\_car(car, park, tmp), car);  
 int money = charge(time, 1);  
 printf("%d号车在车场内停车%d分钟，应交费%d元。\n", car->id, time, money);  
 car = &c;  
 }else{  
 int time = parking\_time(out\_car(car, park, tmp), car);  
 int money = charge(time, 1);  
 printf("%d号车在车场内停车%d分钟，应交费%d元。\n", car->id, time, money);  
 car = NULL;  
 }  
 break;  
 case 'E': flag = 0; break; // 结束停车场函数循  
 }  
 }  
  
}  
  
int main(){  
 parking();  
 return 0;  
}

# 运行结果



# 分析与思考

1. 根据实际问题选择结构的实现方式（顺序还是链表）