Experiment topic: Design and Implementation of Parking Lot Management System

1. Experimental content

project 2 parking lot management system (8 credit hours)

1. Description of the problem: There is only one narrow passageway for n cars in the parking lot, and there is only one gate for cars to enter and exit. Cars are arranged in the parking lot in the order of their arrival time from north to south (the gate is at the southernmost end, and the first car that arrives first is parked at the northernmost end of the parking lot), if the parking lot is full of n cars , then the later cars can only wait on the sidewalk outside the door. Once a car drives away, the first car on the sidewalk can drive in; when a car in the parking lot is about to leave, the car behind it The entering vehicle must first exit the parking lot to make way for it. After the vehicle drives out of the gate, other vehicles enter the parking lot in the original order. Each car parked in the parking lot must be based on the length of time it stays Pay the fee. Try to compile a simulation program for the parking lot that is managed according to the above requirements.

2. basic requirements

(1) The parking lot is simulated by the stack, the sidewalk outside the parking lot is simulated by the queue, and the simulation management is carried out according to the input data sequence read from the terminal.

(2) Each set of input data includes three data items: the "arrival" or "departure" information of the car, the license plate number of the car, and the time of arrival or departure. The output data after operating each set of input data is: If the vehicle arrives, output the parking position of the car in the parking lot or on the sidewalk; if the car leaves , then output the time the car stays in the parking lot and the fee to be paid (the time spent on the sidewalk is not charged).

(3) The stack is implemented in a sequential structure, and the queue is implemented in a linked list structure .

(4) Design independently according to the requirements of the topic, and write a design report as required after the design is completed .

Experiment ideas:

1、Using stacks and queues

（1）Prepare stack and queue related functions. Build two sequential stacks and a linked queue. One of the sequential stacks is used to simulate the parking lot, and the other stack is used to store the outer cars first when there are inner cars in the parking lot that need to go out, and then pour out the stack and store them back in the parking lot after the inner cars go out .

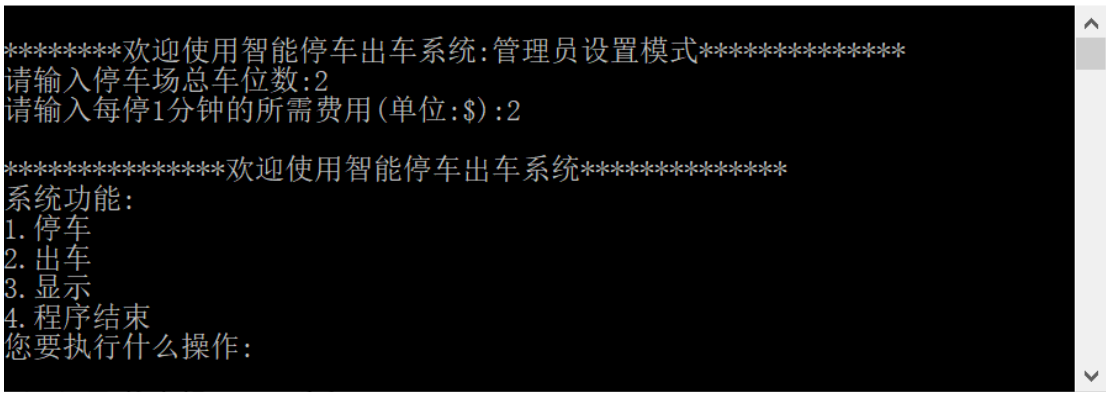
The queue is used to simulate the sidewalk. When the parking lot is full, the car that arrives here first stops at the front of the queue, and so on. If there is a car coming out of the parking lot, after the two stacks are processed, the car at the front of the queue will be stored in the parking lot (first

in, first out).

2、other

（1）Design the user interface and simulate the operation process. Design parking and exit functions. Design the display function to display the parking situation of the parking lot and sidewalk to judge whether the parking is successful or not.

（2）Design a friendly user interface. The user interface setting allows the user to input the number of parking lots (sequential stack length) and the parking unit price. When leaving the car, the payable amount will be given according to the parking time and unit price.



3. Experiment code

Two versions are given below: (divided into single-file version and multi-file version)

Code (single-file version):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define OK 1
#define ERROR 0
#define MAXQSIZE 5
#define OVERFLOW 0

struct Element
{
    int locate;
    char plate[10];
    structure
    {
        int entime1,entime2;
        int outtime1, outtime2;
    } time;
};
```

```
//******************Stack********************//

typedef struct
{
	Element *base;
	Element *top;
	int stacksize;
	int num;
}SqStack;

int InitStack(SqStack &S, int MAXSIZE)
{ // Construct an empty stack
	S.base=new Element[MAXSIZE]; //Allocate an array space with a maximum capacity of MAXSIZE for the sequential stack
	if(!S.base)
		exit(OVERFLOW); //Storage allocation failed
	S.top=S.base; //top is initially base, empty stack
	S.stacksize=MAXSIZE; //stacksize is set to the maximum capacity of the stack MAXSIZE
	S.num=0;
	return OK;
}

int Push(SqStack &S,Element e)
{ // Insert element e as the new top element of the stack
	if(S.top-S.base==S.stacksize)
		return ERROR; //The stack is full
	*S.top++=e; //Element e is pushed onto the top of the stack, and the top pointer of the stack is incremented by 1
	S.num++;
	return OK;
}

int Pop(SqStack &S,Element &e)
{ //Delete the top element of S, return its value with e
	if(S.top==S.base)
		return ERROR; //Stack is empty
	e=*--S.top; //The top pointer of the stack is decremented by 1, and the top element of the stack is assigned to e
	S.num--;
	return OK;
}

int EmptyStack(SqStack S)
```

```
{
	if(S.top-S.base==0)
		return 1;
	else
		return 0;
}
int FullStack(SqStack S, int MAXSIZE)
{
	if(S.top-S.base==MAXSIZE)
		return 1;
	else
		return 0;
}

Element GetTop(SqStack S)
{ //Return the top element of S, without modifying the top pointer
	if(S.top!=S.base) //Stack is not empty
		return *(S.top-1); //Return the value of the top element of the stack, the top pointer of
the stack remains unchanged
}


//********************Chain Queue************************//
typedef struct QNode
{
	Element data;
	struct QNode *next;
}QNode,*QueuePtr;
typedef struct
{
	QueuePtr front;
	QueuePtr rear;
	int num;
}LinkQueue;

int InitQueue(LinkQueue &Q)
{
	Q.front=Q.rear=new QNode;
	Q. front->next=NULL;
	Q.num=0;
	return OK;
}

int EnQueue(LinkQueue &Q,Element e)
```