



武汉纺织大学
WUHAN TEXTILE UNIVERSITY

数学与计算机学院

停车场管理系统设计报告

杨鑫

计算机 11902 班

2021 年 2 月 9 日

本作品采用 CC-BY-NC-SA 协议进行许可



1 需求分析

1.1 问题描述

停车场内只有一个可停放 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停车场的最北端），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，待该辆车开出大门外，其它车辆再按原次序进入车场，每辆停放在车场的车在它离开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。

1.2 基本要求

- (1) 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；
- (2) 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：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）；
- (3) 栈以顺序结构实现，队列以链表实现。

2 概要设计

2.1 数据结构

2.1.1 时间类

时间是进行收费的依据，此处假定时间在一天之内。时间类既可以表示一个时刻，也可以表示一段时间长度。

时间类数据成员：

- **hour**: 指示时钟
- **minute**: 指示分钟

时间类成员函数：

- **Time**: 构造函数
- **operator+=**: 进行时间的复合加法，即将相加后的结果赋给当前对象，并返回当前对象
- **operator-=**: 进行时间的复合减法，即将相减后的结果赋给当前对象，并返回当前对象

Time
+ hour: int
+ minute: int
+ Time()
+ Time(hour: int, minute: int)
<<friend>> operator+(left: const Time&, right: const Time&): Time
+ operator+=(right: const Time&): Time
<<friend>> operator-(left: const Time&, right: const Time&): Time
+ operator-=(right: const Time&): Time
<<friend>> operator==(left: const Time&, right: const Time&): bool
<<friend>> operator<(left: const Time&, right: const Time&): bool
<<friend>> operator<=(left: const Time&, right: const Time&): bool
<<friend>> operator>(left: const Time&, right: const Time&): bool
<<friend>> operator>=(left: const Time&, right: const Time&): bool
<<friend>> operator<<(os: ostream&, t: const Time&): ostream&

时间类非成员函数：

- **operator+:** 时间加法
- **operator-:** 时间减法
- **operator==:** 判断时间是否相等
- **operator<:** 判断前一个时间是否小于后一个时间
- **operator<=:** 判断前一个时间是否小于等于后一个时间
- **operator>:** 判断前一个时间是否大于后一个时间
- **operator>=:** 判断前一个时间是否大于等于后一个时间
- **operator<<:** 将时间输出到输出流中

2.1.2 汽车类

汽车是后续操作的基础元素，结合题目要求，汽车应包含牌照号码、停车时间等数据。

汽车类数据成员：

- **number:** 汽车牌照号码
- **lastTime:** 最后一次在停车场（栈）内的时刻
- **parking:** 停车时间

lastTime 和 parking 并非实时更新，而是当有车辆到达或离去时才会做相应的改变。

Car
<pre> - number: string - lastTime: Time - parking: Time + Car() + Car(number: string) + operation==(car: const Car&): bool + updateLastTime(time: const Time&) + setLastTime(time: const Time&) + getLastTime(): Time + addParkingTime(time: const Time&) <<friend>> operator<<(os: ostream&, t: const Car&): ostream& </pre>

汽车类成员函数：

- Car: 构造函数
- operation==: 判断是否为同一辆汽车（只判断牌照号码）
- updateLastTime: 设置最后一次在停车场（栈）内的时刻，并增加停车时间
- setLastTime: 设置最后一次在停车场（栈）内的时刻
- getLastTime: 获取最后一次在停车场（栈）内的时刻
- addParkingTime: 增加停车时间
- getParingTime: 获取停车时间

汽车类非成员函数：

- operator<<: 将汽车信息输出到输出流中

2.1.3 栈类模板

栈是一种“先进后出”的数据结构，此处用于组织停车场内汽车的停放。

我们选用线性结构实现栈，并进行模板化，以便下次复用。

模板参数 T^1 ，表示基础元素类型，在编写代码时，可设为 Car。

¹从模板参数 T，定义出以下几种嵌套类型：

- $T \rightarrow value_type$ ，基础元素类型
- $T* \rightarrow pointer$ ，指针
- $const T* \rightarrow const_pointer$ ，常指针
- $T\& \rightarrow reference$ ，引用
- $const T\& \rightarrow const_reference$ ，常引用

Stack<T>
- _top: pointer - _base: pointer - _cap: pointer
+ Stack() ~ Stack() + operator[] (pos: size_type): reference + operator[] (pos: size_type): const_reference + top(): reference + top(): const_reference + empty(): bool + size(): size_type + capacity(): size_type + push(data: const_reference) + pop() + index(data: const_reference): size_type + print() + reserve(newCapacity: size_type)

1

栈类数据成员：

- _top 栈顶
- _base 栈底
- _cap 当前栈存储空间的尾部

特别指出，栈顶指针 _top，其初值指向栈底，即 _top == _base 可作为栈空的标记，每当插入新的栈顶元素时，指针 _top 增 1；删除栈顶元素时，指针 _top 减 1，因此，非空栈中的栈顶指针始终在栈顶元素的下一个位置上。

_cap 指针则仅指示当前栈存储空间的尾部，其真实地址则是尾部的下一个位置。

这两种设计都是为了方便程序的编写。

栈类成员函数：

- Stack: 构造函数
- ~Stack: 析构函数
- operator[]: 访问器²
- top: 访问栈顶

¹size_type 是无符号超长整型，源于 std::size_t。

²设置访问器仅用于方便本次项目的代码编写，破坏了栈的设计原则，在编写一般程序时，不应在栈中设置访问器。

- `empty`: 判断栈是否为空
- `size`: 获取栈长
- `capacity`: 获取栈的存储容量
- `push`: 入栈
- `pop`: 出栈
- `index`: 寻找元素的第一个位置
- `print`: 打印栈
- `reserve`: 重新分配存储空间容量

2.1.4 队列类模板

队列，是一种“先进先出”的数据结构，此处可用于模拟停车场外的便道。我们选用链式结构实现队列，并进行模板化，以便下次复用。模板参数 T^1 ，表示基础元素类型，在编写代码时，可设为 `Car`。首先设计一个队列节点类，与链表节点类似：

QNode<T>
- <code>_data</code> : <code>value_type</code> - <code>_next</code> : <code>node_ptr</code>
+ <code>QNode()</code> + <code>QNode(data: const_reference, next = nullptr: const_node_ptr)</code> + <code>data(): reference</code> + <code>data(): const_reference</code> + <code>setData(data: const_reference)</code> + <code>next(): node_ptr</code> + <code>next(): const_node_ptr</code> + <code>setNext(next: const_node_ptr)</code>

队列节点类数据成员：

- `_data`: 数据域

¹从模板参数 T 定义出的嵌套类型与 `Stack` 类相似，但增加了 4 个嵌套定义：

- `QNode < T > * \longrightarrow node_ptr`，节点指针
- `const QNode < T > * \longrightarrow const_node_ptr`，节点常指针
- `QNode < T > & \longrightarrow node_ref`，节点引用
- `const QNode < T > & \longrightarrow const_node_ref`，节点常引用

- `_next`: 指针域, 指向下一节点

队列节点类成员函数:

- `QNode`: 构造函数
- `data`: 返回数据域
- `setData`: 设置数据域
- `next`: 返回下一节点
- `setNext`: 设置下一节点

然后以队列节点为基础设计队列类模板:

Queue<T>
- <code>_front: node_ptr</code>
- <code>_back: node_ptr</code>
+ <code>Queue()</code>
~ <code>Queue()</code>
+ <code>front(): reference</code>
+ <code>front(): const_reference</code>
+ <code>back(): reference</code>
+ <code>back(): const_reference</code>
+ <code>empty(): bool</code>
+ <code>size(): size_type</code>
+ <code>push(data: const_reference)</code>
+ <code>pop()</code>
+ <code>erase(data: const_reference)</code>
+ <code>print()</code>

队列类数据成员:

- `_front`: 队头指针
- `_back`: 队尾指针

队列类成员函数:

- `Queue`: 构造函数
- ~ `Queue`: 析构函数
- `front`: 返回队头
- `back`: 返回队尾

- **empty**: 返回队列是否为空
- **size**: 返回队列的大小
- **push**: 入队列
- **pop**: 出队列
- **erase**: 直接擦除第一个值为 **data** 的节点¹
- **print**: 打印队列

2.1.5 停车场类

依据题意，整个停车场包含一个有限容量的停车场（栈）和场外的便道（队列）。将栈和队列综合，封装成一个停车场类。

ParkLot
- s: Stack<Car> - q: Queue<Car> - size: size_t
+ ParkLot() + ParkLot(size: size_t) + add(carNumber: const string&, arriveTime: const Time&) + reduce(carNumber: const string&, leaveTime: const Time&) + print()

停车场类数据成员：

- **s**: 停车场（栈）
- **q**: 便道（队列）
- **size**: 停车场容量（栈最大大小）

停车场类函数成员：

- **ParkLot**: 构造函数
- **add**: 到达汽车
- **reduce**: 离开汽车
- **print**: 打印停车场

¹设计此函数仅用于方便本项目代码的编写，破坏了队列的设计原则，在编写一般程序时，不应在队列中设置此函数。

2.2 程序模块

绝大部分算法均已经在各数据结构中实现。

下面设计车辆收费算法。

假定：

车辆仅收取在停车场（栈）上停留的费用，在便道（队列）上停留的时间不收费。

- 1、 停留时间小于 1 小时，不收费；
- 2、 停留时间在 1 小时以上，2 小时以下，收费 5 元；
- 3、 停留时间在 2 小时以上，2 小时以上的部分，每小时收取 3 元（不满 1 小时，按 1 小时计算）。

例如：

停留半小时，不收费；

停留 1 个半小时，收费 5 元；

停留 4 个小时，收费 11 元。

3 详细设计

3.1 类的函数实现

3.1.1 时间类

时间类的函数实现如代码1所示。

```
1 struct Time {
2     /****** 公有数据成员 *****/
3     int hour;    // 时
4     int minute; // 分
5
6     // 构造函数
7     Time() : Time(0, 0) {}
8
9     Time(int hour, int minute) {
10         this->hour = hour;
11         this->minute = minute;
12     }
13
14     friend Time operator+(const Time &left, const Time &right);
15     friend Time operator-(const Time &left, const Time &right);
16     friend bool operator==(const Time &left, const Time &right);
```

```

17     friend bool operator<(const Time &left, const Time &right);
18     friend bool operator<=(const Time &left, const Time &right);
19     friend bool operator>(const Time &left, const Time &right);
20     friend bool operator>=(const Time &left, const Time &right);
21     friend std::ostream &operator<<(std::ostream &os, const Time
        &t);
22
23     Time operator+=(const Time &right) {
24         *this = *this + right;
25         return *this;
26     }
27
28     Time operator-=(const Time &right) {
29         *this = *this - right;
30         return *this;
31     }
32 };
33
34 Time operator+(const Time &left, const Time &right) {
35     int m = (left.minute + right.minute) % 60;
36     int h = left.hour + right.hour + (left.minute + right.minute)
        / 60;
37     return Time{h, m};
38 }
39
40 Time operator-(const Time &left, const Time &right) {
41     if (left.hour < right.hour ||
42         left.hour == right.hour && left.minute < right.minute)
43         return Time{0, 0};
44
45     int h = left.hour - right.hour;
46     int m = left.minute - right.minute;
47     if (m < 0) {
48         m += 60;
49         h--;
50     }
51     return Time(h, m);
52 }
53

```

```

54 bool operator==(const Time &left, const Time &right) {
55     return left.hour == right.hour && left.minute == right.minute
56         ;
57 }
58 bool operator<(const Time &left, const Time &right) {
59     if (left.hour < right.hour)
60         return true;
61     else if (left.hour == right.hour) {
62         return left.minute < right.minute;
63     } else
64         return false;
65 }
66
67 bool operator<=(const Time &left, const Time &right) {
68     return left == right || left < right;
69 }
70
71 bool operator>(const Time &left, const Time &right) {
72     return !(left <= right);
73 }
74
75 bool operator>=(const Time &left, const Time &right) {
76     return !(left < right);
77 }
78
79 // 将时间输出到流中
80 std::ostream &operator<<(std::ostream &os, const Time &t) {
81     os << t.hour << "h " << t.minute << "m";
82     return os;
83 }

```

代码 1: Time 类的实现

3.1.2 汽车类

汽车类的函数实现如代码2所示。

```

1 class Car {
2     private:
3         /***** 私有数据成员 *****/

```

```
4         std::string number; // 牌照号码
5         Time lastTime;      // 最后一次在栈内的时刻
6         Time parking;       // 停车时间
7
8     public:
9         // 构造函数
10        Car() = default;
11
12        explicit Car(std::string number) {
13            this->number = number;
14            this->parking = {0, 0};
15        }
16
17        // 判断是否为同一辆车
18        bool operator==(const Car &car) {
19            return this->number == car.number;
20        }
21
22        // 设置最后一次在栈内的时刻,并增加停车时间
23        void updateLastTime(const Time &time) {
24            addParkingTime(time - lastTime);
25            this->lastTime = time;
26        }
27
28        // 设置最后一次在栈内的时刻
29        void setLastTime(const Time &time) {
30            this->lastTime = time;
31        }
32
33        // 获取最后一次在栈内的时刻
34        Time getLastTime() {
35            return lastTime;
36        }
37
38        // 增加停车时间
39        void addParkingTime(const Time &time) {
40            this->parking += time;
41        }
42
```

```

43         // 获取停车时间
44         Time getParingTime() {
45             return parking;
46         }
47
48         friend std::ostream &operator<<(std::ostream &os, const
49             Car &c);
50
51         // 将汽车输出到流中
52         std::ostream &operator<<(std::ostream &os, const Car &c) {
53             os << c.number;
54             return os;
55         }

```

代码 2: Car 类的实现

3.1.3 栈类模板

栈类模板的函数实现如代码3所示。

```

1  template<typename T>
2  class Stack {
3  public:
4      /******* 嵌套类型 *****/
5      typedef T          value_type;          // 基础元素类型
6      typedef T*         pointer;             // 指针
7      typedef const T*   const_pointer;       // 常指针
8      typedef T&         reference;           // 引用
9      typedef const T&   const_reference;     // 常引用
10     typedef std::size_t size_type;           // 无符号长整型
11
12 private:
13     /******* 私有数据成员 *****/
14     pointer _top;    // 栈顶(后一位)
15     pointer _base;   // 栈底
16     pointer _cap;    // 当前栈存储空间的尾部(后一位)
17
18 public:
19     // 构造一个空栈
20     Stack() {

```

```
21     try {
22         _base = new value_type[Stack_Init_Size];
23         _top = _base;
24         _cap = _base + Stack_Init_Size;
25     } catch (std::bad_alloc &) { // 分配内存异常
26         std::cerr << "Exception: Can not construct a new
27             Stack object,"
28             << "because there is no available memory.
29             "
30             << std::endl;
31     }
32 }
33
34 // 销毁栈
35 ~Stack() {
36     delete[] _base;
37     _base = _top = _cap = nullptr;
38 }
39
40 // 返回pos位置的引用
41 reference operator[](size_type pos) {
42     assert(!empty());
43     return *(_base + pos);
44 }
45
46 // 返回pos位置的常引用
47 const_reference operator[](size_type pos) const {
48     assert(!empty());
49     return *(_base + pos);
50 }
51
52 // 返回栈顶的引用
53 reference top() {
54     assert(!empty());
55     return *(_top - 1);
56 }
57
58 // 返回栈顶的常引用
59 const_reference top() const {
```

```
58         assert(!empty());
59         return *(_top - 1);
60     }
61
62     // 返回栈是否为空
63     bool empty() const noexcept {
64         return _top == _base;
65     }
66
67     // 获取栈长
68     size_type size() const noexcept {
69         return (_top - _base);
70     }
71
72     // 获取栈的存储容量
73     size_type capacity() const noexcept {
74         return _cap - _base;
75     }
76
77     // 入栈
78     void push(const_reference data) {
79         if (size() >= capacity())
80             try {
81                 reserve(capacity() * 3 / 2);
82             } catch (std::bad_alloc &) {
83                 std::cerr << "Exception: Can not push new data,"
84                     << "because there is no available
85                     memory."
86                     << std::endl;
87             }
88         *(_top++) = data;
89     }
90
91     // 出栈
92     void pop() {
93         assert(!empty());
94         _top--;
95     }
```

```
96 // 寻找元素的第一个位置
97 size_type index(const_reference data) {
98     for (size_type i = 0; i < size(); i++) {
99         if (*(_base + i) == data)
100             return i;
101     }
102     throw std::out_of_range{"not found"};
103 }
104
105 // 打印栈
106 void print() {
107     if (empty())
108         std::cout << "null" << std::endl;
109     else {
110         for (size_type i = 0; i < size(); i++) {
111             std::string ch = ((i + 1) % 5 ? " " : "\n");
112             if (i + 1 == size())
113                 ch = "\n";
114             std::cout << "(" << i + 1 << ", " << *(_base + i)
115                 << ")" << ch;
116         }
117         std::cout << std::endl;
118     }
119 }
120
121 // 重新分配存储空间容量
122 void reserve(size_type newCapacity) {
123     size_type _size = size();
124
125     if (newCapacity <= capacity())
126         return;
127
128     pointer temp = _base;
129     _base = new value_type[newCapacity];
130     for (int i = 0; i < _size; i++)
131         *(_base + i) = *(temp + i);
132     delete[] temp;
133
134     _top = _base + _size;
```



```

134         _cap = _base + newCapacity;
135     }
136 };

```

代码 3: Stack 类的实现

3.1.4 队列类模板

队列节点类模板的函数实现如代码4所示。

```

1  template<typename T>
2  class QNode {
3  public:
4      /******* 嵌套类型 *****/
5      typedef T                      value_type;           // 基础元素类
6          型
7      typedef T*                      pointer;             // 指针
8      typedef const T*                const_pointer;       // 常指针
9      typedef T&                      reference;           // 引用
10     typedef const T&                 const_reference;     // 常引用
11     typedef QNode<T>*                node_ptr;           // 节点指针
12     typedef const QNode<T>*          const_node_ptr;     // 节点常指针
13     typedef QNode<T>&                node_ref;           // 节点引用
14     typedef const QNode<T>&          const_node_ref;     // 节点常引用
15     typedef std::size_t              size_type;         // 无符号超长
16          整型
17 private:
18     /******* 私有数据成员 *****/
19     value_type _data; // 数据域
20     node_ptr _next;  // 指针域, 指向下一节点
21 public:
22     // 默认构造函数
23     QNode() : QNode{value_type()} {}
24
25     // 有参构造函数
26     explicit QNode(const_reference data, const_node_ptr next =
27         nullptr) {
28         this->_data = data;
29         this->_next = const_cast<node_ptr>(next);

```

```

29     }
30
31     // 返回数据域的引用
32     reference data() {
33         return _data;
34     }
35
36     // 返回数据域的常引用
37     const_reference data() const {
38         return _data;
39     }
40
41     // 修改数据域
42     void setData(const_reference data) {
43         this->_data = data;
44     }
45
46     // 返回指针域
47     node_ptr next() {
48         return this->_next;
49     }
50
51     // 返回指针域
52     const_node_ptr next() const {
53         return this->_next;
54     }
55
56     // 修改指针域
57     void setNext(const_node_ptr next) {
58         this->_next = const_cast<node_ptr>(next);
59     }
60 };

```

代码 4: QNode<T> 类模板的实现

队列类模板的函数实现如代码5所示。

```

1  template<typename T>
2  class Queue {
3  public:
4      /**/ 嵌套类型  ***/

```

```

5     typedef typename QNode<T>::value_type           value_type;
        // 基础元素类型
6     typedef typename QNode<T>::reference           reference;
        // 节点引用
7     typedef typename QNode<T>::const_reference
        const_reference;    // 节点常引用
8     typedef typename QNode<T>::node_ptr            node_ptr;
        // 节点指针
9     typedef typename QNode<T>::const_node_ptr
        const_node_ptr;    // 节点常指针
10    typedef typename QNode<T>::node_ref             node_ref;
        // 节点引用
11    typedef typename QNode<T>::const_node_ref
        const_node_ref;    // 节点引用
12    typedef std::size_t                             size_type;
        // 无符号超长整型
13
14
15 private:
16     /******* 私有数据成员 *****/
17     node_ptr _front; // 队头指针
18     node_ptr _back;  // 队尾指针
19
20 public:
21     // 构造一个空队列
22     Queue() {
23         try {
24             _front = _back = new QNode<value_type>;
25         } catch (std::bad_alloc &) {
26             std::cerr << "Exception: Can not construct a new
                Stack object,"
27                         << "because there is no available memory.
                "
28                         << std::endl;
29         }
30         _front->setNext(nullptr);
31     }
32
33     // 销毁队列

```

```
34 ~Queue() {
35     while (_front) {
36         _back = _front->next();
37         delete _front;
38         _front = _back;
39     }
40 }
41
42 // 返回队头的引用
43 reference front() {
44     return _front->next()->data();
45 }
46
47 // 返回队头的常引用
48 const_reference front() const {
49     return _front->next()->data();
50 }
51
52 // 返回队尾的引用
53 reference back() {
54     return _back->data();
55 }
56
57 // 返回队尾的常引用
58 const_reference back() const {
59     return _back->data();
60 }
61
62 // 返回队列是否为空
63 bool empty() const noexcept {
64     return _front == _back;
65 }
66
67 // 返回队列的大小
68 size_type size() {
69     node_ptr temp = _front->next();
70     size_type size{0};
71     while (temp) {
72         size++;
```

```
73         temp = temp->next();
74     }
75     return size;
76 }
77
78 // 入队列
79 void push(const_reference data) {
80     node_ptr p;
81     try {
82         p = new QNode<value_type>;
83     } catch (std::bad_alloc &) {
84         std::cerr << "Exception: Can not push new data,"
85                     << "because there is no available memory."
86                     << std::endl;
87     }
88     p->setData(data);
89     p->setNext(nullptr);
90     _back->setNext(p);
91     _back = p;
92 }
93
94 // 出队列
95 void pop() {
96     assert(!empty());
97     node_ptr p = _front->next();
98     _front->setNext(p->next());
99     if (_back == p)
100         _back = _front;
101     delete p;
102 }
103
104 // 直接擦除第一个值为 data 的节点
105 void erase(const_reference data) {
106     assert(!empty());
107     node_ptr before = _front;
108     node_ptr current = _front->next();
109     while (current) {
110         auto next = current->next();
```

```

111         if (current->data() == data) {
112             auto toDelete = current;
113             if (_back == current)
114                 _back = before;
115             current = next;
116             before->setNext(next);
117             delete toDelete;
118             return;
119         } else {
120             before = current;
121             current = next;
122         }
123     }
124     throw std::out_of_range("not found");
125 }
126
127 // 打印队列
128 void print() {
129     if (empty()) {
130         std::cout << "null" << std::endl;
131     } else {
132         node_ptr p = _front->next();
133         auto cnt{1};
134         while (p) {
135             std::string ch = (cnt % 5 ? " " : "\n");
136             if (cnt == size())
137                 ch = "\n";
138             std::cout << "(" << cnt++ << ", " << p->data() <<
139                 ")" << ch;
140             p = p->next();
141         }
142     }
143 };

```

代码 5: Queue<T> 类模板的实现

3.1.5 停车场类

停车场类的函数实现如代码6所示。

```

1  class ParkLot {
2  private:
3      /******* 私有数据成员 *****/
4      Stack<Car> s;          // 停车场(栈)
5      Queue<Car> q;         // 便道(队列)
6      std::size_t size{};   // 停车场容量(栈最大大小)
7
8  public:
9      // 构造函数
10     ParkLot() = default;
11
12     explicit ParkLot(std::size_t size) {
13         this->size = size;
14     }
15
16     // 到达汽车
17     void add(const std::string &carNumber, const Time &arriveTime
18             ) {
19         if (s.size() >= size) {
20             Car c = Car{carNumber};
21             q.push(c);
22             std::cout << "The car arrive at No." << q.size() << "
23                 position in corridor." << std::endl;
24         } else {
25             Car c = Car{carNumber};
26             c.setLastTime(arriveTime);
27             s.push(c);
28             std::cout << "The car arrive at No." << s.size() << "
29                 position in parking lot." << std::endl;
30         }
31     }
32
33     // 离开汽车
34     void reduce(const std::string &carNumber, const Time &
35         leaveTime) {
36         if (s.size() < size || (s.size() == size && q.empty())) {
37             try {
38                 int pos = s.index(Car{carNumber});
39                 s[pos].updateLastTime(leaveTime);

```

```

36         std::cout << "Parking time: " << s[pos].
           getParingTime() << std::endl;
37         std::cout << "Fees to be paid: " << fees(s[pos])
           << std::endl;
38         for (std::size_t i = pos + 1; i < s.size(); i++)
           {
39             s[i - 1] = s[i];
40         }
41         s.pop();
42     } catch (const std::out_of_range &e) {
43         std::cerr << "No matching Car In Parking Lot." <<
           '\n';
44     }
45 } else {
46     try {
47         int pos = s.index(Car{carNumber});
48         s[pos].updateLastTime(leaveTime);
49         std::cout << "Parking time: " << s[pos].
           getParingTime() << std::endl;
50         std::cout << "Fees to be paid: " << fees(s[pos])
           << std::endl;
51         for (std::size_t i = pos + 1; i < s.size(); i++)
           {
52             s[i - 1] = s[i];
53         }
54         s.pop();
55         Car c = q.front();
56         c.setLastTime(leaveTime);
57         q.pop();
58         s.push(c);
59     } catch (const std::out_of_range &) {
60         try {
61             q.erase(Car{carNumber});
62             std::cout << "Parking time: " << Time{0, 0}
               << std::endl;
63             std::cout << "Fees to be paid: " << 0 << std
               ::endl;
64             std::cout << "The car left from corridor
               directly." << std::endl;

```



```

65         } catch (const std::out_of_range &) {
66             std::cerr << "No matching Car." << '\n';
67         }
68     }
69 }
70 }
71
72 // 打印停车场
73 void print() {
74     std::cout << "In Parking Lot:" << std::endl;
75     s.print();
76     std::cout << "In Corridor:" << std::endl;
77     q.print();
78 }
79 };

```

代码 6: ParkLot 类模板的实现

3.2 程序模块

收费算法的函数实现如代码8所示。

```

1  int fees(Car c) {
2      int fee;
3      Time t = c.getParingTime();
4      if (t <= Time{1, 0}) {
5          fee = 0;
6      } else if (t <= Time{2, 0}) {
7          fee = 5;
8      } else {
9          t -= Time{2, 0};
10         fee = 5 + (t.hour + (t.minute ? 1 : 0)) * 3;
11     }
12     return fee;
13 }

```

代码 7: 收费算法的函数实现

3.3 其他函数

main 函数、menu 函数，用于组织程序结构：

```
1 int main() {
2     cout << "Welcome to use Parking Management System!" << endl;
3     cout << "Please input the capacity of the Parking Lot: ";
4     Stack<Car>::size_type size{0};
5     cin >> size;
6     ParkLot p{size};
7     while (true) {
8         switch (menu()) {
9             case 1: {
10                 cout << "Please input the License Number of the
11                     car, arrive time" << endl
12                     << "(the format is \"number hour minute
13                         \", like: A-00001 12 59):" << endl;
14                 std::string number;
15                 int hour{0}, minute{0};
16                 cin >> number >> hour >> minute;
17                 p.add(number, Time{hour, minute});
18             } break;
19             case 2: {
20                 cout << "Please input the License Number of the
21                     car, leave time" << endl
22                     << "(the format is \"number hour minute
23                         \", like: A-00001 12 59):" << endl;
24                 std::string number;
25                 int hour{0}, minute{0};
26                 cin >> number >> hour >> minute;
27                 p.reduce(number, Time{hour, minute});
28             } break;
29             case 3:
30                 p.print();
31                 break;
32             case 0:
33                 cout << "Thanks for using!" << endl;
34                 exit(0);
35             }
36     }
37 }
38
39 int menu() {
```

```

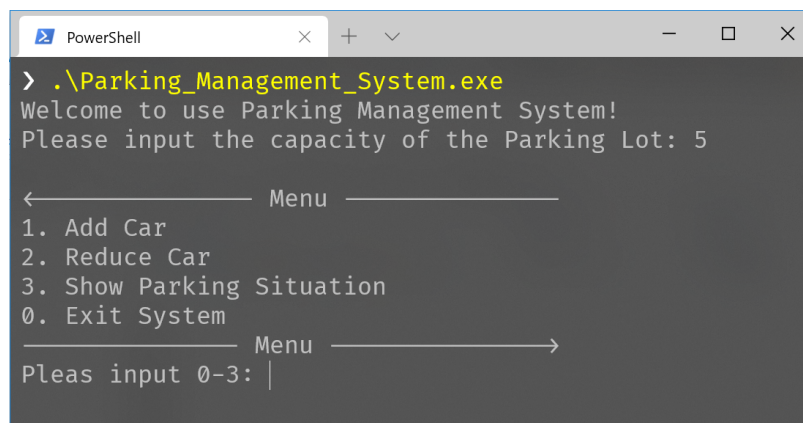
36     auto sn{0};
37     cout << endl
38         << "<----- Menu -----" << endl
39         << "1. Add Car" << endl
40         << "2. Reduce Car" << endl
41         << "3. Show Parking Situation" << endl
42         << "0. Exit System" << endl
43         << "----- Menu ----->" << endl
44         << "Pleas input 0-3: ";
45     while (true) {
46         cin >> sn;
47         if (sn < 0 || sn > 3) {
48             cout << "Error Number! Please Input 0-3: ";
49         } else
50             break;
51     }
52     return sn;
53 }

```

代码 8: 其他函数

4 测试分析

使用 G++ 编译器编译本实例，运行测试。



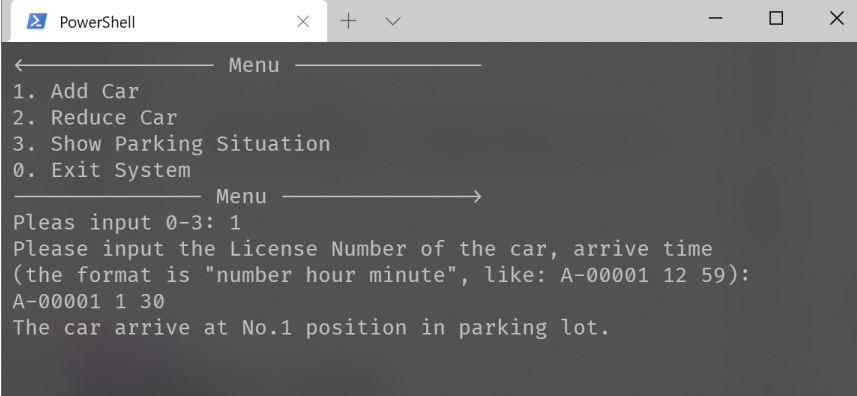
```

PowerShell
> .\Parking_Management_System.exe
Welcome to use Parking Management System!
Please input the capacity of the Parking Lot: 5

<----- Menu -----
1. Add Car
2. Reduce Car
3. Show Parking Situation
0. Exit System
----- Menu ----->
Pleas input 0-3: |

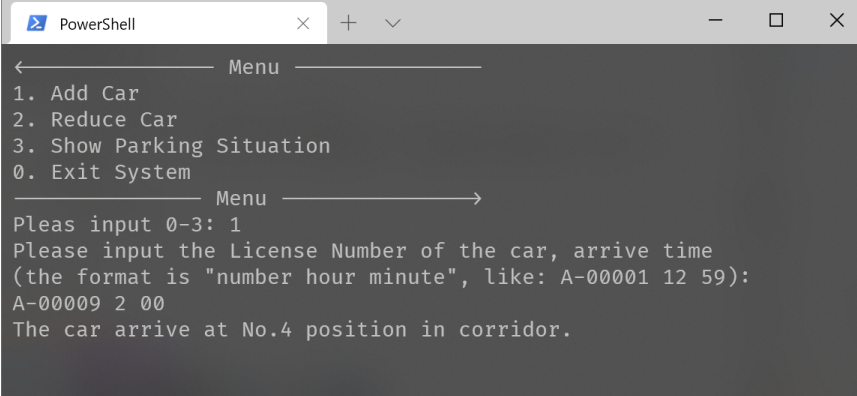
```

图 1: 初始化及菜单



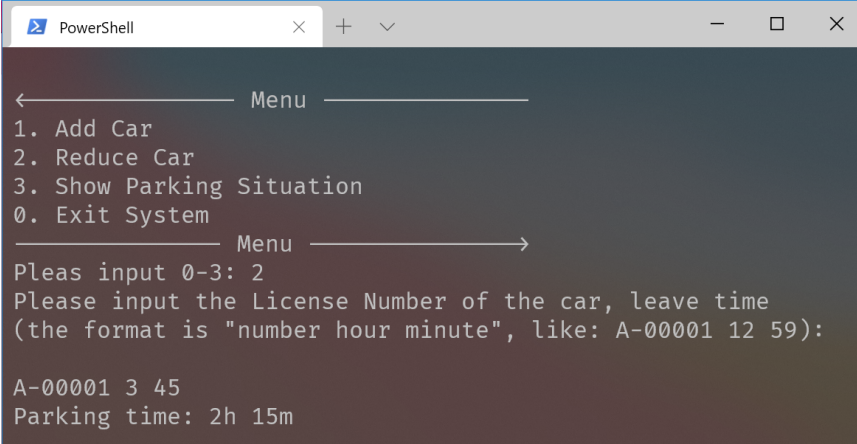
```
PowerShell
<----- Menu ----->
1. Add Car
2. Reduce Car
3. Show Parking Situation
0. Exit System
----- Menu ----->
Pleas input 0-3: 1
Please input the License Number of the car, arrive time
(the format is "number hour minute", like: A-00001 12 59):
A-00001 1 30
The car arrive at No.1 position in parking lot.
```

图 2: 到达停车场



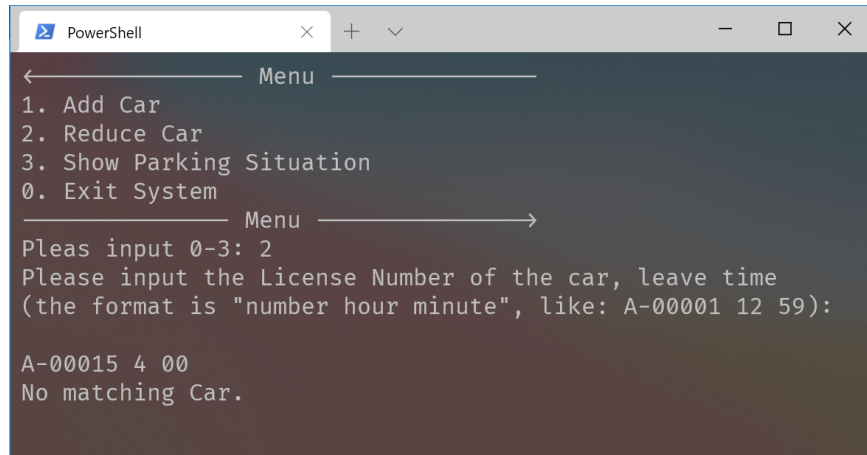
```
PowerShell
<----- Menu ----->
1. Add Car
2. Reduce Car
3. Show Parking Situation
0. Exit System
----- Menu ----->
Pleas input 0-3: 1
Please input the License Number of the car, arrive time
(the format is "number hour minute", like: A-00001 12 59):
A-00009 2 00
The car arrive at No.4 position in corridor.
```

图 3: 到达便道



```
PowerShell
<----- Menu ----->
1. Add Car
2. Reduce Car
3. Show Parking Situation
0. Exit System
----- Menu ----->
Pleas input 0-3: 2
Please input the License Number of the car, leave time
(the format is "number hour minute", like: A-00001 12 59):
A-00001 3 45
Parking time: 2h 15m
```

图 4: 从停车场离去车辆

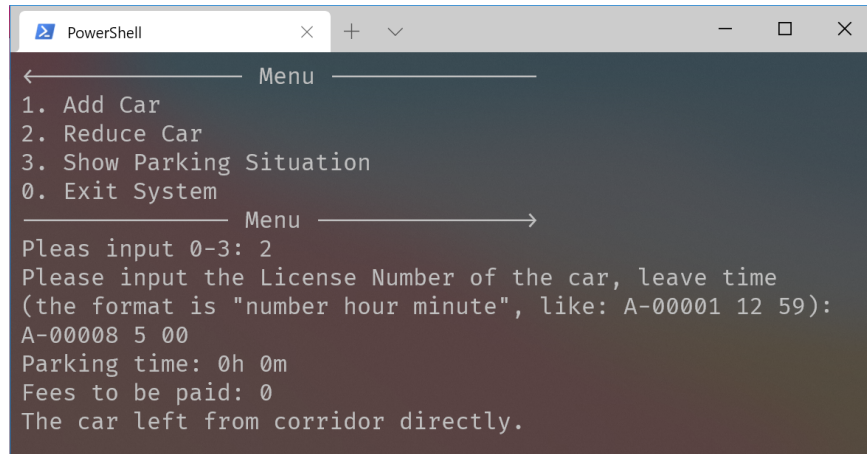


```

PowerShell
←———— Menu —————→
1. Add Car
2. Reduce Car
3. Show Parking Situation
0. Exit System
———— Menu —————→
Pleas input 0-3: 2
Please input the License Number of the car, leave time
(the format is "number hour minute", like: A-00001 12 59):
A-00015 4 00
No matching Car.

```

图 5: 没有找到车辆

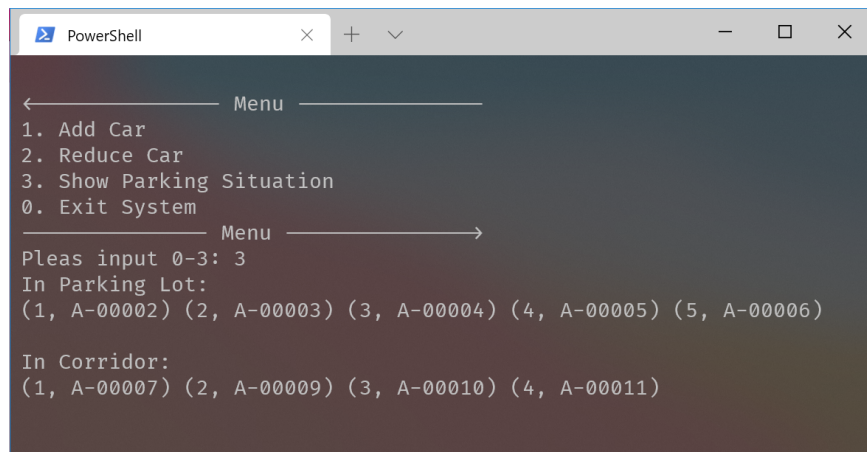


```

PowerShell
←———— Menu —————→
1. Add Car
2. Reduce Car
3. Show Parking Situation
0. Exit System
———— Menu —————→
Pleas input 0-3: 2
Please input the License Number of the car, leave time
(the format is "number hour minute", like: A-00001 12 59):
A-00008 5 00
Parking time: 0h 0m
Fees to be paid: 0
The car left from corridor directly.

```

图 6: 从便道离去车辆



```

PowerShell
←———— Menu —————→
1. Add Car
2. Reduce Car
3. Show Parking Situation
0. Exit System
———— Menu —————→
Pleas input 0-3: 3
In Parking Lot:
(1, A-00002) (2, A-00003) (3, A-00004) (4, A-00005) (5, A-00006)

In Corridor:
(1, A-00007) (2, A-00009) (3, A-00010) (4, A-00011)

```

图 7: 显示停车情况

5 总结

通过本次项目实践，熟练地掌握了栈、队列这两种数据结构，并实现了顺序栈、链队列模板的编写，增强了编码能力。

结合栈、队列设计停车场管理系统，使得栈和队列得到实际应用。在实践过程中，需要借助栈和队列对停车场内以及便道上的车辆进行有序地调度，增强了离散事件问题的模拟算法设计与求解的能力。同时，设计入栈、出栈、入队列、出队列、计时收费等系列算法的设计，对计算机如何模拟实际问题有了较好的感性理解。

此次项目实践，仍有不足之处，如：只设计了一天以内的计时收费，这与生活实际有出入。这是项目的改进方向。

6 附录

本项目实例使用 CMake 构建，并要求编译器为 G++，使用的操作系统是 Windows。

项目主要文件清单：

```

/.....项目根目录
├── bin.....输出文件夹
│   └── Parking_Management_System.exe.....已经编译的可执行文件
├── CMakeLists.txt.....CMake 项目配置文件
├── docs.....项目文档目录
│   ├── images.....文档使用的图片资源
│   ├── project2.pdf.....项目文档
│   └── project2.tex.....项目文档 LATEX 源文件
├── src.....源代码
│   ├── includes.....头文件包含目录
│   │   ├── Queue.h.....Queue 类模板头文件
│   │   └── Stack.h.....Stack 类模板头文件
│   ├── Car.h.....Time 类、Car 类头文件
│   ├── main.cpp.....主程序源代码
│   └── ParkLot.h.....ParkLot 类头文件

```

这些源文件可以在 <https://github.com/DianDengJun/Course-Design/tree/main/Winter/Project2> 中查看。

构建本实例的命令 (Bash 或 Powershell) 如下：

进入根目录，执行：

```

mkdir build
cd build
cmake -G "MinGW Makefiles" ..
make # 或者是 mingw32-make

```

然后进入 bin 目录运行 Parking_Management_System.exe。

```
cd ../bin
./Parking_Management_System
```