

第三章 栈与队列

姓名：黄昌斌

学号：2023140902005

Github仓库： https://github.com/Rain0832/Data_Structure_Course.git

上机时间：第 10 周星期三（11 月 6 日）

上机地点：基础实验大楼506

题目描述

- 设有一个可以停放 n 辆汽车的狭长停车场，它只有一个大门可以供车辆进出。
- 车辆按到达停车场时间的早晚，依次从停车场最里面向大门口处停放（最先到达的第一辆车 放在停车场的最里面）。
- 如果停车场已停放 n 辆车，则后来的车辆只能在停车场大门外的便道上等待，一旦停车场内有车开走，则排在便道上的第一辆车就进入停车场。
- 停车场内如有某辆车要开走，在它之后进入停车场的车都必须先退出停车场为它让路，待其开出停车场后，这些车辆再依原来的次序进场。
- 每辆车在离开停车场时，都应根据它在停车场内停留的时间长短交费。如果停留在便道上的车未进停车场要离去，允许其离去，不收停车费，并且仍然保持在便道上等待的车辆次序。

基本思想：根据题目要求，停车场只有一个大门，因此可用一个栈来模拟。而当栈满后，继续来到车辆只能停在便道上，根据便道停车的特点，可知这可以用一个队列来模拟，先排队的车辆先离开便道，进入停车场。由于排在停车场中间的车辆可以提出离开停车场，并且要求在离开车辆到停车场大门之间的车辆都必须离开停车场，让此车辆离去，然后再让这些车辆 依原来的次序进入停车场，因此在一个栈和一个队列的基础上，还需要有一个地方保存为了让路离开停车场的车辆，很显然这也应该用一个栈来模拟，因此，本题中要用到两个栈和一个队列

1. 链队列模拟便道停车

1.1 链队列结构定义与初始化

```
1  typedef struct Qnode {
2      QElemtype data;          // 数据域（存储车牌号和入场时间）
3      struct Qnode *next;      // 指针域
4  } Qnode, *Qptr;
5
6  typedef struct {
7      Qptr front;              // 队头指针
8      Qptr rear;               // 队尾指针
9  } LinkQ;
```

```
1  bool InitQueue(LinkQ &Q) {
2      Q.front = Q.rear = (Qptr)malloc(sizeof(Qnode)); // 申请一个节点空间
3      if (!Q.front) {
4          return false;
5      }
6      Q.front->next = NULL;
7      return true;
8  }
```

1.2 链队列入队操作

```
1  bool EnQueue(LinkQ &Q, QElemtype e) {
2      Qptr p = (Qptr)malloc(sizeof(Qnode));
3      if (!p) {
4          return false;
5      }
6      p->data = e;
7      p->next = NULL;
8      Q.rear->next = p;
9      Q.rear = p;
10     return true;
11 }
```

1.3 链队列出队操作

```
1  bool DeQueue(LinkQ &Q, QElemtype &e) {
2      if (Q.rear == Q.front) {
3          return false;
4      }
5      Qptr p = Q.front->next;
6      e = p->data;
7      Q.front->next = p->next;
8      if (p == Q.rear) {
9          Q.rear = Q.front;
10     }
11     free(p);
12     return true;
13 }
```

2. 顺序栈模拟停车场停车、临时

2.1 顺序栈结构定义与初始化

```
1  typedef struct {
2      SElemtype *base; // 栈底指针
3      SElemtype *top;  // 栈顶指针
4      int stacksize;   // 当前栈容量
5  } Sequences;
```

```
1  bool InitStack(Sequences &S) {
2      S.base = (SElemtype *)malloc(STACK_SIZE * sizeof(SElemtype));
3      if (!S.base) {
4          return false;
5      }
6      S.top = S.base;
7      S.stacksize = STACK_SIZE;
8      return true;
9  }
```

2.2 顺序栈入栈操作

```
1  bool PushStack(Squences &S, SElemtype e) {
2      // 栈满, 扩容
3      if (S.top - S.base >= S.stacksize) {
4          SElemtype *I_temp = (SElemtype *)realloc(S.base, (S.stacksize +
STACK_INCREMENT) * sizeof(SElemtype));
5          if (!I_temp) {
6              return false;
7          }
8          S.base = I_temp;
9          S.top = S.base + S.stacksize;
10         S.stacksize += STACK_INCREMENT;
11     }
12     *(S.top) = e;
13     S.top++;
14     return true;
15 }
```

2.3 顺序栈出栈操作

```
1  bool PopStack(Squences &S, SElemtype &e) {
2      if (S.top == S.base) {
3          return false;
4      }
5      S.top--;
6      e = *(S.top);
7      return true;
8  }
```

3. 程序模拟停车场停车

3.1 停车费计算

使用<time.h>库函数取系统时间进行计算（为方便测试，取收费标准为每秒 1 元，后续可更改）

```
1  int calculateFee(time_t enterTime) {
2      time_t currentTime = time(NULL);
3      double seconds = difftime(currentTime, enterTime); // 时间差（秒）
4      return (int)(seconds * PRICE); // 每秒1元
5  }
```

3.2 程序运行结果

- 程序初始交互界面（终端），显示当前停车状态。

```
欢迎来到停车场
当前停车场状态：
共 0 辆车在停车场内
当前便道停车状态：
共 0 辆车在便道上

1. 入车
2. 出车
3. 刷新页面
4. 退出程序
请选择操作：
```

- 选择入车操作时，输出车辆编号，若停车场未满（初始设定为 5），则进入停车场（栈模拟），停车。

```
○ 欢迎来到停车场
当前停车场状态：
车辆 1： 车牌号 1
共 1 辆车在停车场内
当前便道停车状态：
共 0 辆车在便道上

1. 入车
2. 出车
3. 刷新页面
4. 退出程序
请选择操作： 1
请输入车辆编号： 2
车辆 2 成功进入停车场！
Press any key to continue . . .
```

- 当停车场已满时，新来的车会进入便道停车（队列模拟）。

```
○ 欢迎来到停车场
当前停车场状态：
车辆 1： 车牌号 1
车辆 2： 车牌号 2
车辆 3： 车牌号 3
车辆 4： 车牌号 4
车辆 5： 车牌号 5
共 5 辆车在停车场内
当前便道停车状态：
车辆 1： 车牌号 6
共 1 辆车在便道上

1. 入车
2. 出车
3. 刷新页面
4. 退出程序
请选择操作： 1
请输入车辆编号： 7
停车场已满，车辆 7 已进入便道！
Press any key to continue . . .
```

- 选择出车操作的时候，如果是便道出车，可以直接离去，不收取停车费。

```
○ 欢迎来到停车场
当前停车场状态：
车辆 1： 车牌号 1
车辆 2： 车牌号 2
车辆 3： 车牌号 3
车辆 4： 车牌号 4
车辆 5： 车牌号 5
共 5 辆车在停车场内
当前便道停车状态：
车辆 1： 车牌号 6
车辆 2： 车牌号 7
共 2 辆车在便道上

1. 入车
2. 出车
3. 刷新页面
4. 退出程序
请选择操作： 2
请输入要出车的车辆编号： 7
车辆 7 从便道离去，不收停车费！
Press any key to continue . . .
```

- 而如果是停车场停车，其他车要进行让道（栈模拟），对应的车要根据时间长短进行收费。

○ 欢迎来到停车场
当前停车场状态：
车辆 1： 车牌号 1
车辆 2： 车牌号 2
车辆 3： 车牌号 3
车辆 4： 车牌号 4
车辆 5： 车牌号 5
共 5 辆车在停车场内
当前便道停车状态：
车辆 1： 车牌号 6
共 1 辆车在便道上

1. 入车
2. 出车
3. 刷新页面
4. 退出程序
请选择操作： 2
请输入要出车的车辆编号： 1
车辆 1 成功出车！ 停车费用： 146元
Press any key to continue . . .

3.3 源程序代码

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #define QElemtype Car
6  #define SElemtype Car
7
8  #define STACK_SIZE 5           // 栈容量
9  #define STACK_INCREMENT 3      // 栈扩容增量
10 #define PRICE 1                // 价格收费 1元/秒
11
12 // 车辆结构体
13 typedef struct Car {
14     int carNum;                // 车牌号
15     time_t enterTime;         // 入场时间
16 } Car;
17
18 // 队列模拟：便道停车
19 // 链队列
20
21 // 存储结构定义
22 typedef struct Qnode {
23     QElemtype data;           // 数据域（存储车牌号和入场时间）
24     struct Qnode *next;       // 指针域
25 } Qnode, *Qptr;
26
27 typedef struct {
28     Qptr front;               // 队头指针
29     Qptr rear;                // 队尾指针
30 } LinkQ;
31
```

```

32 // 链队列初始化
33 bool InitQueue(LinkQ &Q) {
34     Q.front = Q.rear = (Qptr)malloc(sizeof(Qnode)); // 申请一个节点空间
35     if (!Q.front) {
36         return false;
37     }
38     Q.front->next = NULL;
39     return true;
40 }
41
42 // 打印队列
43 void PrintQueue(LinkQ &Q) {
44     int cnt = 0;
45     Qptr p = Q.front->next;
46     while (p != NULL) {
47         printf("车辆 %d :  车牌号 %d\n", cnt + 1, p->data.carNum);
48         p = p->next;
49         cnt++;
50     }
51     printf("共 %d 辆车在便道上\n", cnt);
52 }
53
54 // 链队列入队操作
55 bool EnQueue(LinkQ &Q, QElemtype e) {
56     Qptr p = (Qptr)malloc(sizeof(Qnode));
57     if (!p) {
58         return false;
59     }
60     p->data = e;
61     p->next = NULL;
62     Q.rear->next = p;
63     Q.rear = p;
64     return true;
65 }
66
67 // 链队列出队操作
68 bool DeQueue(LinkQ &Q, QElemtype &e) {
69     if (Q.rear == Q.front) {
70         return false;
71     }
72     Qptr p = Q.front->next;
73     e = p->data;
74     Q.front->next = p->next;
75     if (p == Q.rear) {
76         Q.rear = Q.front;
77     }
78     free(p);
79     return true;
80 }
81
82 // 栈模拟：停车场停车、临时让路停车
83 // 顺序栈
84 typedef struct {
85     SElemtype *base; // 栈底指针
86     SElemtype *top;  // 栈顶指针
87     int stacksize;   // 当前栈容量
88 } SequenceS;
89

```

```

90 // 顺序栈初始化
91 bool InitStack(Squences &S) {
92     S.base = (SElementype *)malloc(STACK_SIZE * sizeof(SElementype));
93     if (!S.base) {
94         return false;
95     }
96     S.top = S.base;
97     S.stacksize = STACK_SIZE;
98     return true;
99 }
100
101 // 打印栈
102 void PrintStack(Squences &S) {
103     int cnt = 0;
104     SElementype *p = S.base;
105     while (p != S.top) {
106         printf("车辆 %d : 车牌号 %d\n", cnt + 1, p->carNum);
107         p++;
108         cnt++;
109     }
110     printf("共 %d 辆车在停车场内\n", cnt);
111 }
112
113 // 顺序栈入栈操作
114 bool PushStack(Squences &S, SElementype e) {
115     // 栈满, 扩容
116     if (S.top - S.base >= S.stacksize) {
117         SElementype *I_temp = (SElementype *)realloc(S.base, (S.stacksize +
STACK_INCREMENT) * sizeof(SElementype));
118         if (!I_temp) {
119             return false;
120         }
121         S.base = I_temp;
122         S.top = S.base + S.stacksize;
123         S.stacksize += STACK_INCREMENT;
124     }
125     *(S.top) = e;
126     S.top++;
127     return true;
128 }
129
130 // 顺序栈出栈操作
131 bool PopStack(Squences &S, SElementype &e) {
132     if (S.top == S.base) {
133         return false;
134     }
135     S.top--;
136     e = *(S.top);
137     return true;
138 }
139
140 // 停车费计算
141 int CalculateFee(time_t enterTime) {
142     time_t currentTime = time(NULL);
143     double seconds = difftime(currentTime, enterTime); // 时间差 (秒)
144     return (int)(seconds * PRICE); // 每秒1元
145 }
146

```



```

147 // 显示菜单
148 void showMenu(Sequences &parkLot, LinkQ &parkway) {
149     printf("欢迎来到停车场\n");
150     printf("当前停车场状态: \n");
151     PrintStack(parkLot);
152     printf("当前便道停车状态: \n");
153     PrintQueue(parkway);
154     printf("\n");
155     printf("1. 入车\n");
156     printf("2. 出车\n");
157     printf("3. 刷新页面\n");
158     printf("4. 退出程序\n");
159     printf("请选择操作: ");
160 }
161
162 int main(void) {
163     // 栈模拟停车场停车
164     Sequences parkLot;
165     InitStack(parkLot);
166
167     // 栈模拟临时让路停车
168     Sequences tempLot;
169     InitStack(tempLot);
170
171     // 链队列模拟便道停车
172     LinkQ parkway;
173     InitQueue(parkway);
174
175     int select = 0;
176     while (true) {
177         showMenu(parkLot, parkway);
178         scanf("%d", &select);
179         switch (select) {
180             case 1: { // 入车
181                 int carNum;
182                 printf("请输入车辆编号: ");
183                 scanf("%d", &carNum);
184                 Car newCar = {carNum, time(NULL)};
185                 if (parkLot.top - parkLot.base < STACK_SIZE) {
186                     PushStack(parkLot, newCar);
187                     printf("车辆 %d 成功进入停车场! \n", carNum);
188                 } else {
189                     printf("停车场已满, 车辆 %d 已进入便道! \n", carNum);
190                     EnQueue(parkway, newCar);
191                 }
192                 break;
193             }
194
195             case 2: { // 出车
196                 int carNum;
197                 printf("请输入要出车的车辆编号: ");
198                 scanf("%d", &carNum);
199
200                 // 检查车辆是否在便道
201                 Qptr prev = parkway.front;
202                 Qptr curr = parkway.front->next;
203                 bool foundInQueue = false;
204

```

```

205         while (curr != NULL) {
206             if (curr->data.carNum == carNum) {
207                 foundInQueue = true;
208                 prev->next = curr->next;
209                 if (curr == parkway.rear) {
210                     parkway.rear = prev;
211                 }
212                 free(curr);
213                 printf("车辆 %d 从便道离去, 不收停车费! \n", carNum);
214                 break;
215             }
216             prev = curr;
217             curr = curr->next;
218         }
219
220         if (foundInQueue) break; // 已处理完, 退出 case
221
222         // 检查车辆是否在停车场
223         Car tempCar;
224         bool foundInStack = false;
225         while (PopStack(parkLot, tempCar)) {
226             if (tempCar.carNum == carNum) {
227                 foundInStack = true;
228                 int fee = CalculateFee(tempCar.enterTime);
229                 printf("车辆 %d 成功出车! 停车费用: %d元\n", carNum,
fee);
230                 break;
231             } else {
232                 PushStack(tempLot, tempCar);
233             }
234         }
235
236         // 恢复让车道车辆
237         while (PopStack(tempLot, tempCar)) {
238             PushStack(parkLot, tempCar);
239         }
240
241         if (!foundInStack) {
242             printf("未找到车辆 %d! \n", carNum);
243         }
244         break;
245     }
246
247     case 3: // 刷新页面
248         break;
249
250     case 4: // 退出程序
251         printf("程序结束.\n");
252         return 0;
253
254     default:
255         printf("无效选择, 请重新输入.\n");
256         break;
257 }
258 system("pause");
259 system("cls");
260 }
261 return 0;

```

3.4 实验结论与结果分析

- 通过实验掌握栈和队列的顺序存储和链式存储结构。
- 通过实验掌握栈和队列的特点和基本运算。
- 通过实验使用栈和队列进行停车场停车问题模拟，可以实现栈和队列的基本功能
- 通过实验将数据结构知识和实际生活结合，开发出实用性强的程序

相比于传统的理论学习方式，这次实验通过具体问题驱动学习，将抽象的数据结构知识融入实际场景，不仅加深了对知识的理解，还培养了逻辑思维能力和编程能力，具有显著的实践意义和学习价值。