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，继续来到的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

```
typedef struct Qnode {
    QElemtype data;          // 数据域（存储车牌号和入场时间）
    struct Qnode *next;      // 指针域
} Qnode, *Qptr;

typedef struct {
    Qptr front;              // 队头指针
    Qptr rear;               // 队尾指针
} LinkQ;
```

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

1.2 链队列入队操作

```
bool EnQueue(LinkQ &Q, QElemtype e) {
    Qptr p = (Qptr)malloc(sizeof(Qnode));
    if (!p) {
        return false;
    }
    p->data = e;
    p->next = NULL;
    Q.rear->next = p;
    Q.rear = p;
    return true;
}
```

1.3 链队列出队操作

```
bool DeQueue(LinkQ &Q, QElemtype &e) {
    if (Q.rear == Q.front) {
        return false;
    }
    Qptr p = Q.front->next;
    e = p->data;
    Q.front->next = p->next;
    if (p == Q.rear) {
        Q.rear = Q.front;
    }
    free(p);
    return true;
}
```

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

```
typedef struct {
    SElemtype *base; // 栈底指针
    SElemtype *top;  // 栈顶指针
    int stacksize;   // 当前栈容量
} SquenceS;
```

```
bool InitStack(SquenceS &S) {
    S.base = (SElemtype *)malloc(STACK_SIZE * sizeof(SElemtype));
    if (!S.base) {
        return false;
    }
    S.top = S.base;
    S.stacksize = STACK_SIZE;
    return true;
}
```

2.2 顺序栈入栈操作

```
bool PushStack(SquenceS &S, SElemtype e) {
    // 栈满，扩容
    if (S.top - S.base >= S.stacksize) {
        SElemtype *I_temp = (SElemtype *)realloc(S.base, (S.stacksize + STACK_INCREMENT) * sizeof(SElemtype));
        if (!I_temp) {
            return false;
        }
        S.base = I_temp;
        S.top = S.base + S.stacksize;
        S.stacksize += STACK_INCREMENT;
    }
    *(S.top) = e;
    S.top++;
    return true;
}
```

2.3 顺序栈出栈操作

```
bool PopStack(SquenceS &S, SElemtype &e) {
    if (S.top == S.base) {
        return false;
    }
    S.top--;
    e = *(S.top);
    return true;
}
```

3. 程序模拟停车场停车

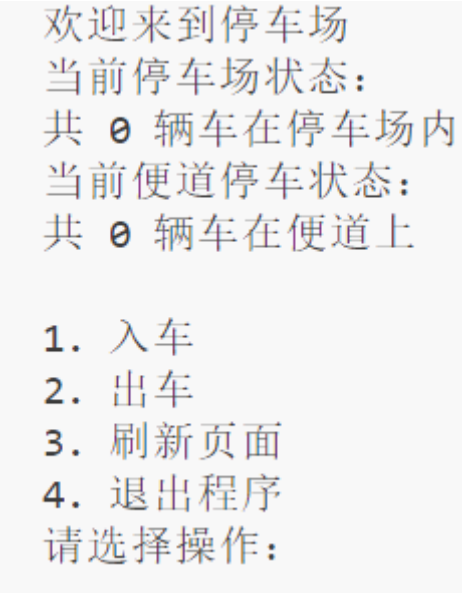
3.1 停车费计算

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）

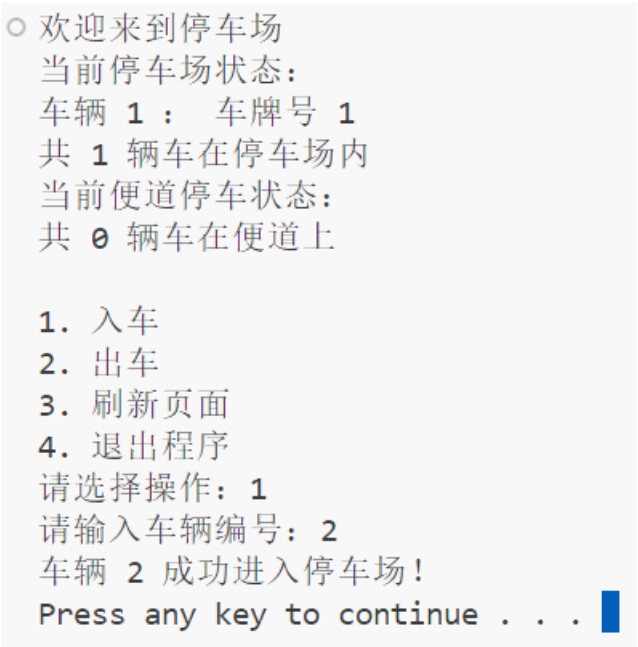
```
int CalculateFee(time_t enterTime) {
    time_t currentTime = time(NULL);
    double seconds = difftime(currentTime, enterTime); // 时间差（秒）
    return (int)(seconds * PRICE); // 每秒1元
}
```

3.2 程序运行结果

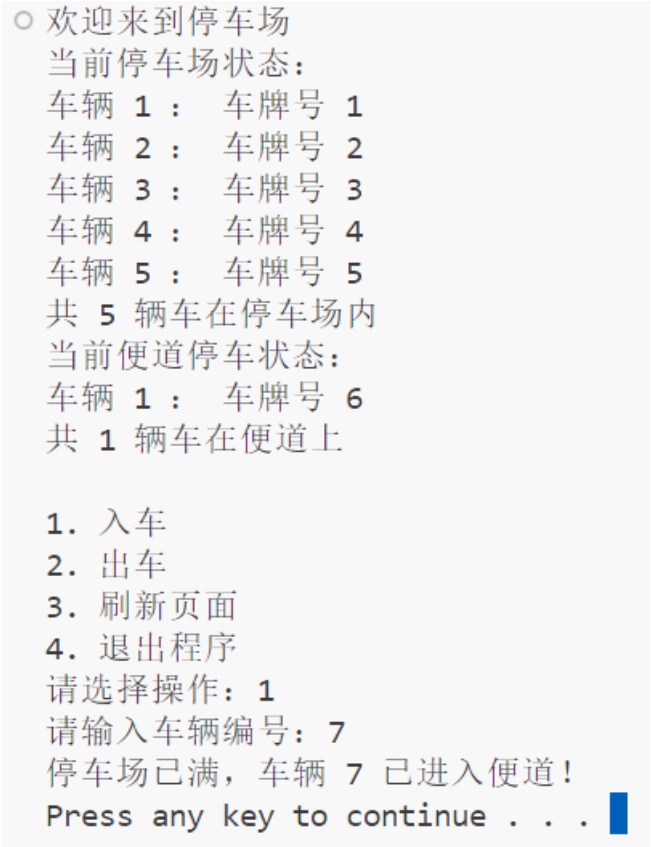
- 程序初始交互界面（终端），显示当前停车状态。



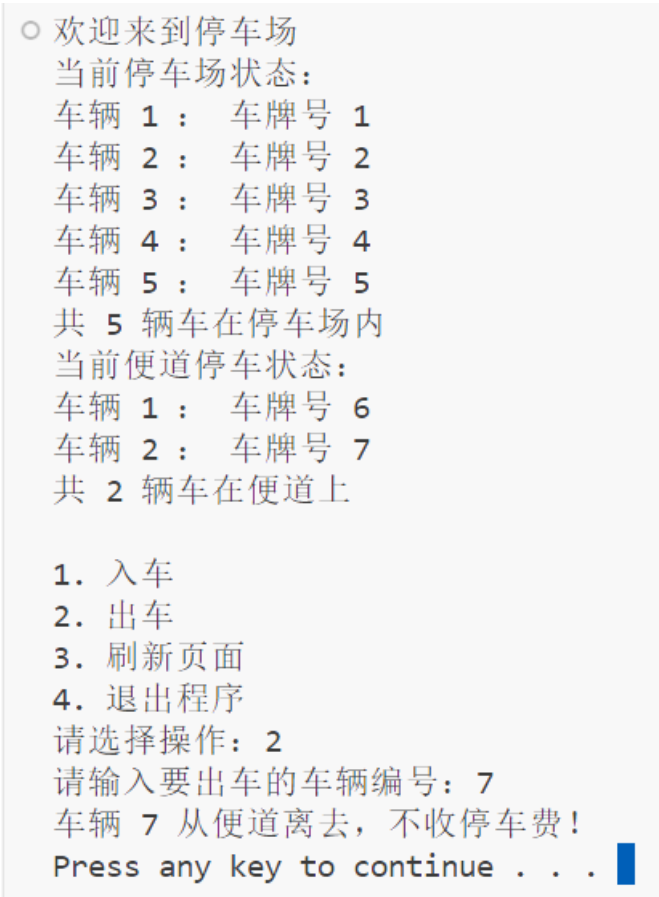
- 选择入车操作时，输出车辆编号，若停车场未满（初始设定为 5 ），则进入停车场（栈模拟），停车。



- 当停车场已满时，新来的车会进入便道停车（队列模拟）。



- 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。



- 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

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define QElemtype Car
#define SElemtype Car

#define STACK_SIZE 5             // 栈容量
#define STACK_INCREMENT 3        // 栈扩容增量
#define PRICE 1                  // 价格收费 1元/秒

// 车辆结构体
typedef struct Car {
    int carNum;         // 车牌号
    time_t enterTime;   // 入场时间
} Car;

// 队列模拟：便道停车
// 链队列

// 存储结构定义
typedef struct Qnode {
    QElemtype data;          // 数据域（存储车牌号和入场时间）
    struct Qnode *next;      // 指针域
} Qnode, *Qptr;

typedef struct {
    Qptr front;              // 队头指针
    Qptr rear;               // 队尾指针
} LinkQ;

```

```cpp
// 链队列初始化
bool InitQueue(LinkQ &Q) {
    Q.front = Q.rear = (Qptr)malloc(sizeof(Qnode)); // 申请一个节点空间
    if (!Q.front) {
        return false;
    }
    Q.front->next = NULL;
    return true;
}

// 打印队列
void PrintQueue(LinkQ &Q) {
    int cnt = 0;
    Qptr p = Q.front->next;
    while (p != NULL) {
        printf("车辆 %d ： 车牌号 %d\n", cnt + 1, p->data.carNum);
        p = p->next;
        cnt++;
    }
    printf("共 %d 辆车在便道上\n", cnt);
}

// 链队列入队操作
bool EnQueue(LinkQ &Q, QElemtype e) {
    Qptr p = (Qptr)malloc(sizeof(Qnode));
    if (!p) {
        return false;
    }
    p->data = e;
    p->next = NULL;
    Q.rear->next = p;
    Q.rear = p;
    return true;
}

// 链队列出队操作
bool DeQueue(LinkQ &Q, QElemtype &e) {
    if (Q.rear == Q.front) {
        return false;
    }
    Qptr p = Q.front->next;
    e = p->data;
    Q.front->next = p->next;
    if (p == Q.rear) {
        Q.rear = Q.front;
    }
    free(p);
    return true;
}

// 栈模拟：停车场停车、临时让路停车
// 顺序栈
typedef struct {
    SElemtype *base; // 栈底指针
    SElemtype *top;  // 栈顶指针
    int stacksize;   // 当前栈容量
} SquenceS;

```

```cpp
// 顺序栈初始化
bool InitStack(SquenceS &S) {
    S.base = (SElemtype *)malloc(STACK_SIZE * sizeof(SElemtype));
    if (!S.base) {
        return false;
    }
    S.top = S.base;
    S.stacksize = STACK_SIZE;
    return true;
}

// 打印栈
void PrintStack(SquenceS &S) {
    int cnt = 0;
    SElemtype *p = S.base;
    while (p != S.top) {
        printf("车辆 %d :  车牌号 %d\n", cnt + 1, p->carNum);
        p++;
        cnt++;
    }
    printf("共 %d 辆车在停车场内\n", cnt);
}

// 顺序栈入栈操作
bool PushStack(SquenceS &S, SElemtype e) {
    // 栈满，扩容
    if (S.top - S.base >= S.stacksize) {
        SElemtype *I_temp = (SElemtype *)realloc(S.base, (S.stacksize +
STACK_INCREMENT) * sizeof(SElemtype));
        if (!I_temp) {
            return false;
        }
        S.base = I_temp;
        S.top = S.base + S.stacksize;
        S.stacksize += STACK_INCREMENT;
    }
    *(S.top) = e;
    S.top++;
    return true;
}

// 顺序栈出栈操作
bool PopStack(SquenceS &S, SElemtype &e) {
    if (S.top == S.base) {
        return false;
    }
    S.top--;
    e = *(S.top);
    return true;
}

// 停车费计算
int CalculateFee(time_t enterTime) {
    time_t currentTime = time(NULL);
    double seconds = difftime(currentTime, enterTime); // 时间差（秒）
    return (int)(seconds * PRICE); // 每秒1元
}

```

```
// 显示菜单
void showMenu(SquenceS &parkLot, LinkQ &parkWay) {
    printf("欢迎来到停车场\n");
    printf("当前停车场状态：\n");
    PrintStack(parkLot);
    printf("当前便道停车状态：\n");
    PrintQueue(parkWay);
    printf("\n");
    printf("1. 入车\n");
    printf("2. 出车\n");
    printf("3. 刷新页面\n");
    printf("4. 退出程序\n");
    printf("请选择操作：");
}

int main(void) {
    // 栈模拟停车场停车
    SquenceS parkLot;
    InitStack(parkLot);

    // 栈模拟临时让路停车
    SquenceS tempLot;
    InitStack(tempLot);

    // 链队列模拟便道停车
    LinkQ parkWay;
    InitQueue(parkWay);

    int select = 0;
    while (true) {
        showMenu(parkLot, parkWay);
        scanf("%d", &select);
        switch (select) {
            case 1: { // 入车
                int carNum;
                printf("请输入车辆编号：");
                scanf("%d", &carNum);
                Car newCar = {carNum, time(NULL)};
                if (parkLot.top - parkLot.base < STACK_SIZE) {
                    PushStack(parkLot, newCar);
                    printf("车辆 %d 成功进入停车场！\n", carNum);
                } else {
                    printf("停车场已满，车辆 %d 已进入便道！\n", carNum);
                    EnQueue(parkWay, newCar);
                }
                break;
            }

            case 2: { // 出车
                int carNum;
                printf("请输入要出车的车辆编号：");
                scanf("%d", &carNum);

                // 检查车辆是否在便道
                Qptr prev = parkWay.front;
                Qptr curr = parkWay.front->next;
                bool foundInQueue = false;

```

```
                while (curr != NULL) {
                    if (curr->data.carNum == carNum) {
                        foundInQueue = true;
                        prev->next = curr->next;
                        if (curr == parkWay.rear) {
                            parkWay.rear = prev;
                        }
                        free(curr);
                        printf("车辆 %d 从便道离去，不收停车费！\n", carNum);
                        break;
                    }
                    prev = curr;
                    curr = curr->next;
                }

                if (foundInQueue) break; // 已处理完，退出 case

                // 检查车辆是否在停车场
                Car tempCar;
                bool foundInStack = false;
                while (PopStack(parkLot, tempCar)) {
                    if (tempCar.carNum == carNum) {
                        foundInStack = true;
                        int fee = CalculateFee(tempCar.enterTime);
                        printf("车辆 %d 成功出车！停车费用：%d元\n", carNum,
fee);
                        break;
                    } else {
                        PushStack(tempLot, tempCar);
                    }
                }

                // 恢复让车道车辆
                while (PopStack(tempLot, tempCar)) {
                    PushStack(parkLot, tempCar);
                }

                if (!foundInStack) {
                    printf("未找到车辆 %d！\n", carNum);
                }
                break;
            }

            case 3: // 刷新页面
                break;

            case 4: // 退出程序
                printf("程序结束。\n");
                return 0;

            default:
                printf("无效选择，请重新输入。\n");
                break;
        }
        system("pause");
        system("cls");
    }
    return 0;
```

```
262 }
```

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。