组号: 5

上海大学计算机工程与科学学院

实 验 报 告

（数据结构 1）

学　　期：2024-2025 年冬季

组　　长：冯俊佳

学　　号：23122721

指导教师：朱能军

成绩评定：　　（教师填写）

二〇二四年十一月二十九日

小组信息				
登记序号	姓名	学号	贡献比	签名
1	冯俊佳	23122721	1/3	冯俊佳
2	钱傲栋	23122803	1/3	钱傲栋
3	邱添	23122720	1/3	邱添

实验概述	
实验零	（熟悉上机环境、进度安排、评分制度；确定小组成员）
实验一	抽取简历
实验二	车厢调度
实验三	
实验四	

实验二

1 实验题目

有一个“丁”字型铁路调度系统如图 1 所示，它由相互垂直的 2 条铁轨组成，水平方向的为主铁轨，竖直方向的为辅助铁轨。辅助铁轨用于对车厢次序进行调整，它在主铁轨中间，把主铁轨分成左右两个部分。主铁轨左边的车厢只能从左边开到右边，或者从主铁轨左边进入辅助铁轨；辅助铁轨上的车厢只可以进入主铁轨右边。

1.1 车厢调度（1）

现在有 n 节火车车厢，编号为 1、2、...、n，在主铁轨的左边按顺序驶入，要求通过这个调度系统，在主铁轨的右边以指定次序开出（例如：有 5 节车厢以 1、2、3、4、5 的次序进入，要求以 3、2、5、4、1 的顺序出站）。请编程求解调度过程。

1.2 车厢调度（2）

现在有 n 节火车车厢，编号为 1、2、...、n，在主铁轨的左边以任意的顺序驶入，要求通过这个调度系统，在主铁轨的右边以 1、2、...、n 的次序开出（例如：有 5 节车厢以 5、3、1、2、4 的次序进入，要求以 1、2、3、4、5 的顺序出站）。请编程求解调度过程。

如果能完成调度，则输出调度过程，否则输出调度失败信息。

（部分相同题干已省略）

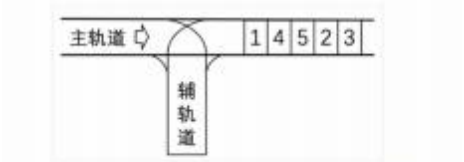


图1 “丁”字型铁路调度系统（按指定次序驶出）

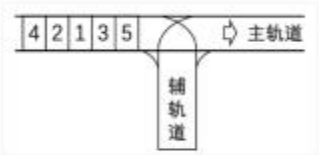


图2 “丁”字型铁路调度系统（按从小到大顺序驶出）

2 实验内容

利用栈/队列来实现车厢调度问题，以达到熟练使用栈/队列数据结构的目的。分别完成顺序驶入，指定次序开出/制定次序驶入，顺序开出的调度任务。并总结规律，深刻理解栈/队列的特点。

3 解决方案

3.1 算法设计

3.1.1 数据结构

本次实验运用 SeqStack 的数据结构完成车厢调度，具体原因如下：

- 利用栈的先进后出特性：SeqStack 是一种栈数据结构，具有先进后出的特性。这能很好地模拟辅助轨道的行为，即最后进入辅助轨道的车厢首先出轨道。栈的特点使得我们可以按照正确的顺序从辅助轨道中将车厢移动到主轨道右边。
- 方便的入栈和出栈操作：SeqStack 提供了 Push 和 Pop 方法，方便地实现了车厢的入栈和出栈操作。当需要将车厢从主轨道左边移动到辅助轨道时，使用 Push 将车厢入栈（从主轨道左边使入辅助轨道）；当需要将车厢从辅助轨道移动到主轨道右边时，使用 Pop 将车厢出栈(从辅助轨道使入主轨道右侧)。
- 栈顶元素的访问和比较：SeqStack 还提供了 GetTop 方法，可以方便地访问栈顶元素。在题目中，我们需要判断栈顶元素与当前输入车厢编号是否匹配(从辅助轨道使入主轨道右侧)，以决定是否移动车厢。GetTop 方法的存在使得这一操作变得简单。此外，我们还可以比较栈顶元素与其他元素进行大小或相等判断。

3.1.2 算法思想

本实验使用栈（Stack）来模拟辅助铁轨的操作。依次读取车厢序列，根据题目要求进行调度操作，具体操作包括车厢的入栈、出栈以及从一个铁轨移动到另一个铁轨等操作，直至满足指定的次序排列或判断调度失败。

3.1.3 主要操作

初始化 a[], b[]数组分别表示主轨道左侧与右侧，输入车厢总数 n 以及入站顺序。对比主轨道第一节车厢与出站车厢是否相同，若相同，从主轨道左边进入右边；若不同，对比辅轨道栈顶车厢与出站车厢是否相同，若相同，从辅轨道进入主轨道右边，若不同，主轨道第一节车厢进入辅轨道；以此类推，直至调度成功或失败（主轨道左边为空且辅轨道栈顶与出站车厢不匹配）。

3.2 源程序代码

以下简介几个该程序的核心代码，其中包括：入栈、出栈等函数。

3.2.1 入栈 Push 函数

若栈未满，将新元素入栈。具体代码如下：

```
1.    void SeqStack::Push(int d)
2.    {
3.        if(top<=MaxSize)
4.        {
5.            a[top]=d;
6.            ++top;
7.        }
8.        else
9.        {
10.           cout << "栈已满，无法入栈" << endl;
11.       }
12.   }
```

3.2.2 出栈 Pop 函数

若栈非空，弹出栈顶元素。具体代码如下：

```
1.    void SeqStack::Pop()
2.    {
3.        if(!IsEmpty()) {top--;  }        //如果栈非空，弹出栈顶元素
4.        else return;
5.    }
```

3.2.3 栈顶元素访问 GetTop 函数

若栈非空，返回栈顶元素；否则返回-1。具体代码如下：

```
1.    int SeqStack::GetTop()
2.    {
3.        if (top>0)
4.        {
5.            return a[top-1]; // 返回栈顶元素
6.        }
7.        else
8.        {
9.            cout <<"栈为空" << endl;
10.           return -1; // 表示栈为空
11.       }
12.   }
```

3.2.4 Ans 计数器

在调度过程中，我们希望检查是否可以完成所有车厢的调度。如果经过一轮操作后，ans 的值达到了车厢的总数 n，但是 a[n-1]和 b[n-1]还没有匹配（即最后一个车厢还没有按正确顺序进入主轨道右边），最后一个元素不匹配，栈内的元素无法出栈（即停留在栈内的车厢无法进入主轨道），就说明无法完成调度，程序将输出“调度无法完成”并退出。

同时，在调度过程中，如果车厢顺序无法完成调度，ans 可以帮助防止程序进入死循环。因为在调度过程中，无论是将车厢从主轨道进入辅轨道，还是将辅轨道的车厢从栈中弹出，都会增加 ans 的值。如果 ans 的值达到 n，说明程序已经执行了 n 步操作，如果仍然无法完成调度，就说明不可能完成调度，从而可以及时结束程序防止进入死循环。

具体代码如下：

```
1.        else if(ans==n&&a[n-1]!=b[n-1]) {
2.                cout << "调度无法完成" << endl;
3.                return 0;
4.             }
```

3.2.5 主函数

初始化 a[]，b[]数组分别表示主轨道左侧与右侧，输入车厢总数 n 以及入站顺序。根据判断主轨道左边，辅轨道栈顶车厢与出站车厢是否相同，进行相应操作，直至调度成功或失败。

```
5.    int main(){
6.        int a[128], b[128];
7.        int n;       // 输入的车厢个数
8.        int ans=0;
9.        cout << "输入车厢数：";
10.       cin >> n;
11.       SeqStack S(n);
12.
13.       cout << "输入" << n << "节车厢的入站顺序：";
14.       for (int i=0;i<n;i++) {  // 初始化 b 数组
15.           cin >> b[i];     // b[]数组为要求的顺序
16.           if(b[i]>n) {
17.               cout << "越界" << endl;
18.               return 0;
19.           }
20.       }
```

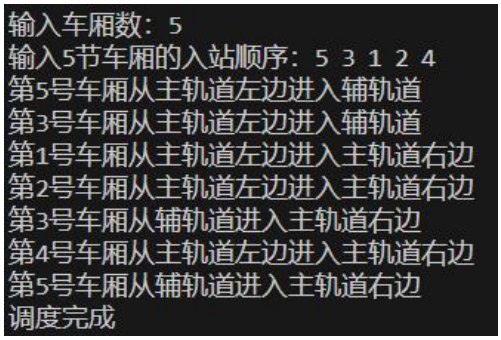
```
21.
22.         for (int i=0;i<n;i++)// 初始化 a 数组
23.             a[i]=i+1;
24.
25.         int j=0;  // a 数组的指针
26.         int i=0;  // b 数组的指针
27.         while (i<n) {  // 用 b[]控制循环的次数
28.             ans++;
29.             if(a[j]==b[i]) { // 如果 a[j]与 b[i]相同
30.                 cout << "第" << b[i] << "号车厢从主轨道左边进入主轨道
    右边" << endl;
31.                 i++;
32.                 j++;
33.             }
34.             else if(S.GetTop()==b[i]&&!S.IsEmpty()) {
35.                 cout << "第" << b[i] << "号车厢从辅轨道进入主轨道右边
    " << endl;
36.                 S.Pop();
37.                 i++;
38.             }
39.             else if(ans==n&&a[n-1]!=b[n-1]) {
40.                 cout << "调度无法完成" << endl;
41.                 return 0;
42.              }
43.             else {
44.                 for(int ct=0;ct<S.top-1;ct++) {
45.                     if(a[ct]==j) {
46.                         cout << "调度无法完成" << endl;
47.                         return 0;
48.                     }
49.                 }
50.                 if (j<n) {
51.                     S.Push(a[j]);
52.                     cout << "第" << a[j] << "号车厢从主轨道左边进入辅
    轨道" << endl;
53.                     j++;
54.                 }
55.                 else {
56.                     cout << "调度无法完成" << endl;
57.                     return 0;
58.                 }
59.             }
60.         }
61.         cout << "调度完成" << endl;
62.         return 0;
63.     }
```

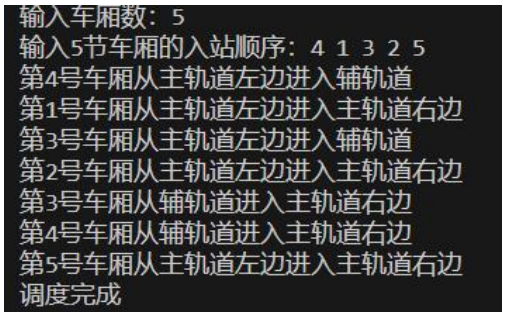
3.3 运行结果

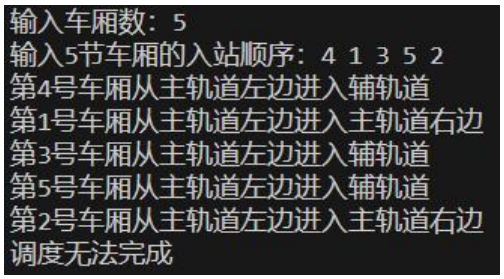
3.3.1 车厢调度（1）运行结果

本问在课上已验收并通过，因此在本实验报告中不加以赘述。

3.3.1 车厢调度（2）运行结果







4 算法分析

4.1 算法时间复杂度分析

初始化 a[]和 b[]数组各需要 O(n)时间，主调度逻辑的时间复杂度为 O(n)，总时间复杂度是 O(n)。

4.2 算法空间复杂度的分析

数组 a[]和 b[]占用 O(n)空间。栈 S 占用 O(n)空间，总空间复杂度是 O(n)。

5 总结与心得

本次实验实现了一个基于栈的车厢调度系统，模拟了列车车厢按照特定顺序通过主轨道的过程。通过栈这一数据结构，我们能够有效地实现车厢的暂时存储和调度，确保列车车厢按照给定的顺序从主轨道出站。栈的特性让我们能够在调

度过程中动态地保存暂时无法按顺序入轨的车厢。栈的操作（入栈和出栈）是常数时间操作，这保证了调度过程的高效性。有些特定的车厢顺序无法在栈中正确调度时，程序会输出“调度无法完成”，表明了栈数据结构在一定条件下的局限性。