组号: 12

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：2023-2023 年冬季

组　　长：孔馨怡

学　　号：22122128

指导教师：朱能军

成绩评定：　　（教师填写）

二〇二三年十二月十八日

小组信息				
登记序号	姓名	学号	贡献比	签名
1	孔馨怡	22122128	33.3%	孔馨怡
2	杨利亚	22122418	33.3%	杨利亚
3	高语涵	22120828	33.3%	高语涵

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	Charter 3----双向链表的应用：发简历
实验二	Charter 4——-队列/栈的应用：车厢调度问题
实验三	(*实验题目*)
实验四	(*实验题目*)

实验二

一、实验题目

车厢调度----问题描述

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）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。

（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）

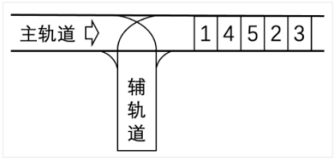


图1“丁”字型铁路调度系统（按指定次序驶出）

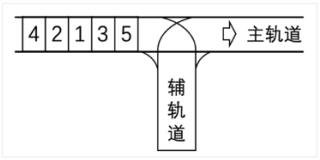


图2“丁”字型铁路调度系统（按从小到大顺序驶出）

二、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。

三、解决方案

1、算法设计（*主要描述数据结构、算法思想、主要操作、用例分析、改进方法等*）

（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 Top 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。Top 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。

（2）算法思想

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）主要操作

Tips：两实验思路相似，此处说明其中一个作为示例

- 创建一个 SeqStack 对象 qa，用于模拟辅助轨道的栈结构。
- 通过 cin 从用户输入中获取车厢数目 n，并初始化变量 No 为 1，表示排在主轨道左边最前面的车厢编号。
- 通过 cout 提示用户输入 n 节车厢的出站顺序，并通过 for 循环依次读取每个车厢的编号 d，进行以下操作：

a. 如果栈顶元素与当前输入的车厢编号匹配，说明辅助轨道中有车厢可以移动到主轨道右边，则输出对应信息，并将该车厢从辅助轨道出栈。

b. 如果当前输入的编号大于等于 No（主轨道左边第一个车厢的编号），则将 No 到 d 之间的车厢依次从主轨道左边移入辅助轨道，并输出相应信息。如果 d 恰好等于 No，则输出该车厢直接从主轨道左边移入主轨道右边。

c. 若主轨道上没有相应编号的车厢，或者顺序不符合要求，则结束循环。

- 最后，根据辅助轨道是否为空，输出调度完成或调度无法完成的信息。

2、源程序代码（*要求有必要注释、格式整齐、命名规范，利于阅读*）

（1）顺序驶入，指定次序开出

```
SeqStack<int> qa;
int n, x, d, No;

cout << "输入车厢数：";
cin >> n;
No = 1;       // No 表示排在主轨道左边最前面的车厢编号，初始值为1.
cout << "输入 " << n << " 节车厢的出站顺序：";
for (int i = 1; i <= n ; i++)
{//一共输入 n 个数据，以表示车厢编号
   cin >> d;
   if(qa.Top(x) == SUCCESS && x == d) {
      //看栈内（副轨道）是否有元素，如果有并且和输入的车厢编号相等，则出栈
      cout << "第 " << x << " 号车厢从辅轨道进入主轨道右边." << endl;
      qa.Pop(x);
   }
   if (No <= d)
   {
      while (No <= n && No < d) {
         //输入的编号大于当前主轨道上第一个车厢对应的编号，则No 号车厢入栈
         cout << "第 " << No << " 号车厢从主轨道左边进入辅轨道." << endl;
         qa.Push(No++);
      }
      if (No == d) {
         //输入的编号恰好等于主轨道上第一个对应的编号，则直接输出
         cout << "第 " << No << " 号车厢从主轨道左边进入主轨道右边." << endl;
         No++;
      }
   }
   else
      break;
}
```

```
if (qa.IsEmpty())
    cout << "调度完成." << endl;
else
    cout << "调度无法完成." << endl;
```

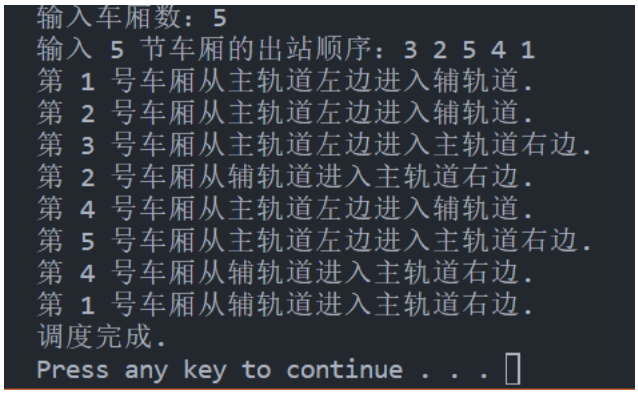
（2）指定次序驶入，顺序开出

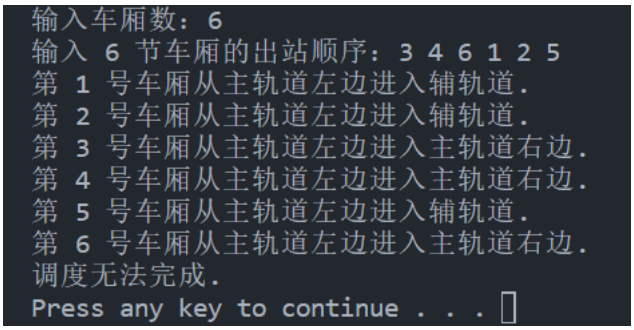
```
SeqStack<int> qa;
int n, x, d;
//4 2 5 3 1，不能完成的调度
cout << "输入车厢数: ";
cin >> n;
cout << "输入 " << n << " 节车厢的入站顺序: ";
for (int i = 1,count=1; i <= n && count<=n; i++,count++){
    cin >> d;

    while (qa.Top(x) == SUCCESS && x == i) {
       cout << "第 " << x << " 号车厢从辅轨道进入主轨道右边." << endl;
       qa.Pop(x);
       i++;
    }
    if (d>i) {
         cout << "第 " << d << " 号车厢从主轨道左边进入辅轨道." << endl;
         qa.Push(d);
         i--;
         continue;
    }
    else  if (d==i) {
         cout << "第 " << d << " 号车厢从主轨道左边进入主轨道右边." << endl;
         while (qa.Top(x) == SUCCESS && x == i+1) {
           cout << "第 " << x << " 号车厢从辅轨道进入主轨道右边." << endl;
           qa.Pop(x);
           i++;
    }
       }
    else
       break;
}
if (qa.IsEmpty())
    cout << "调度完成." << endl;
else
    cout << "调度无法完成." << endl;
```

3、实验结果（*展示实验结果、测试情况、结果分析等*）

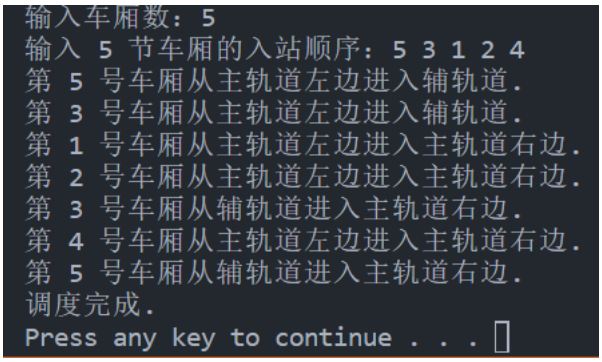
（1）实验结果及测试情况：

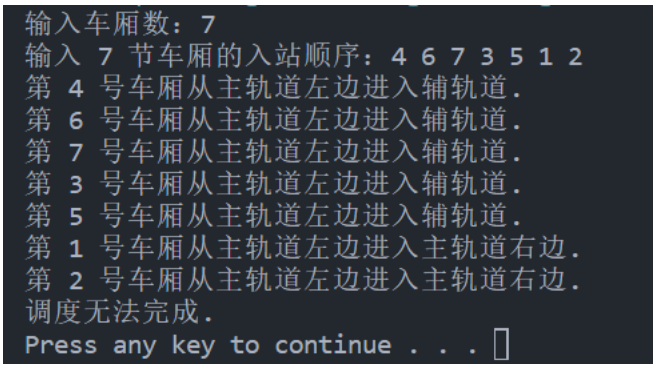
实验一：





实验二：





（2）结果分析及规律总结：

根据下表我们分析主轨道/副轨道中序号顺序来得出结论：

（副轨道/栈中从左到右是栈底到栈顶）

- **实验一：**

  指定输出序列：32541-----调度成功

驶入车厢	1	2	3		4	5		
主轨道			3	32	32	325	3254	32541
副轨道/栈	1	12	12	1	14	14	1	

指定输出序列：346125----调度失败

驶入车厢	1	2	3	4	5	6
主轨道			3	34	34	346
副轨道	1	12	12	12	125	125

可以发现在顺序始入，指定顺序输出时，在每次遍历时，副轨道中从栈顶到栈底的先后顺序必须与输入的指定序列顺序先后一致。

- **实验二：**

指定输入序列：53124----调度成功

驶入车厢	5	3	1	2		4	
主轨道			1	12	123	1234	12345
副轨道	5	53	53	53	5	5	

指定输入序列：4673512----调度失败

驶入车厢	4	6	7	3	5	1	2
主轨道						1	12
副轨道	4	46	467	4673	46735	46735	46735

可以发现在指定顺序始入，顺序输出时，在每次遍历时，副轨道中从栈顶到栈底的先后顺序必须与输出指定序列顺序先后一致（即从栈顶到栈底须为从小到大）。

4、算法分析（*对算法空间、时间效率进行必要分析，可能的改进建议等*）

（1）空间复杂度：

两段代码中都使用了 SeqStack 数据结构来模拟辅助轨道的操作，其空间复杂度取决于输入车厢数 n，因为辅助轨道的大小与车厢数相关。所以，空间复杂度为 O(n)。

（2）时间复杂度：

输入车厢的出/入站顺序是依次进行处理的，循环次数等于车厢数 n。因此，主要消耗时间的操作是 for 循环内的逻辑，操作的时间复杂度是 O(n)。

在循环内部，进行了一系列条件判断、入栈、出栈和输出操作，这些操作的时间复杂度都是 O(1)。

因此，整个代码的时间复杂度是 O(n)。（两实验均是）

（3）改进建议：

可以加入一些错误处理机制，例如检查用户输入的有效性，以及对栈操作的异常情况进行处理，以及异常错误情况的提示。

（涉及到"Assistance.h"库中设置的且在 SeqStack 源代码中涉及到的提示与报错）

5、总结与心得（*主要描述实验过程中存在的问题、原因、解决方法、收获、对实验内容的其他应用思考等*）

对实验内容的其他应用思考：

利用 SeqStack 数据结构的特点，我们可以将其应用于各种需要后进先出特性的场景。

下面是实例：

（1） 逆序输出：将输入的数据逐个入栈，然后再依次出栈，实现逆序输出

（2） 回文判断：将字符串的一半字符入栈，然后与另一半字符进行比较。

（3）任务调度：任务调度涉及在多任务需要同时进行的时候根据优先级和依赖关系来调度任务的执行顺序。可以使用类似的算法来调度任务，并根据任务之间的前后关系进行排序和调度，以确保任务的正确执行顺序。