

Project 1

火车车厢重排调度

班级：教务 2 班

宋渝杰 18340146

廖家源 18340105

缪贝琪 18340131

刘依澜 18340121

【题目要求】

输入一串火车车厢编号，要将火车车厢按编号 1, 2, 3……顺序输出（即出轨），其中可任意添加缓冲轨道，规定重排调度时车厢只能从入轨到缓冲轨道，或者从缓冲轨道到出轨，且要尽量少用缓冲轨道。

【数据结构与算法】

数据结构：

1. stack：一个 int 类型的入轨轨道；若干个 int 类型的缓冲轨道（缓冲轨道在后面经过判断需要时构建）。

2. vector：一个 stack<int>类型的缓冲轨道组，用来存储所有的缓冲轨道；一个出轨轨道 outQueue（用来存储当前应出轨的火车编号顺序）。

算法：

1. 算法及代码解析：

定义一个 int 类型的 stack 叫 wait（即入轨轨道），一个 stack<int>类型的 vector 叫 bufferRoad（即缓冲轨道组），一个 int 类型的 vector 叫 outQueue，定义 int 类型的变量：step 表示调动次数，outQueue[index] 表示下一个该出轨的火车车厢编号，top 表示入轨轨道 wait 的最外面的火车车厢编号，i 用来说明第几条缓冲轨道。

输入混乱的火车车厢编号，通过自定义的 input 函数，确认格式无误后，把输入值压入入轨轨道栈中。

入栈之后，当入轨轨道不为空时，有以下循环：

先在 top 中存入等待被取出的火车车厢编号。分类讨论：

a. 如果 $top == outQueue[index]$ ，即入轨轨道即将被取出的火车编号等于下一个要出轨的火车编号，那么可以将 top 直接出轨，然后步数加 1， $index++$ 。

b. 如果不等，则不能直接出轨，再进行分类讨论：

①先判断已有的缓冲轨道中是否有可以出轨的火车车厢，即判断 $bufferRoad[i].top()$ 是否等于 $outQueue[index]$ ，有则直接通过 pop 进行出轨。

②如果已有的缓冲轨道中没有可以出轨的火车车厢，那么要将 top 转移到缓冲轨道中，先判断该火车车厢是否能够直接进入已有的缓冲轨道（遍历缓冲轨道，若该火车车厢编号小于某缓冲轨道顶部，则将该车厢压入该缓冲轨道中）。若该火车车厢编号大于已有的所有缓冲轨道的顶部编号，那么则要创建新的缓冲轨道来存放该节车厢。

重复以上循环直到入轨的车厢已全部出轨或进入缓冲轨道。

上述步骤完成后，即当入轨的车厢已全部出轨或进入缓冲轨道时，处理缓冲轨道剩下的火车车厢。遍历每一段缓冲轨道的 top 火车编号，判断是否等于 $outQueue[index]$ ，相等则输出。

最后输出总调动步数以及所需要的缓冲轨道数目，然后清除缓存区的数据流，重置标志位。

2. 输入函数：通过自己构造的新函数 `input()` 进行输入——输入混乱的火车编号，判断输入是否合理（不存在字符型），通过 `push_back` 函数先把编号压入轨道 `outQueue`，输入回车键时结束输入，返回 `vector`。该 `vector` 用于判断出轨的车厢编号，后续代码将利用该 `vector` 生成一个 `stack`，代表入轨轨道

3. 异常处理：加入 `cin.good()` 判断输入是否为 `int` 类型的编号：若输入的为字母则 `cin.good()` 返回 0，调用异常处理代码输出“格式有误”，清除内存并重新开始输入。另外，输入格式正确后，通过 `sort` 函数排序后将应该出轨的编号存入 `outQueue` 中，从而对于火车编号重复或不连续的情况，也能进行相对合理的编号调度。

【测试数据、结果及分析】

1. 正确输入的处理：（如图 1）

输入 7，4，8，2，9，3，1，5，6，系统判断输入正确，运行下列调度过程：

6 号车厢不能直接出轨，系统生成 1 号缓冲轨道，将 6 号车厢压入 1 号缓冲轨道；

5 号车厢不能直接出轨，可压入 1 号缓冲轨道；

1 号车厢可以直接出轨；

3 号车厢不能直接出轨，可压入 1 号缓冲轨道；

9 号车厢不能直接出轨，不可压入 1 号缓冲轨道，系统生成 2 号缓冲轨道，将 9 号车厢压入 2 号缓冲轨道；

2 号车厢可以直接出轨；

3 号车厢可以从 1 号缓冲轨道出轨；

8 号车厢不能直接出轨，不可压入 1 号缓冲轨道，可压入 2 号缓冲轨道，系统将 8 号车厢压入 2 号缓冲轨道；

4 号车厢可以直接出轨；

5 号车厢可以从 1 号缓冲轨道出轨；

6 号车厢可以从 1 号缓冲轨道出轨；

7 号车厢可以直接出轨；

8 号车厢可以从 2 号缓冲轨道出轨；

9 号车厢可以从 2 号缓冲轨道出轨；

最后输出调度步数和所需缓冲轨道数量。

```

---请输入一列待调度的火车的编号：
7 4 8 2 9 3 1 5 6
---过程：
6号车厢进入1号缓冲轨道
5号车厢进入1号缓冲轨道
1号车厢直接出轨
3号车厢进入1号缓冲轨道
9号车厢进入2号缓冲轨道
2号车厢直接出轨
3号车厢从1号缓冲轨道出轨
8号车厢进入2号缓冲轨道
4号车厢直接出轨
5号车厢从1号缓冲轨道出轨
6号车厢从1号缓冲轨道出轨
7号车厢直接出轨
8号车厢从2号缓冲轨道出轨
9号车厢从2号缓冲轨道出轨
---调度总步数：14
---需要缓冲轨道数量：2
--按任意键退出...

```

图 1 正确输入的处理

2. 火车编号重复或不连续的情况：（如图 2）

输入 3, 1, 3, 2, 5, 有不连续和重复输入的错误，系统亦可按照 1, 2, 3, 3, 5 的顺序进行正确的调度，调度方式和正常方式类似，此处不再赘述。

```

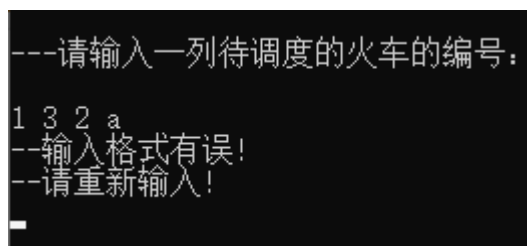
---请输入一列待调度的火车的编号：
3 1 3 2 5
---过程：
5号车厢进入1号缓冲轨道
2号车厢进入1号缓冲轨道
3号车厢进入2号缓冲轨道
1号车厢直接出轨
2号车厢从1号缓冲轨道出轨
3号车厢直接出轨
3号车厢从2号缓冲轨道出轨
5号车厢从1号缓冲轨道出轨
---调度总步数：8
---需要缓冲轨道数量：2
--按任意键退出...

```

图 2 不连续和重复输入的处理

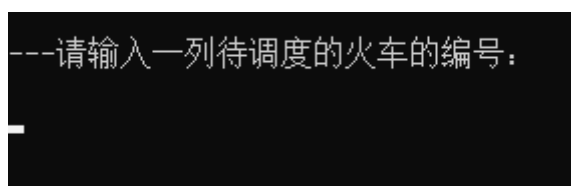
3. 输入字母编号的情况：（如图 3-1，图 3-2）

输入 1, 3, 2, a, 出现字母编号，系统判断输入格式有误，出现图 3-1 的错误提示，经过 0.5 秒后，调用 system 清屏函数，再重新显示输入提示（如图 3-2 所示）。



```
---请输入一列待调度的火车的编号:
1 3 2 a
--输入格式有误!
--请重新输入!
```

图 3-1 字母编号的处理



```
---请输入一列待调度的火车的编号:
```

图 3-2 字母编号处理后的重新显示

【分工、贡献%、自我评分】

分工：

宋渝杰：构思算法及编写代码 30% 10 分

廖家源：构思算法及编写代码 30% 10 分

缪贝琪：编写实验报告 20% 10 分

刘依澜：编写实验报告 20% 10 分

【项目总结】

从这次的小组实验中，我们收获了很多，增强了团结精神，并且从大家身上学习到了很多知识。而在对火车调度问题的模拟过程中，明白了模拟在实际生活中的重要性，并且对 stack 和 vector 有了更深的理解和更灵活的应用。

【程序清单】

```
#include <iostream>
#include <vector>
#include <stack>
```

```

#include <algorithm>
#include <windows.h>
#include <conio.h>
using namespace std;

vector<int> input()//输入函数
{
    bool flag = true; //功能标志参数
    vector<int> tmp;
    while (true){
        if(flag){
            cout << "\n---请输入一列待调度的火车的编号：" << endl<< endl;
            flag=false;
        }
        int n;
        cin >> n;
        if(!cin.good()){
            cin.clear();//读取失败时清空缓冲区
            cin.sync();
            cout << "--输入格式有误!" << endl;
            cout << "--请重新输入!" << endl;
            Sleep(500);    //程序暂停 0.5s
            flag=true;
            tmp.clear();
            system("cls");
            continue;
        }
        tmp.push_back(n);
        if (cin.get() == '\n') break;//输入回车键时结束输入
    }
    return tmp;
}

int main()

```

```

{

vector<stack<int>>> bufferRoad; //缓冲轨道
vector<int> outQueue; //输出队列, 主要用于解决非正常输入, 即输入不完整的 1~n
stack<int> wait; //存储入轨的火车车厢编号
int step = 0; //调度总步数
int index = 0; //结合 outqueue 指定当前应出轨的车厢编号
int top; // 入轨车厢的头部编号
int i; //循环遍历参数

outQueue=input();
for(i=0;i<outQueue.size();i++){ //确定输入数据格式无误后, 将数据入栈
    wait.push(outQueue[i]);
}

sort(outQueue.begin(),outQueue.end()); //指明火车出轨顺序

cout << endl << "---过程:" << endl<< endl;
while (wait.empty() == 0) //当存在入轨的火车车厢时
{
    top = wait.top(); //取出等待的头节火车车厢

    if (top == outQueue[index]) { //判断是否能直接出轨
        cout << "    " << outQueue[index] << "号车厢直接出轨" << endl;
        wait.pop();
        index++;
        step++;
        continue;
    }
    else { //如果不能直接出轨
        for (i = 0; i < bufferRoad.size(); i++) {
            if (bufferRoad[i].empty() == 0 and bufferRoad[i].top() ==
outQueue[index]) { //判断缓冲轨道中是否有车厢可以出轨
                cout << "    " << outQueue[index] << "号车厢从" << i + 1 << "号缓

```

```

    冲轨道出轨" << endl;

        bufferRoad[i].pop();
        index++;
        step++;
        goto th;
    }
}
//如果没有车厢可以出轨
for (i = 0; i < bufferRoad.size(); i++) {
    if (bufferRoad[i].empty() == 1 or top <= bufferRoad[i].top()) { //判
断能否放入已有的缓冲轨道
        bufferRoad[i].push(top);
        cout << "    " << top << "号车厢进入" << i+1 << "号缓冲轨道" << endl;
        wait.pop();
        step++;
        goto th;
    }
}
//如果不能放入任何已有的缓冲轨道，则新建一个新的缓冲轨道
stack<int> newRoad;
newRoad.push(top);
wait.pop();
step++;
bufferRoad.push_back(newRoad);
cout << "    " << top << "号车厢进入" << bufferRoad.size() << "号缓冲轨
道" << endl;
}
th://goto 语句用于直接开始新的循环
}

```

//当入轨的车厢已全部出轨或进入缓冲轨道时，处理缓冲轨道剩下的车厢：

```

for (i = 0; i < bufferRoad.size(); i++) {
    if (bufferRoad[i].empty() == 0 and bufferRoad[i].top() == outQueue[index]) { //
判断哪个缓冲轨道的车厢可以出轨
        cout << "    " << outQueue[index] << "号车厢从" << i + 1 << "号缓冲轨道

```



```

出轨" << endl;
    step++;
    index++;
    bufferRoad[i].pop();
    i = -1;
    continue;
}
}

//出轨过程结束
cout << endl<< "---调度总步数：" << step << endl;
cout << "---需要缓冲轨道数量：" << bufferRoad.size() << endl;
cin.sync();
cin.clear();
cout<<"\n--按任意键退出...";
getch();
}

```