数据结构实验报告

1. 实验要求

i. 实验目的:

（1） 通过上机编程，学会数据结构中栈和队列的使用。

（2） 熟悉 C++语言的基本编程方法，掌握集成编译环境的调试方法，熟练改错方法。

（3） 熟悉设计算法的过程

（4） 进一步掌握指针、模板类、异常处理的使用
（5） 掌握栈的操作的实现方法
（6） 掌握队列的操作的实现方法
（7） 学习使用栈解决实际问题的能力
（8） 学习使用队列解决实际问题的能力

ii. 实验内容：

利用队列结构实现车厢重排问题。车厢重排问题如下：

一列货车共有 n 节车厢，每个车厢都有自己的编号，编号范围从 1~n。给定任意次序的车厢，通过转轨站将车厢编号按顺序重新排成 1~n。转轨站共有 k 个缓冲轨，缓冲轨位于入轨和出轨之间。开始时，车厢从入轨进入缓冲轨，经过缓冲轨的重排后，按 1~n 的顺序进入出轨。缓冲轨按照先进先出方式，编写一个算法，将任意次序的车厢进行重排，输出每个缓冲轨中的车厢编号。

提示：

一列火车的每个车厢按顺序从入轨进入不同缓冲轨，缓冲轨重排后的进入出轨，重新编排成一列货车。比如：编号为 3 的车厢进入缓冲轨 1，则下一个编号小于 3 的车厢则必须进入下一个缓冲轨 2，而编号大于 3 的车厢则进入缓冲轨 1，排在 3 号车厢的后面，这样，出轨的时候才可以按照从小到大的顺序重新编排。

iii. 代码要求：

1、必须要有异常处理，比如删除空链表时需要抛出异常；
2、保持良好的编程的风格：
代码段与段之间要有空行和缩近
标识符名称应该与其代表的意义一致
函数名之前应该添加注释说明该函数的功能
关键代码应说明其功能
3、递归程序注意调用的过程，防止栈溢出

2. 程序分析

火车缓冲问题描述的是一列乱序的火车，经过 K 个缓冲轨之后实现顺序排列。在这个程

序中，我首先联想到了队列具有先入先出的特点，可以将 K 个缓冲轨道设计为 K 条链队列或者循环队列。当前的轨道若为空，则火车的一节车厢可按目前的顺序进入一个，若当前车厢的序号大于后面一节的，若两节车厢在同一缓冲轨则依队列的特点出列是将不能实现从大到小排列，故下一列车应该进入下一个缓冲轨。

每节车厢都按照上述进行判断再安排进入缓冲轨。缓冲轨数量 k 是一个确定数，当算法中所需要的缓冲轨数大于 k 时将无法实现重新排列，故需要提醒设计者多安排几条缓冲轨道。若实际所需轨道数小于 k 则会有空的缓冲轨道。

出轨的时候应按照由小至大的顺序依次出轨，需要在各条缓冲轨道之间查找相应的列车节数。（但是其实可以边入轨，当该车厢在缓冲轨中的后一节车厢入轨后可以直接出轨，不一定要等到全部都入轨再出轨。）

2.1 存储结构

链队列的存储示意图：

缓冲轨道 1:

缓冲轨道 2

``````

缓冲轨道 k：

## 2.2 关键算法分析：

a) 设定并初始化计数器 j=0 ，have=1（这是一个 bool 型变量，用来判断当初循环是否还具有缓冲轨可用。）

b) 用 while 判断是否入轨，用 i<n 判断是否还有车厢需要进入缓冲轨，have 判断是否还有剩余缓冲轨，只有二者同时成立时才可以进行下一步排序。

c) 在循环中现将 have=0；

for 循环中，重新排列车厢使之进入缓冲轨道 ；

(1) 若缓冲轨中最后一个元素比此时将被排序的车厢序号小，

则车厢可以进入该缓冲轨；

排序实现一次，j++；

可以实现入轨，令 have=1.

跳出 for 循环并进入 while 循环.

(2) 若缓冲轨为空，则车厢直接入内，成为第一个元素。

排序实现一次，j++；

可以实现入轨，令 have=1.

跳出 for 循环并进入 while 循环.

（3）判断是否还能继续排轨。

① have=0 无法实现重新排列，故需要提醒设计者多安排几条缓冲轨道。

② have=1 有合适的缓冲轨，则遍历缓冲轨，排列完毕后由小至大输出。

时间复杂度的计算：o（$n^2$）
``````

代码如下：

```
void change(int q[], LinkQueue<int> d[], int n, int k)
{
    int j = 0;              //已经重排车厢的数目

    bool have = 1;              //还有缓冲轨道可以使用

    while ((j<n) && (have))          //当入轨中有车厢时
    {
        have = 0;                  //在后面的程序中改变have的值，若不变则说明没有排这节车厢
        for (int i = 0; i<k; i++)
        {
            if (d[i].GetRear()<q[j])   //若所需加入的数据大于目前缓冲轨道中的数据就可以在此缓冲轨道加入
            {
                d[i].inQ(q[j]);
                j++;
                have = 1;
                break;
            }
        }
    }
    if (have == 0)                          //说明没有办法排这节车厢
    {
        cout << "车厢重排失败，请试试安排更多的缓冲轨道" << endl;
    }
    else                                    //全部n节车厢都重排完毕
    {
        for (int l = 1; l<k + 1; l++)
        {
            cout << "序号为" << l << "的缓冲轨道停有";
            d[l - 1].read();
            cout << endl;
        }
      }
     }
```

2.3 其他

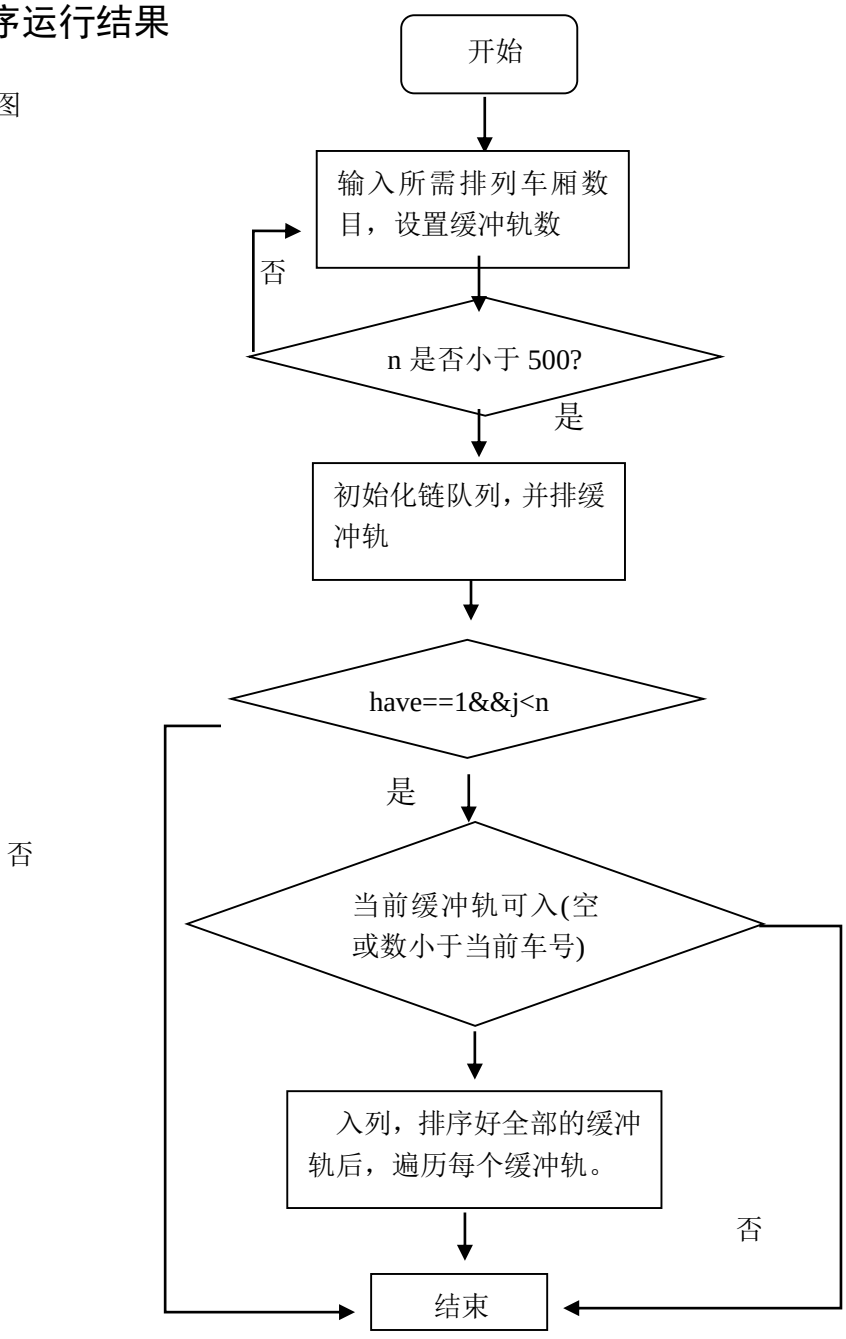
关于链队列的基本操作：

1. 构造链队列：在本程序中，先使用构造函数将链队列初始化为空队列。
   rear=front=Null。
2. 使用尾插法将队列赋予值。
   工作指针p指向一个新结点，然后p->next=Null；
3. 为了检查每个缓冲轨是否有存入正确的车厢序号，设计了一个遍历函数，将每个队列中

的元素遍历出来。

3. 程序运行结果

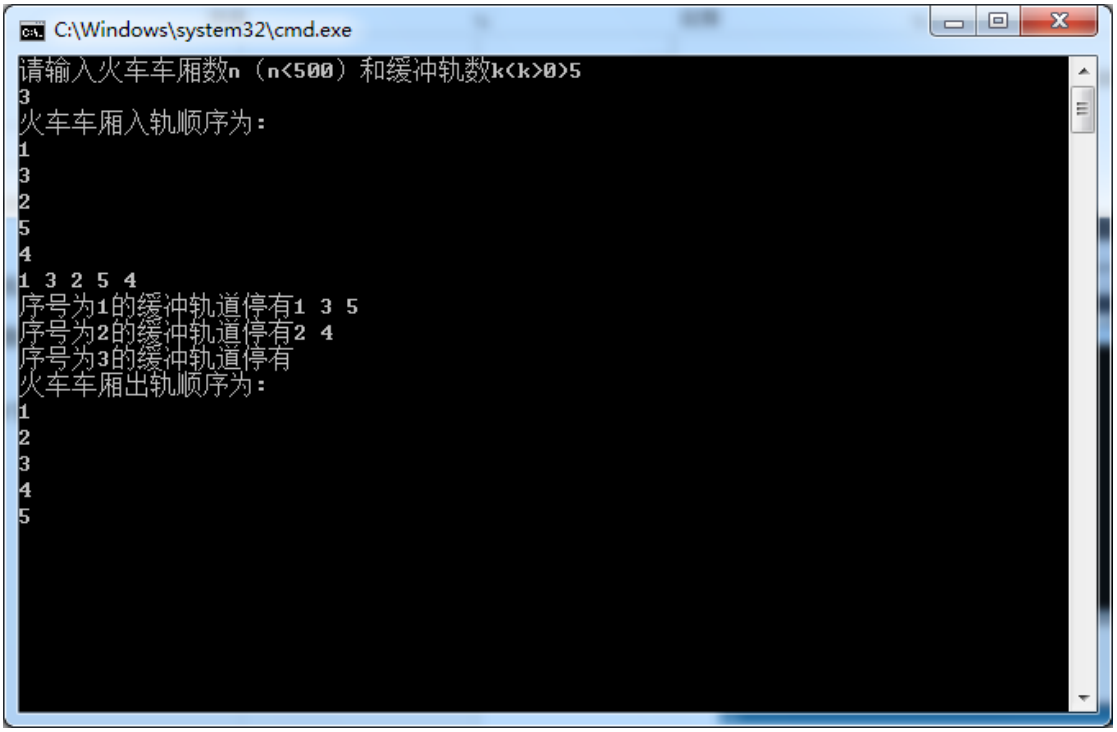
1. 流程图



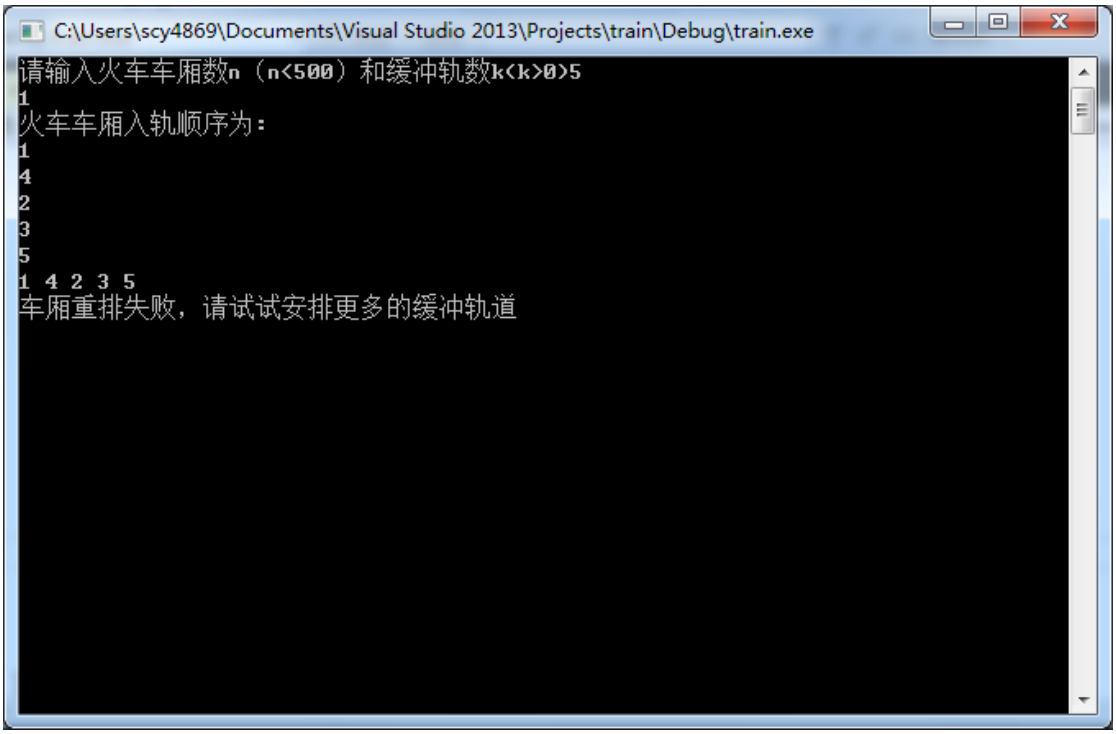
2. 测试条件：输入列车的车厢数目小于 500，缓冲轨的设定要合理

3. 测试结果：

缓冲轨设置合理时：



缓冲轨设置不够合理时：



4. 总结：

通过这次实验，我对栈和队列有了形象一点更为具体的认识，熟悉了模板类的使用。

在调试中出现的问题及改正：

1. 在编写程序时，格式不对，实在太粗心，经过调试修正如下：

```
T GetFront(){
    if (front != NULL) return front->data;}
```

2. 编写程序时，不清楚如何安排入列的判断，后面经过学习，发现可以使用与非门的思想，设置一个控制量have来实行判断。

3. 全部入列再输出的过程中出现格式错误，输出错误等情况，经过多次调试成功运行的代码如下：

```
#include<iostream>
#include<ctime>
using namespace std;
const int size = 500;
template<class T>
struct Node
{
    T data;
    Node<T> *next;
};

template<class T>
class LinkQueue                          //链队列模板类
{
public:
    LinkQueue();                         //构造函数
    ~LinkQueue();                        //析构函数
    void read();                         //遍历缓冲轨
    void inQ(T x);                       //进入队列
    T outQ();                            //离开队列
    T GetFront(){
        if (front != NULL) return front->data;
    }
    T GetRear()
    {
        if (rear != NULL) return rear->data;
        else return 0;
    }                                    //查找队尾元素
    bool Empty() { front == NULL ? return 1 : return 0; }      //判断队列是否为空
    friend void change(int q[], LinkQueue<T> d[], int n, int k);
private:
    Node<T> *front, *rear;
};

template<class T>
LinkQueue<T>::LinkQueue()
```

```
{
	front = rear = NULL;
}
template<class T>
LinkQueue<T>::~LinkQueue()
{
	while (front)
	{
		rear = front->next;
		delete front;
		front = rear;
	}
}
template<class T>
void LinkQueue<T>::inQ(T x)//尾插入
{

	if (front == NULL){
		Node<T> *p = new Node<T>;
		rear = front = p;
		p->data = x;
		p->next = NULL;
	}
	else
	{
		Node<T> *p = new Node<T>;
		rear->next = p;
		p->next = NULL;
		p->data = x;
		rear = p;
	}
}
template<class T>
T LinkQueue<T>::outQ()
{
	Node<T> * p = front;
	if (p) throw"空链表";

	//if (!p) throw"下溢";
	T x = p->data;
	if (p->next == NULL) front = NULL;
	delete p;
	return x;
}
```

```
template<class T>
void LinkQueue<T>::read()
{
    Node<T> *p = front;
    while (p)
    {
        cout << p->data << " ";
        p = p->next;
    }
}
void change(int q[], LinkQueue<int> d[], int n, int k)
{
    int j = 0;                    //已经重排车厢的数目

    bool have = 1;                //还有缓冲轨道可以使用

    while ((j<n) && (have))       //当入轨中有车厢时
    {
        have = 0;                 //在后面的程序中改变have的值，若不变则说明没有排这节车厢
        for (int i = 0; i<k; i++)
        {
            if (d[i].GetRear()<q[j])          //若所需加入的数据大于目前缓冲轨道中的数据就可以在此缓冲轨道加入
            {
                d[i].inQ(q[j]);
                j++;
                have = 1;
                break;
            }
        }
    }
    if (have == 0)                //说明没有办法排这节车厢
    {
        cout << "车厢重排失败，请试试安排更多的缓冲轨道" << endl;
    }
    else                          //全部n节车厢都重排完毕
    {
        for (int l = 1; l<k + 1; l++)
        {
            cout << "序号为" << l << "的缓冲轨道停有";
            d[l - 1].read();
            cout << endl;
```

```
            }
            cout << "火车车厢出轨顺序为:" << endl;
            int s = 1;//计数每个缓冲轨道的头值
            int g;
            while (s != n + 1){
                for (g = 0; g<k; g++)
                {
                    if (d[g].GetFront() == s)
                    {
                        s = d[g].GetRear() + 1;//输出尾部，以便下次头值
                        break;
                    }
                }
                d[g].read();
            }
        }
}

void main()
{

    int n, k;
    cout << "请输入火车车厢数n（n<500）和缓冲轨数k(k>0)";
    cin >> n >> k;
    cout << "火车车厢入轨顺序为:" << endl;
    int *series = new int[n];
    for (int p = 0; p < n; p++)
        cin >> series[p];
    for (int m = 0; m < n; m++)
        cout << series[m] << " ";
    cout << endl;
    LinkQueue<int> *turn;
    turn = new LinkQueue<int>[k];
    change(series, turn, n, k);
    cin.get();
    cin.get();
    cin.get();
    cin.get();
    cin.get();
    cin.get();
}
```