|  |
| --- |
| 哈尔滨工业大学(深圳) |
| **《数据结构》实验报告** |
|  |
| 实验一  线性结构及其应用 |

# 问题分析

（1）题目要求收集两个班的学生成绩信息并且按照降序排列，其中每个学生的数据包括学生姓名和成绩，需要用一个结构体作为一个整体存储。由于数据是可以不断添加进来的，可以用两个线性表来分别存储两个班级学生的信息。但如果用数组等顺序存储方式来存储数据，每存储一位学生的信息都需要进行大量的移动操作，并且由于学生数量未知，用顺序存储可能造成资源的浪费。这时利用链表的插入和删除操作时间复杂度仅为O（1）的特点，并且可以根据实际情况动态扩展内存，可以选择线性链表为存储结构。由于输入学生信息是是按成绩升序输入，则只需利用头插法将学生信息插入链表的表头即可完成按成绩降序储存。

（2）题目要求将学生信息按照学生成绩升序重新排列。由于原先的链表已经依照分数降序，将两个链表使用就地反转的方法，即可按照升序将链表进行反转。由于就地反转只需在遍历链表时进行操作，所以时间复杂度仅为O(n)。

（3）题目要求从两个后半部分交叉在一起的链表之中找到第一个相交节点。由于两个链表的长度未知，则首先分别遍历两个链表来获得链表的长度，然后将链表的尾部对齐后，对两个链表的每一位进行比较。这样操作同样也是在遍历链表时进行操作，可得时间复杂度为O(n)。

# 二、详细设计

## 2.1 设计思想

（1）首先建立一个不含表头的线性链表，每当输入输入一个学生数据时，新建一个链表节点，每个节点包含学生的两个信息和一个指向下一个节点的指针。将学生数据储存在此节点后，将此节点插入链表的头部。两个班级的链表分别用两个头指针管理。

（2）对于给定的链表，将链表的第一个节点的后驱指向NULL。从第二个节点开始，用一个指针p记录此节点的前驱，一个指针q记录此节点，用s记录此节点的后驱。将此节点的后驱指向上一个节点（即此节点的前驱），将p，q，s分别向后移一位，直到q指向的节点为空，再将q定义为头节点。

（3）对于给定的两个链表头节点，首先遍历两个链表，得到两个链表的长度，用m、n记录。先将两个指针p、q指向两个链表。若m>n，则将p向后移一位，m减一；若n>m，则将q向后移一位，直到m等于n。（此操作将两个链表的尾部对齐）再判断p、q所指向的节点的值是否相等，若不相等，则将p、q移向下一位，再次进行判断，直到p、q所指向的节点的值相等。返回此节点的值即为第一个相交点。

## 2.2 存储结构及操作

存储结构：

(1) 存储结构（一般为自定义的数据类型，比如单链表，栈等。）

链表节点结构体StudentLinkedListNode

内容包括学生成绩Grade（int）,学生姓名(char)，指向结构体StudentLinkedListNode的指针next。

(2) 涉及的操作（一般为自定义函数，可不写过程，但要注明该函数的含义。）

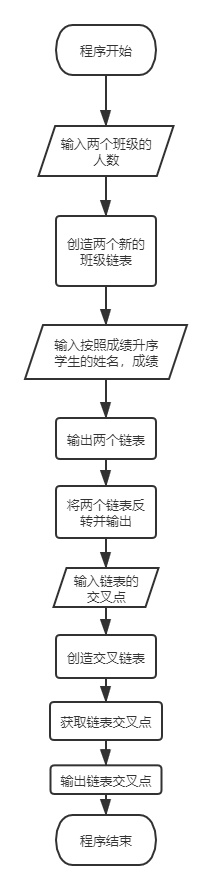
操作：

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 参数 | 功能 | 返回值 |
| printLinkedListNode | 链表节点指针 | 打印单个节点 |  |
| outputStudentLinkedList | 链表头指针 | 输出该表的成绩情况 |  |
| studentLinkedListCreate | 学生姓名，成绩 | 新建一个链表node并返回 | 指向新节点的指针 |
| studentLinkedListAdd | 链表头指针，要插入的链表节点指针 | 按照降序插入学生的成绩情况,并返回链表头指针 | 链表头指针 |
| reverseLinkedList | 链表头指针 | 反转链表 | 反转后的链表头指针 |
| findCrossBeginNode | 两个链表的头指针 | 找到相交的第一个结点 | 两个链表的第一个相交节点 |
| Getlen | 链表头指针 | 获取链表的长度 | 链表长度 |

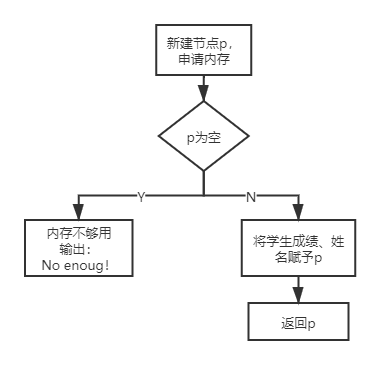
## 2.3 程序整体流程

画出整体流程，及核心算法流程。

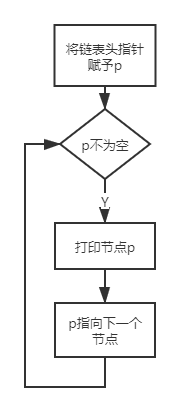
整体流程：



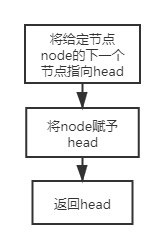
创造新链表节点：



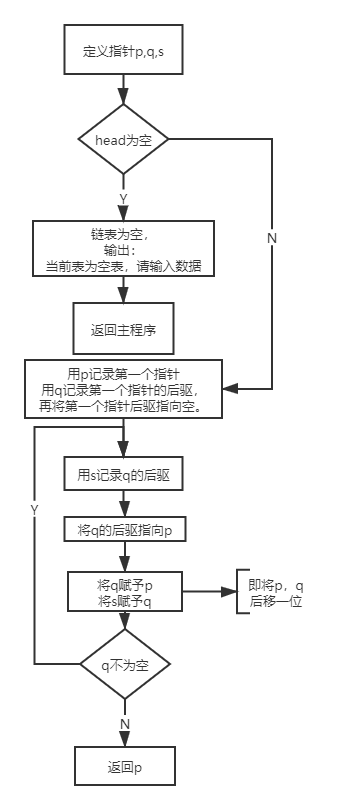
输出该表的成绩情况：



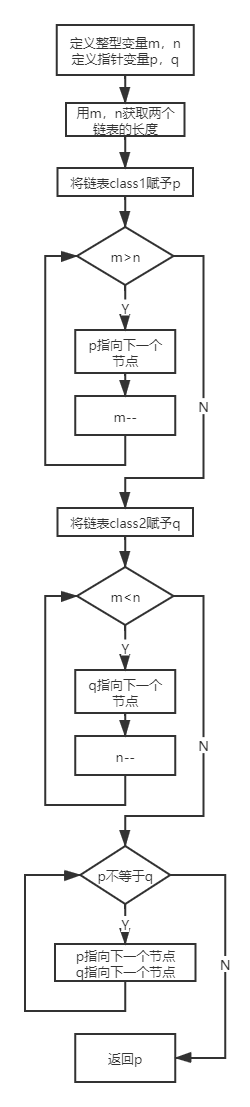
按照降序插入学生的成绩情况,并返回链表头指针：



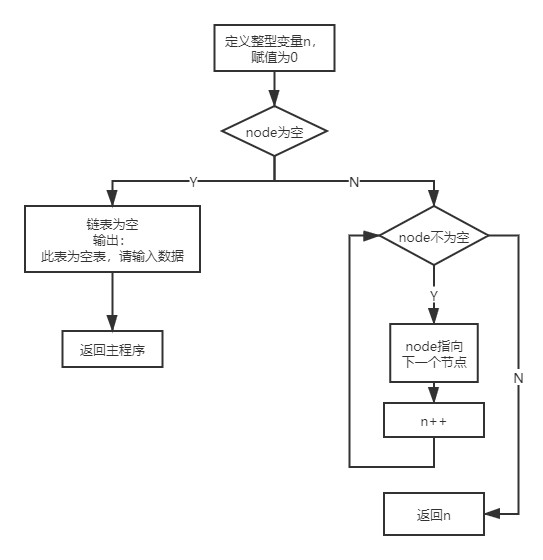
反转链表：



找到第一个相交节点：



获取链表长度：



# 三、用户手册

如：(1)输入数据的方式；(2)实现各种功能的操作方式等。

1. 进入程序后，首先输入两个班级的人数

如：10 11

1. 输入班级人数后，按照车经过几代升序依次输入学生的姓名与成绩

如：student1 43

student5 58

student4 61

student2 77

student3 86

student6 87

student7 95

student9 96

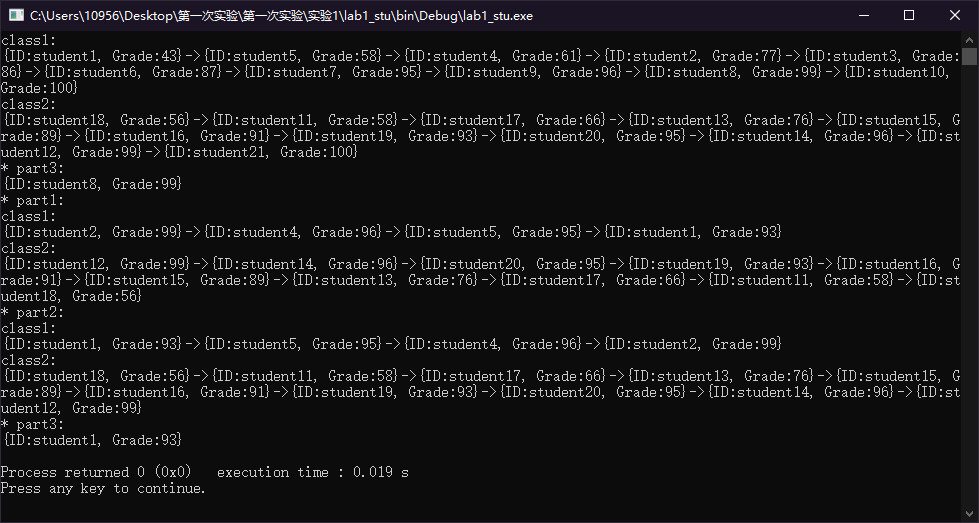
student8 99

1. 输入两个链表交叉的位置

如：8 9

# 四、结果

程序正确运行的结果截图。



# 五、总结

该实验涉及到的数据结构和算法，以及遇到的问题和收获。

数据结构：线性链表

算法：

1、链表的头插法

2、链表的就地反转

3、链表的尾部对齐

遇到的问题：

1、链表无头指针，因为忘记了这一点导致程序设计出错

2、链表判断相等以及判断链表是否到尾部时漏了一项

收获：

1、练习了线性链表的建立、插入等操作，熟悉了链表的使用

2、复习了Debug的流程

实验二

栈与队列的应用

**一、问题分析**

（1）通过数组来实现栈的基本操作：Push（将元素压入栈顶）、Pop（获取并移除栈顶元素）、GetTOp（获取栈顶元素）、StackEmpty（判断栈顶是否为空）

（2）通过（1）中实现的栈来实现队列的基本操作：EnQueue（将元素插入队列尾）、DeQueue（获取并且移除队列头的元素）、GetHead（获取开头的元素）、QueueEmpty（判断队列是否为空）

**二、详细设计**

**2.1设计思想**

（1）设置一个记号top记录栈顶位置，top原始值设定为-1以便与数组对齐，一个数组seq[]。seq[top]为栈顶元素。当元素压入栈顶时，top+1，seq[top] 等于该元素。当执行Pop时，用e返回seq[top]，将seq[top]置为零，top-1。GetTop用e返回seq[top]实现。执行StackEmpty时，若top=-1，则栈为空，否则栈不为空。

（2）建立两个栈stack1、stack2，当有元素进队列时，直接将其压入栈stack1。当有元素要出队列时，将stack1中除了栈底元素之外的元素依次弹出，压入stack2，再将栈底元素弹出，用e返回值。最后将stack2中的元素依次弹出，压入到stack1。获取队列头元素时，将stack1中除了栈底元素之外的元素依次弹出，压入stack2，再用e返回栈底元素的值。最后将stack2中的元素依次弹出，压入到stack1。获取队列的一个拷贝时，先将stack1中的元素依次弹出，压入stack2。这时将stack2的元素依次弹出即可。

**2.2存储结构及操作**

存储结构：

(1)栈的节点结构体

包含：栈顶指针（int）、储存数据的数组（Datatype）

(2)队列的节点结构体

包含：两个栈节点stack1,stack2

操作：

(1)栈：

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 参数 | 功能 | 返回值 |
| Push | 栈节点指针，操作数 | 向栈插入一个元素 | 布尔值（是否操作成功） |
| Pop | 栈节点指针，接受弹出栈值的变量指针 | 从栈中弹出一个元素 | 布尔值（是否操作成功） |
| GetTop | 栈节点，接受栈顶元素值的变量指针 | 获取栈顶元素，不删除该元素 | 布尔值（是否操作成功） |
| StackEmpty | 栈节点 | 判断栈是否空 | 布尔值（栈是否为空） |

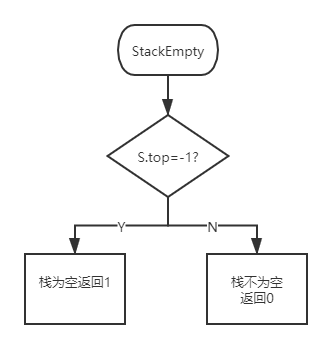
(2)队列：

|  |  |  |  |
| --- | --- | --- | --- |
| 函数名 | 参数 | 功能 | 返回值 |
| QueueEmpty | 队列节点 | 判断队列是否为空 | 布尔值（队列是否为空） |
| EnQueue | 队列节点指针，插入元素 | 将元素插入队列 | 布尔值（操作是否成功） |
| DeQueue | 队列节点指针，接受出队元素的变量指针 | 从队列中取出一个元素 | 布尔值（操作是否成功） |
| GetHead | 队列节点，接受队列头元素的变量指针 | 获取队列头 | 布尔值（操作是否成功） |
| QueueToArray | 队列节点，数组地址 | 获取队列的一个数组拷贝，顺序为从队列头到队列尾 | 无 |

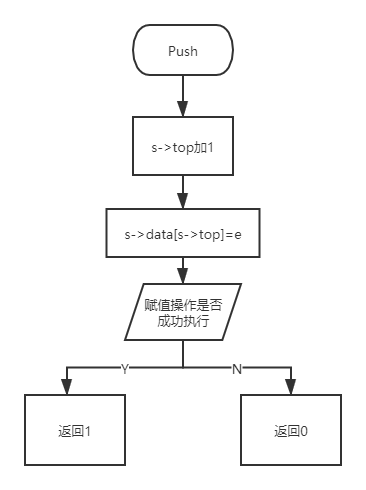
**2.3程序整体流程**

1、栈

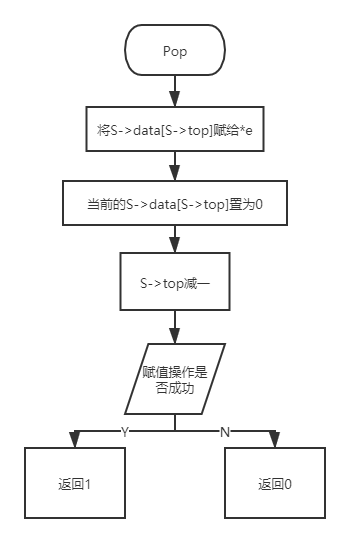
(1)StackEmpty:



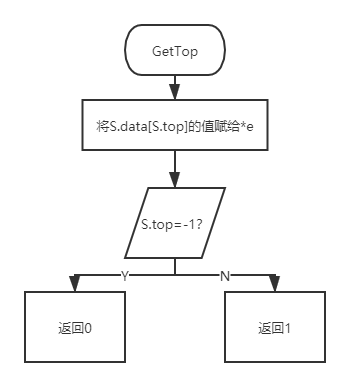
(2)Push



(3)Pop

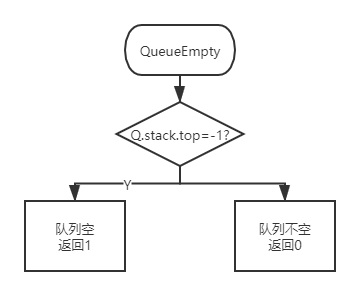


(4)GetTop

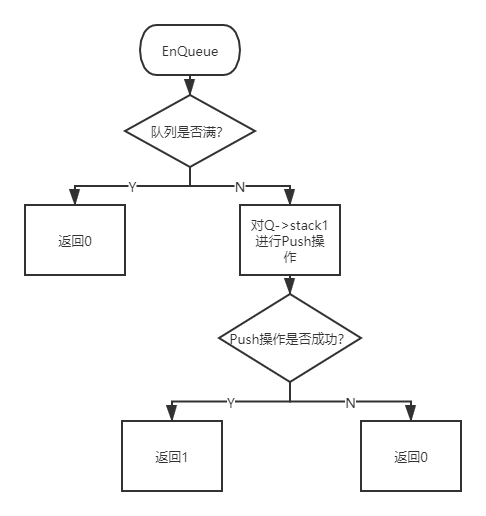


2、队列

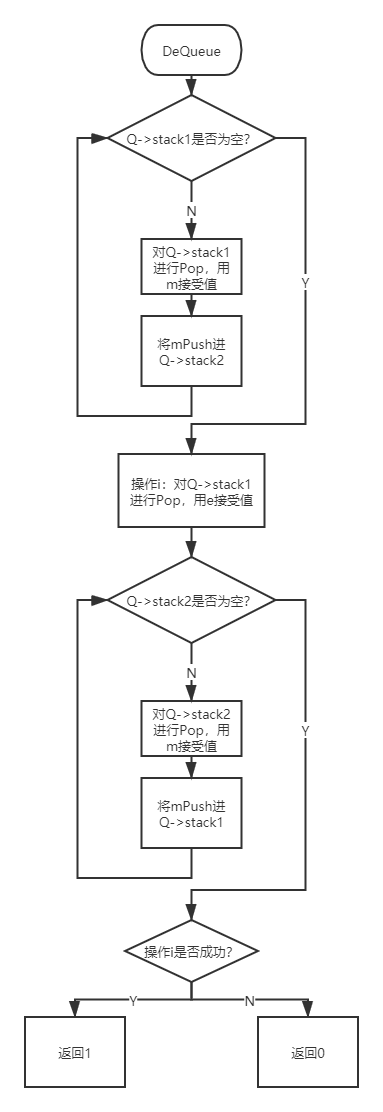
(1)QueueEmpty



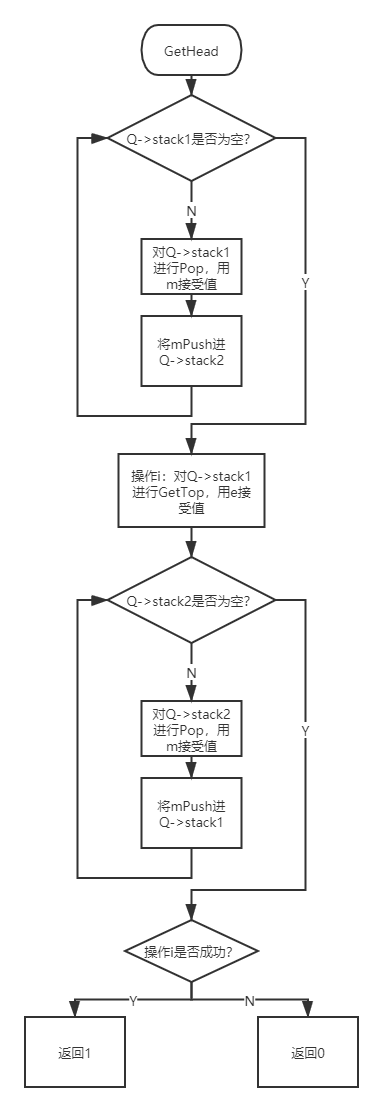
(2)EnQueue



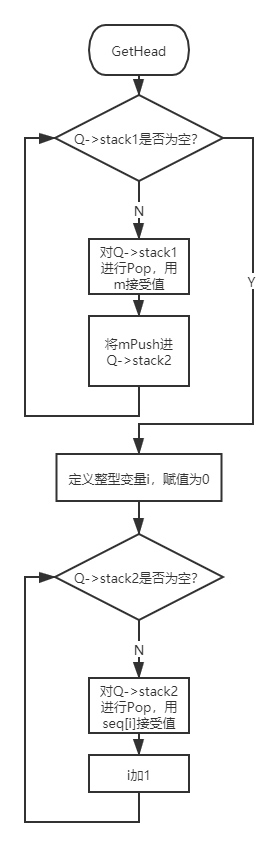
(3)DeQueue



(4)GetHead



(5)QueueToArrey



**三、用户手册**

程序接受数据的方式时从一个名为“test\_stu.in”中的程序读取

输入格式如下：

(1)输入0，进行Push操作

此时需输入多个正整数作为输入栈的数据，输入小于等于0的整数后结束输入。此后会打印出压入栈中的数据。

(2)输入1，进行Pop操作

此时无需输入数据，程序将栈顶元素成功弹出后，将其打印。若Pop操作失败，打印失败提示。

(3)输入2，进行GetTop操作

此时无需输入数据，程序获取栈顶元素后将其打印。若GetTop操作失败，打印失败提示。

(4)输入3，进行StackEmpty操作

此时需输入数据，若栈为空，打印栈为空提示；若栈不为空，打印栈不为空提示。

(5)输入4，进行EnQueue操作

此时需输入多个正整数作为输入队列的数据，输入小于等于0的整数后结束输入。此后会打印出输入队列中的数据。

(6)输入5，进行DeQueue操作

此时无需输入数据，程序将队列头元素成功取出后，将其打印。若DeQueue操作失败，打印失败提示。

(7)输入6，进行GetHead操作

此时无需输入数据，程序获取队列头元素后将其打印。若GetHead操作失败，打印失败提示。

(8)输入7，进行QueueEmpty操作

此时需输入数据，若队列为空，打印队列为空提示；若队列不为空，打印队列不为空提示。

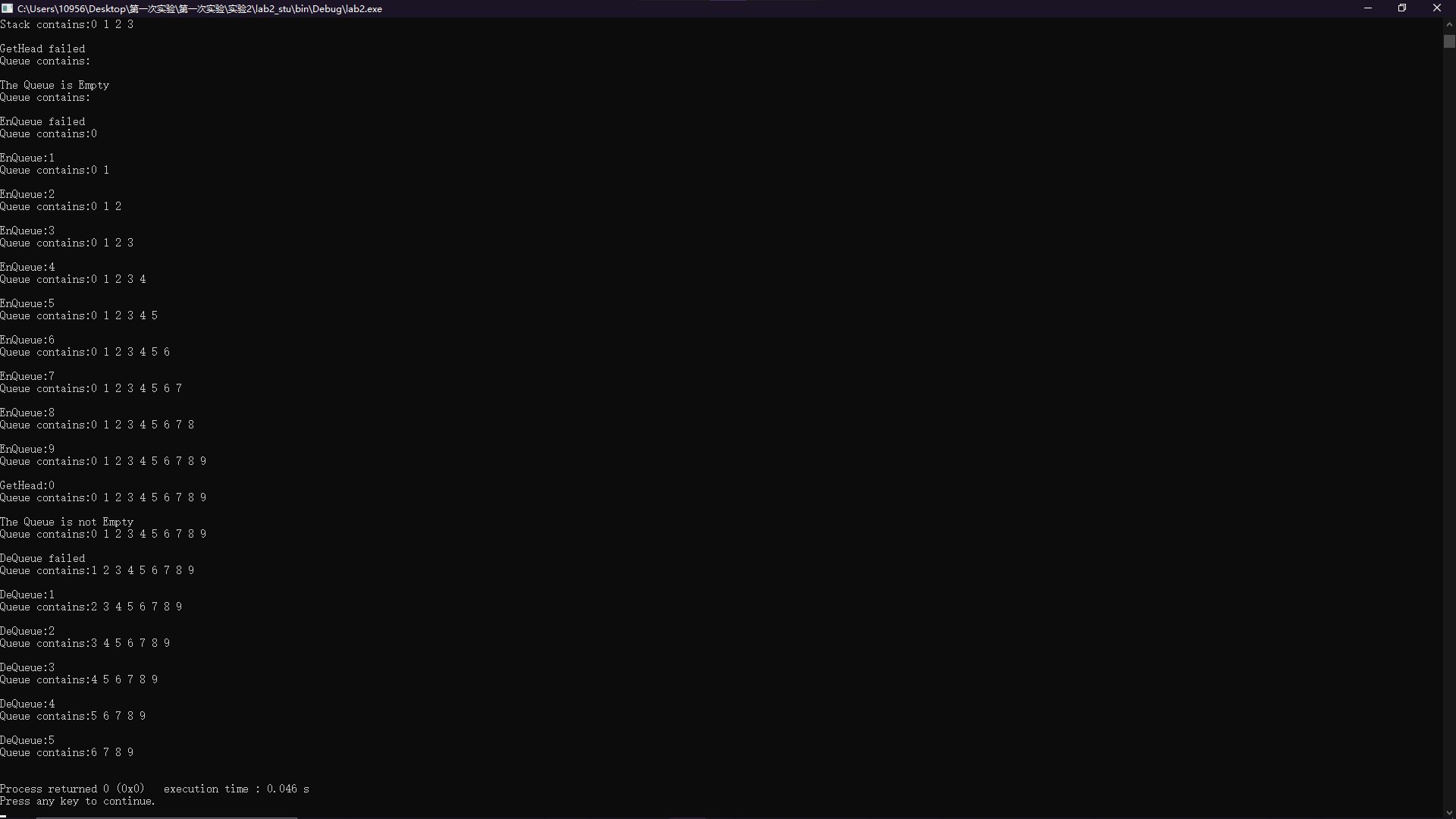
(9)输入-1，结束程序

注：若输入其他值，无效，需重新输入。

程序从头开始读取，运行程序。直到读到文件末尾或者在除了进行Push和EnQueue时读到-1时，终止程序

**四、结果**

****

****

**五、总结**

数据结构：队列，数组，栈

算法：

1. 设置头标志top，用seq[top]代表栈顶，从而实现了用数组实现栈。
2. 设置两个栈stack1，stack2，stack1作为储存栈，stack2作为辅助栈。stack1的栈底作为队列头，栈顶作为队列尾。当对队列头进行操作时，先将stack1中除了栈底元素外其他元素依次弹出，压入stack2中，再对栈底元素进行Pop或者GetTop。从而用栈的LIFO实现了队列的FIFO。

遇到的问题：

进行队列程序的编写时，没有注意哪些变量是指针变量，导致编写错误

收获：

1. 学会了用数组实现栈，用栈实现队列
2. 学会了用ProcessOn绘制流程图