# 

# 《计算机实践》

# 实验报告I—数据结构

班号：0321501 学号：032150115

姓名：易博天

Email：yiboxiaotian@nuaa.edu.cn

签名：

南京航空航天大学

## 实验2：栈和队列的实验

### 一、实验目的

　　掌握栈和队列的顺序存储结构和链式存储结构的相关操作实现。

### 二、实验内容

实现顺序栈和链栈的定义、创建、入栈和出栈操作。

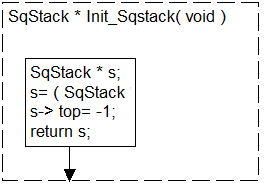
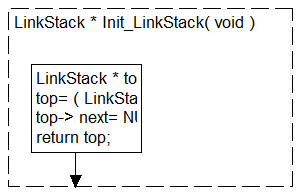
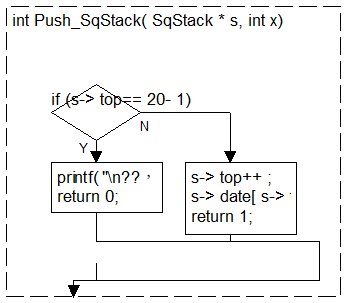
（1）初始化顺序栈，并创建顺序栈的初始序列；实现顺序栈的插入和删除操作，并输出操作后的序列； （2）初始化链栈，并创建链栈的初始序列；实现链栈的插入和删除操作，并输出操作后的序列。

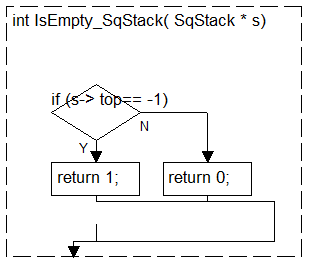
### 三、算法分析与程序设计

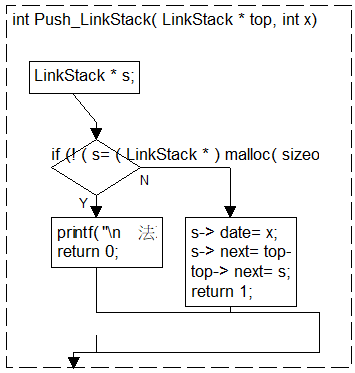
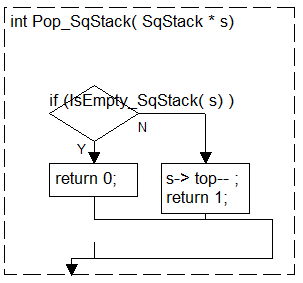
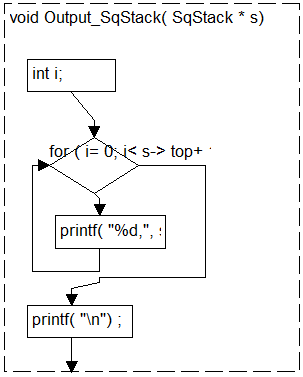
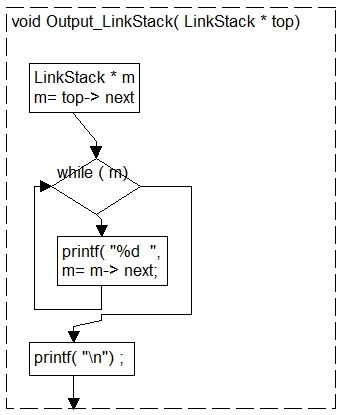
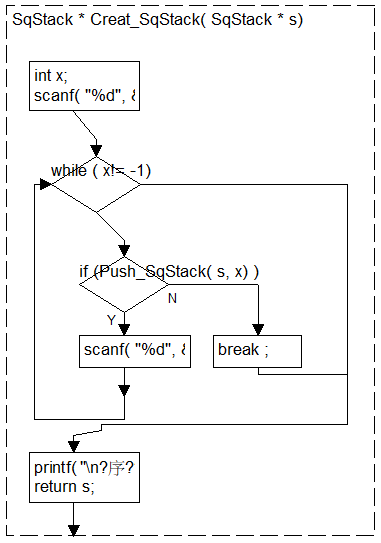
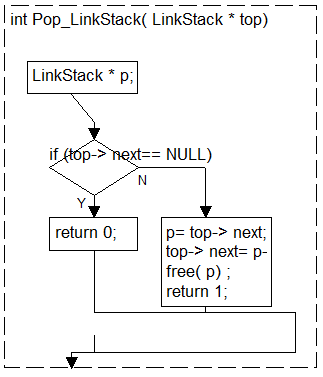
1.输入数据：建立输入处理，按栈里数据的逻辑顺序输入数据。

2.输出形式：分别输出栈里元素的初始序列以及进行入栈和出栈操作以后的正确序列。

3.算法描述

给出算法描述的程序流程图





4. 程序设计

给出实现程序功能的函数及说明如下：

(1) 本程序包含的函数

主函数 main()

SqStack \*Init\_Sqstack(void); // 初始化函数，建立顺序空栈

LinkStack \*Init\_LinkStack(void); // 初始化函数，建立空链栈

int IsEmpty\_SqStack(SqStack \*s); // 判空函数

int Push\_SqStack(SqStack \*s, int x); // 压栈函数 顺序栈

int Push\_LinkStack(LinkStack \*top, int x); // 压栈函数 链栈

SqStack \*Creat\_SqStack(SqStack \*s); // 建立初始序列

LinkStack \*Creat\_LinkStack(LinkStack \*top);

int Pop\_SqStack(SqStack \*s); // 出栈函数 顺序栈

int Pop\_LinkStack(LinkStack \*top); // 出栈函数 链栈

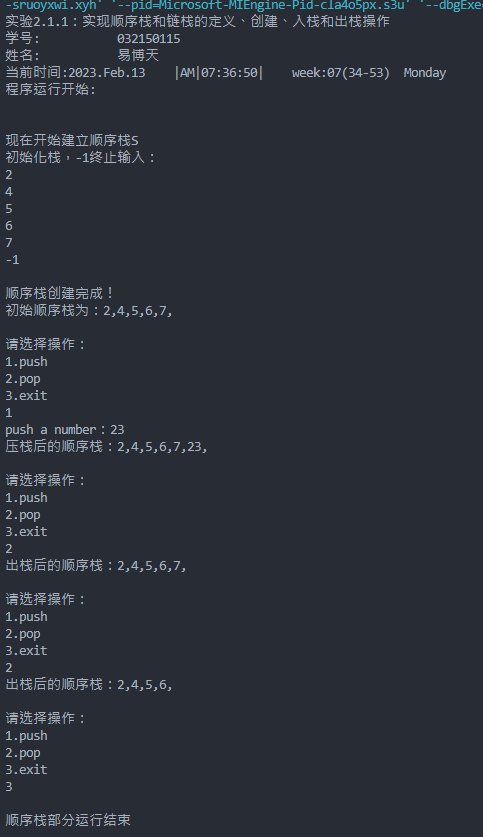
void Output\_SqStack(SqStack \*s); // 输出函数 顺序栈

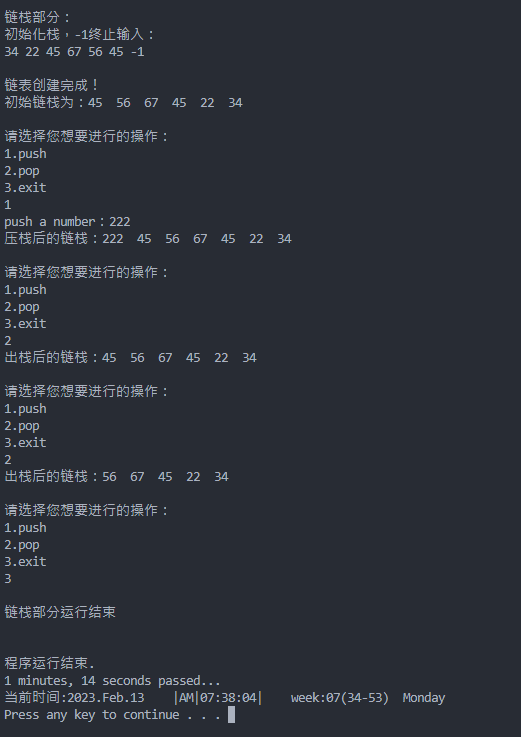
void Output\_LinkStack(LinkStack \*top); // 输出函数 链栈址

### 四、调试分析

#### 1、调试过程及说明

* 以截屏图像的方式给出，包括程序运行开始和结束时的系统时间、姓名、学号、实验名称等信息.





#### 2、调试分析

1.在init函数中初始化将顺序表置零过程中因为重用cnt而导致cnt在总是等于数组大小，在加入true\_cnt暂存之后解决了这个问题。

2.通过刷新输入区函数fflush解决了scanf异常输入的问题

### 五、使用说明

按照程序运行过程中给出的提示输入即可，此程序使用-1作为顺序表结束的标志，也可以用其他字符或者条件替换

创建、插入、删除功能是顺序执行的，只是为了测试对应功能

**六、测试结果**

内容：给出测试用例测试程序。写出测试的情况，使用的输入以及得到的结果。

要求：测试的情况越全面越好。

示例：

测试例子如上图所示

**七、实验的收获和感想**

使用符合顺序表结构的算法实现能大大简化代码，双向链表虽然通用，但实现功能还是尽可能简洁的好。

**八、附录：源代码（双栏排版，加必要的注释）**

#include "homework.h"

#define MaxSize 20

typedef struct stacktype

{

int date[MaxSize];

int top;

} SqStack;

typedef struct lstacktype

{

int date;

struct lstacktype \*next;

} LinkStack;

SqStack \*Init\_Sqstack(void); // 初始化函数，建立顺序空栈

LinkStack \*Init\_LinkStack(void); // 初始化函数，建立空链栈

int IsEmpty\_SqStack(SqStack \*s); // 判空函数

int Push\_SqStack(SqStack \*s, int x); // 压栈函数 顺序栈

int Push\_LinkStack(LinkStack \*top, int x); // 压栈函数 链栈

SqStack \*Creat\_SqStack(SqStack \*s); // 建立初始序列

LinkStack \*Creat\_LinkStack(LinkStack \*top);

int Pop\_SqStack(SqStack \*s); // 出栈函数 顺序栈

int Pop\_LinkStack(LinkStack \*top); // 出栈函数 链栈

void Output\_SqStack(SqStack \*s); // 输出函数 顺序栈

void Output\_LinkStack(LinkStack \*top); // 输出函数 链栈

int main()

{

SqStack \*S;

LinkStack \*LS;

int x, i;

puts("实验2.1.1：实现顺序栈和链栈的定义、创建、入栈和出栈操作");

homework\_start();

printf("\n现在开始建立顺序栈S\n");

S = Init\_Sqstack();

printf("初始化栈，-1终止输入：\n");

S = Creat\_SqStack(S);

printf("初始顺序栈为：");

Output\_SqStack(S);

for (;;)

{

printf("\n请选择操作：\n");

printf("1.push\n");

printf("2.pop\n");

printf("3.exit\n");

scanf("%d", &i);

switch (i)

{

case 1:

printf("push a number：");

scanf("%d", &x);

Push\_SqStack(S, x);

printf("压栈后的顺序栈：");

Output\_SqStack(S);

break;

case 2:

Pop\_SqStack(S);

printf("出栈后的顺序栈：");

Output\_SqStack(S);

break;

case 3:

break;

}

if (i == 3)

break;

}

printf("\n顺序栈部分运行结束\n");

printf("\n链栈部分：\n");

LS = Init\_LinkStack();

printf("初始化栈，-1终止输入：\n");

LS = Creat\_LinkStack(LS);

printf("初始链栈为：");

Output\_LinkStack(LS); // 因为下面所述的原因，这里从初始端（指向另一节点的一端）开始插入，从初始端读取就是栈顺序（FILO），而从另一端开始就是队列顺序（需要双向链表）（FIFO）

// 队列实现可以以标记和传递尾指针实现（固定的尾指针与NULL之间插入）

// 用链表实现可以分为以链表头做队列头，也可以向上面那样以链尾做头（其实双向也就无所谓头尾了）

// 固定一端的单链表就能实现一种顺序。因此也只需要在一端操作。固定一端以标记一端的办法实现。栈的头自然可以充当标记。队的尾需要另设标记。

for (;;)

{

printf("\n请选择您想要进行的操作：\n");

printf("1.push\n");

printf("2.pop\n");

printf("3.exit\n");

scanf("%d", &i); // 这里用的是一种插入算法，因为只需要从一端开始读取，所以只要在最初两个连接的节点中把握一端插入即可，单链表即可实现

switch (i)

{

case 1:

printf("push a number：");

scanf("%d", &x);

Push\_LinkStack(LS, x);

printf("压栈后的链栈：");

Output\_LinkStack(LS);

break;

case 2:

Pop\_LinkStack(LS);

printf("出栈后的链栈：");

Output\_LinkStack(LS);

break;

case 3:

break;

}

if (i == 3)

break;

}

printf("\n链栈部分运行结束\n");

homework\_done();

return 0;

}

SqStack \*Init\_Sqstack(void) // 初始化函数，建立顺序空栈

{

SqStack \*s;

s = (SqStack \*)malloc(sizeof(SqStack));

s->top = -1;

return s;

}

LinkStack \*Init\_LinkStack(void) // 初始化函数，建立空链栈

{

LinkStack \*top;

top = (LinkStack \*)malloc(sizeof(LinkStack));

top->next = NULL;

return top;

}

int IsEmpty\_SqStack(SqStack \*s) // 判空函数

{

if (s->top == -1)

return 1;

else

return 0;

}

int Push\_SqStack(SqStack \*s, int x) // 压栈函数 顺序栈

{

if (s->top == MaxSize - 1) // 栈满，不能存入

{

printf("\n栈满，无法存入！");

return 0;

}

else

{

s->top++;

s->date[s->top] = x;

return 1;

}

}

int Push\_LinkStack(LinkStack \*top, int x) // 压栈函数 链栈

{

LinkStack \*s;

if (!(s = (LinkStack \*)malloc(sizeof(LinkStack))))

{

printf("\n无法取得地址，出现错误！");

return 0;

}

else

{

s->date = x;

s->next = top->next;

top->next = s;

return 1;

}

}

SqStack \*Creat\_SqStack(SqStack \*s) // 建立初始序列

{

int x;

scanf("%d", &x);

while (x != -1)

{

if (Push\_SqStack(s, x))

scanf("%d", &x);

else

break;

} // 输入数据创建A链表

printf("\n顺序栈创建完成！\n");

return s;

}

LinkStack \*Creat\_LinkStack(LinkStack \*top)

{

int x;

scanf("%d", &x);

while (x != -1)

{

if (Push\_LinkStack(top, x))

scanf("%d", &x);

else

break;

}

printf("\n链表创建完成！\n");

return top;

}

int Pop\_SqStack(SqStack \*s) // 出栈函数 顺序栈

{

if (IsEmpty\_SqStack(s))

return 0; // 栈空，不能出栈

else

{

s->top--;

return 1;

}

}

int Pop\_LinkStack(LinkStack \*top) // 出栈函数 链栈

{

LinkStack \*p;

if (top->next == NULL)

return 0;

else

{

p = top->next;

top->next = p->next;

free(p);

return 1;

}

}

void Output\_SqStack(SqStack \*s) // 输出函数 顺序栈

{

int i;

for (i = 0; i < s->top + 1; i++)

printf("%d,", s->date[i]);

printf("\n");

}

void Output\_LinkStack(LinkStack \*top) // 输出函数 链栈

{

LinkStack \*m;

m = top->next;

while (m)

{

printf("%d ", m->date);

m = m->next;

}

printf("\n");

}

# 实验 2.2 队列的基本实验

### 二、实验内容

1、实现顺序循环队列和链队的定义、创建、入队和出队操作。

（1）初始化顺序循环队列，并创建顺序循环队列的初始序列；实现顺序循环队列的插入和删除操作，并输出操作后的序列；

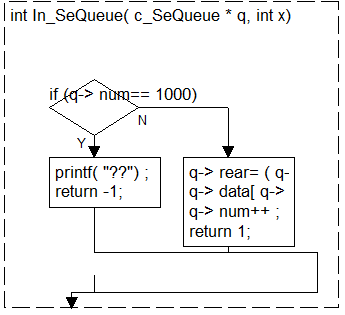
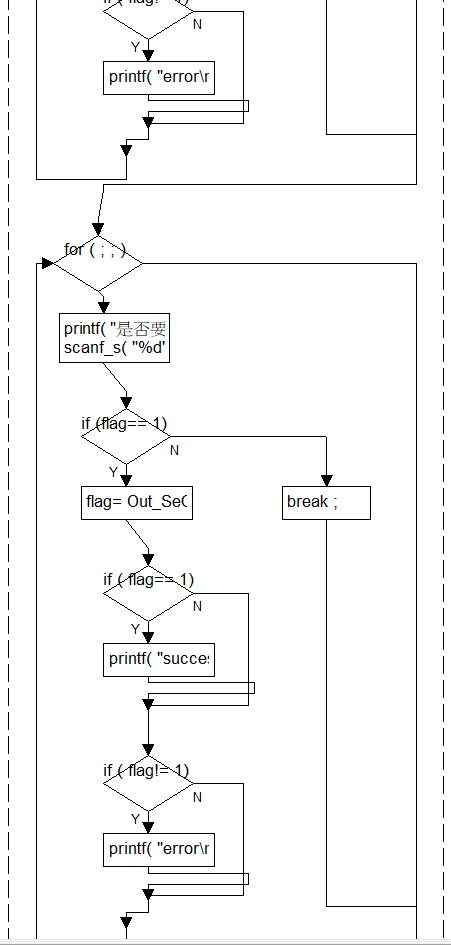
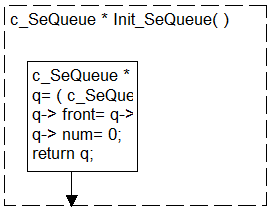
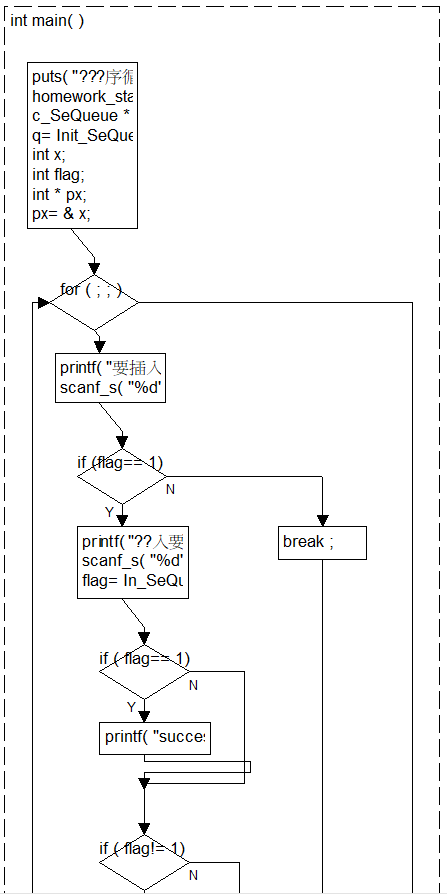
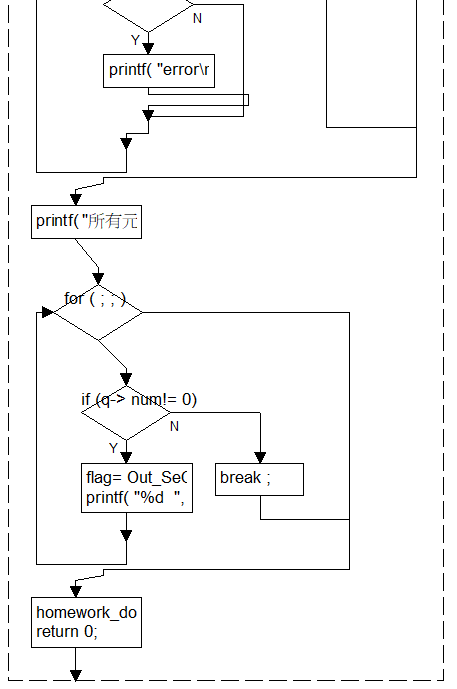
（2）初始化链队，并创建链队的初始序列；实现链队的插入和删除操作，并输出操作后的序列。

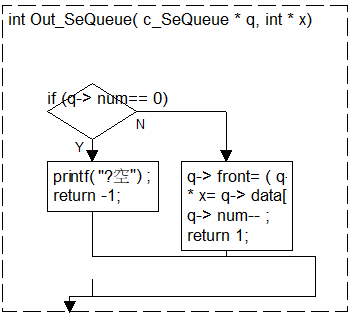
2、算法分析与程序设计

输入数据：建立输入处理，按队列里数据的逻辑顺序输入数据。 输出形式：分别输出队列的初始序列以及进行入队和出队操作以后的正确序列。

3. 算法描述

给出算法描述的程序流程图



4. 程序设计

给出实现程序功能的函数及说明，示例如下：

(1) 本程序包含的函数

主函数 main()

c\_SeQueue \*Init\_SeQueue() // 置空队

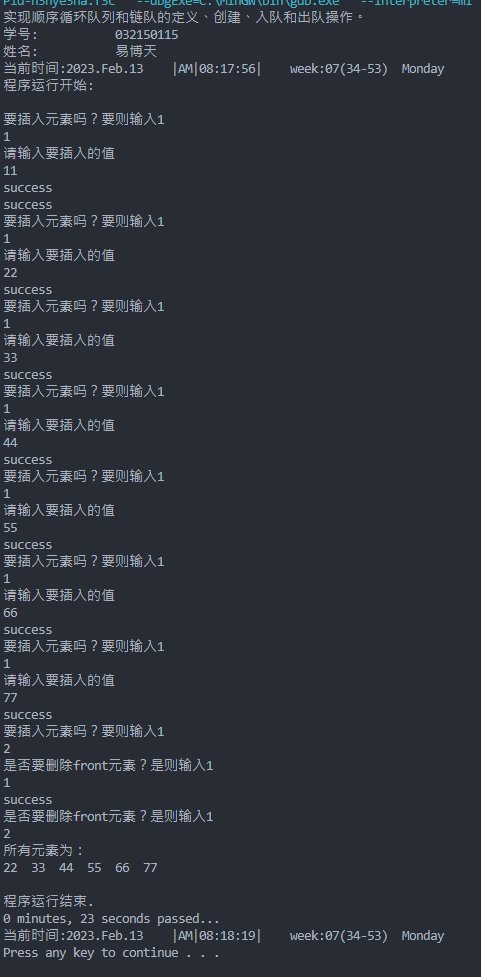
int In\_SeQueue(c\_SeQueue \*q, ElemType x) // 入队

int Out\_SeQueue(c\_SeQueue \*q, ElemType \*x) // 出队

### 四、调试分析

#### 1、调试过程及说明

* 以截屏图像的方式给出，包括程序运行开始和结束时的系统时间、姓名、学号、实验名称等信息.



#### 2、调试分析

如图

### 五、使用说明

按照运行时提示进行输入

**六、测试结果**

内容：给出测试用例测试程序。写出测试的情况，使用的输入以及得到的结果。

要求：测试的情况越全面越好。

如上图所示

**七、实验的收获和感想**

其实还可以把输出函数单独封装，这样就能完整实现要求了

**八、附录：源代码（双栏排版，加必要的注释）**

#include "homework.h"

#define MaxSize 1000 // 队列的最大容量

#define ElemType int

typedef struct c\_queue

{

ElemType data[MaxSize]; // 队列的存储空间

int front, rear; // 队列的队头队尾指针

int num; // 队列的队中元素个数

} c\_SeQueue;

c\_SeQueue \*Init\_SeQueue() // 置空队

{

c\_SeQueue \*q;

q = (c\_SeQueue \*)malloc(sizeof(c\_SeQueue));

q->front = q->rear = MaxSize - 1;

q->num = 0;

return q;

}

int In\_SeQueue(c\_SeQueue \*q, ElemType x) // 入队

{

if (q->num == MaxSize) // 队满不能入队

{

printf("队满");

return -1;

}

else

{

q->rear = (q->rear + 1) % MaxSize;

q->data[q->rear] = x;

q->num++;

return 1;

} // 入队完成

}

int Out\_SeQueue(c\_SeQueue \*q, ElemType \*x) // 出队

{

if (q->num == 0)

{

printf("队空");

return -1;

} // 队空不能出队

else

{

q->front = (q->front + 1) % MaxSize;

\*x = q->data[q->front]; // 读出队头元素

q->num--;

return 1;

} // 出队完成

}

int main()

{

puts("实现顺序循环队列和链队的定义、创建、入队和出队操作。");

homework\_start();

c\_SeQueue \*q;

q = Init\_SeQueue(); // 创建

ElemType x;

int flag;

ElemType \*px;

px = &x;

for (;;)

{

printf("要插入元素吗？要则输入1\n");

scanf\_s("%d", &flag, 1);

if (flag == 1)

{

printf("请输入要插入的值\n");

scanf\_s("%d", &x, 1);

flag = In\_SeQueue(q, x);

if (flag == 1)

printf("success\n");

if (flag != 1)

printf("error\n");

}

else

break;

}

for (;;)

{

printf("是否要删除front元素？是则输入1\n");

scanf\_s("%d", &flag, 1);

if (flag == 1)

{

flag = Out\_SeQueue(q, px);

if (flag == 1)

printf("success\n");

if (flag != 1)

printf("error\n");

}

else

break;

}

printf("所有元素为：\n");

for (;;)

{

if (q->num != 0)

{

flag = Out\_SeQueue(q, px);

printf("%d ", \*px);

}

else

break;

}

homework\_done();

return 0;

}