计算机与信息工程学院实验报告

••••••••••••••••••••••••••••••••• 密 ••••••••••••••••••••••••••••••••• 封 ••••••••••••••••••••••••••••••••• 线 •••••••••••••••••••••••••••••••••

姓名：\_\_\_于晨彤\_\_\_\_\_ 学号：\_2012080038\_\_\_\_\_\_\_ 专业：\_\_\_信息安全\_\_\_\_\_ 年级：\_\_2020\_\_\_\_\_\_

课程： 数据结构 主讲教师：\_\_刘成\_\_\_\_\_\_ 辅导教师：\_\_\_\_刘成\_\_\_

实验时间：\_2022\_\_年 \_4\_月 \_\_12\_日 \_下\_午\_5\_时至\_7\_时，实验地点\_\_\_计算机与信息学院608\_\_\_\_\_

实验题目： 顺序栈的基本操作 链队列的基本操作

实验目的： 通过该实验，让学生掌握栈的相关基本概念，认识栈是插入和删除集中在一端进行的线性结构，掌握栈的“先入后出”操作特点。栈在进行各类操作时，栈底指针固定不动，掌握栈空、栈满的判断条件。

通过该实验，使学生理解链队列的构造特点并灵活应用，掌握链队基本操作的编程实现，认识栈是在一端进行插入，在另一端进行删除集中操作的线性结构，掌握队列的“先入先出”操作特点，知道判断队列空和满的条件，进一步熟悉C语言中指针操作。

实验环境（硬件和软件） 普通PC机 CodeBlocks

实验内容：

用顺序存储结构，实现教材定义的栈的基本操作，提供数制转换功能，将输入的十进制整数转换成二进制。

用链式存储结构，实现教材定义的队列的基本操作。

实验步骤：

顺序栈：

在CodeBlocks中输入

#include<iostream>

#define MAXSIZE 100

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define OVERFLOW -2

using namespace std;

typedef int SElemType;

typedef int Status;

typedef struct{

SElemType \*base;

SElemType \*top;

int satcksize;

}SqStack;

//初始化栈

Status InitStack(SqStack &S){

S.base = new SElemType[MAXSIZE];

if(!S.base)

exit(OVERFLOW);

S.top = S.base;

S.satcksize=MAXSIZE;

return OK;

}

//入栈

Status Push(SqStack &S,SElemType e){

if(S.top-S.base==S.satcksize)

return ERROR;

\*S.top++=e;

return OK;

}

//出栈

Status Pop(SqStack &S,SElemType &e){

if (S.top==S.base){

return ERROR;

}

e=\*(--S.top);

return OK;

}

//取栈顶元素

SElemType GetTop(SqStack S){

if (S.top==S.base){

return ERROR;

}

SElemType e = \*(S.top-1);

return e;

}

//销毁栈

Status DestoryStack(SqStack &S){

if(S.base){

delete[] S.base;

S.satcksize = 0;

S.base = S.top = NULL;

}

return OK;

}

//清空栈

Status ClearStack(SqStack &S){

if(S.base){

S.top=S.base;

}

return OK;

}

//判断栈空

bool StackEmpty(SqStack S){

if(S.top==S.base)

return true;

else

return false;

}

//返回栈的长度

int StackLength(SqStack S){

return S.top-S.base;

}

//输出栈内元素

Status ShowStack(SqStack S){

if (S.base==S.top){

return ERROR;

}

while(S.top!=S.base){

S.top--;

cout<<\*S.top<<" ";

}

cout<<endl;

return OK;

}

//创建并输入栈元素

Status CreateStack(SqStack &S){

S.base = new SElemType[MAXSIZE];

if(!S.base)

exit(OVERFLOW);

S.top = S.base;

S.satcksize=MAXSIZE;

int n;

cout<<"请输入栈元素：";

while(cin>>n){

if(S.top-S.base==MAXSIZE){

cout<<"超过最大长度"<<endl;

return ERROR;

}

\*S.top = n;

S.top++;

if(cin.get()=='\n')

break;

}

return OK;

}

int main(){

SqStack S;

int n;

int num = 8;

int a[100];

InitStack(S);

int k=0;

while(num!=0){

Push(S,num%2);

num = num/2;

k++;

}

int e;

for(int i=0;i<k;i++){

Pop(S,e);

cout<<e;

}

cout<<endl;

while(1){

cout<<""<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1.初始化为空栈 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2.销毁栈 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3.将栈置空 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 4.判断栈是否为空 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 5.返回栈的长度 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 6.求栈顶元素 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 7.插入元素，并使其成为栈顶元素 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 8.删除元素并返回其值 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 9.输出栈内元素 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 10.创建并输入栈元素 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 11.退出 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"请输入选择";

cin>>n;

if(n==1){

if(InitStack(S)){

cout<<"初始化成功"<<endl;

}

else{

cout<<"初始化失败"<<endl;

}

}

else if(n==2){

if(DestoryStack(S)){

cout<<"成功销毁栈"<<endl;

}

else{

cout<<"销毁失败"<<endl;

}

}

else if (n==3){

if(ClearStack(S)){

cout<<"栈已清空"<<endl;

}

else{

cout<<"清空失败"<<endl;

}

}

else if(n==4){

if(StackEmpty(S)){

cout<<"栈为空"<<endl;

}

else{

cout<<"栈不为空"<<endl;

}

}

else if(n==5){

cout<<"栈的长度为："<<StackLength(S)<<endl;

}

else if(n==6){

SElemType e;

if(!GetTop(S)){

cout<<"栈为空，返回失败"<<endl;

}

else{

e = GetTop(S);

cout<<"栈顶元素为:"<<e<<endl;

}

}

else if(n==7){

SElemType e;

cout<<"请输入要插入的元素:";

cin>>e;

if(Push(S,e)){

cout<<"插入成功"<<endl;

}

else{

cout<<"栈满，插入失败"<<endl;

}

}

else if (n==8){

SElemType e;

if(Pop(S,e)){

cout<<"删除成功，栈顶元素为:"<<e<<endl;

}

else{

cout<<"删除失败"<<endl;

}

}

else if(n==9){

ShowStack(S);

}

else if(n==10){;

if(!CreateStack(S)){

cout<<"创建失败"<<endl;

}

}

else if(n==11){

break;

}

else{

cout<<"输入不合法"<<endl;

}

}

return 0;

}

链队列：

#include<iostream>

#define MAXSIZE 100

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define OVERFLOW -2

using namespace std;

typedef int QElemType;

typedef int Status;

typedef struct QNode{

QElemType data;

struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct{

QueuePtr fron;

QueuePtr rear;

}LinkQueue;

//构造空队列

Status InitQueue(LinkQueue &Q){

Q.fron = Q.rear = new QNode;

Q.fron->next = NULL;

return OK;

}

//2.销毁队列

Status DestoryQueue(LinkQueue &Q){

while(Q.fron){

Q.rear = Q.fron->next;

delete Q.fron;

Q.fron = Q.rear;

}

return OK;

}

//清空队列

Status ClearQueue(LinkQueue &Q){

Q.fron->next = Q.rear;

Q.rear->data=NULL;

return OK;

}

//判断链队列是否为空

bool QueueEmpty(LinkQueue Q){

return(Q.fron==Q.rear);

}

//求链队列的队头元素

Status GetHead(LinkQueue Q,QElemType &e){

if(Q.fron==Q.rear)

return ERROR;

e = Q.fron->next->data;

return OK;

}

//入队操作

Status EnQueue(LinkQueue &Q,QElemType e){

QueuePtr p = new QNode;

p->data = e;

p->next = NULL;

Q.rear->next = p;

Q.rear = p;

return OK;

}

//出队操作

Status DeQueue(LinkQueue &Q,QElemType &e){

if(Q.fron==Q.rear){

return ERROR;

}

QueuePtr p = Q.fron->next;

e = p->data;

Q.fron->next = p->next;

if(Q.rear==p){

Q.rear=Q.fron;

delete p;

return OK;

}

}

//创建队列

Status CreatQueue(LinkQueue &Q){

Q.fron =Q.rear=new QNode;

Q.fron->next = NULL;

QElemType e;

cout<<"请输入要的元素："<<endl;

while(cin>>e){

EnQueue(Q,e);

if(cin.get()=='\n'){

break;

}

}

return OK;

}

//返回队列中元素的个数

int NumQueue(LinkQueue Q){

int n = 0;

while(Q.fron->next){

n++;

Q.fron = Q.fron->next;

}

return n;

}

//返回队头元素

QElemType GetQueueFront(LinkQueue Q){

if(Q.fron==Q.rear){

return ERROR;

}

return Q.fron->next->data;

}

//显示队列中的元素

Status ShowQueue(LinkQueue Q){

if(QueueEmpty(Q)){

return ERROR;

}

while(Q.fron->next){

Q.fron = Q.fron->next;

cout<<Q.fron->data<<" ";

}

cout<<endl;

return OK;

}

int main(){

LinkQueue Q;

int n;

while(1){

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1.初始化队列 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2.销毁队列 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3.清空队列 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 4.判断队列是否为空 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 5.返回队列中元素的个数 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 6.返回队列中的队头元素 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 7.插入新的队尾元素 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 8.删除队头元素 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 9.初始化并创建队列 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 10.输出队列元素 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 11.退出 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"请输入选择";

cin>>n;

if(n==1){

if(InitQueue(Q)){

cout<<"初始化成功"<<endl;

}

}

else if(n==2){

if(DestoryQueue(Q)){

cout<<"成功销毁队列"<<endl;

}

}

else if(n==3){

if(ClearQueue(Q)){

cout<<"成功清空队列"<<endl;

}

}

else if(n==4){

if(QueueEmpty(Q)){

cout<<"队列为空"<<endl;

}

else{

cout<<"队列不为空"<<endl;

}

}

else if(n==5){

cout<<"队列中元素的个数为："<<NumQueue(Q)<<endl;

}

else if (n==6){

if(GetQueueFront(Q)){

cout<<"队头元素为："<<GetQueueFront(Q)<<endl;

}

}

else if (n==7){

QElemType e;

cout<<"请输入要入队的元素:";

cin>>e;

if(EnQueue(Q,e)){

cout<<"插入成功"<<endl;

}

}

else if(n==8){

QElemType e;

if(DeQueue(Q,e)){

cout<<"删除成功"<<endl;

}

else{

cout<<"删除失败"<<endl;

}

}

else if(n==9){

CreatQueue(Q);

}

else if(n==10){

ShowQueue(Q);

}

else if(n==11){

break;

}

}

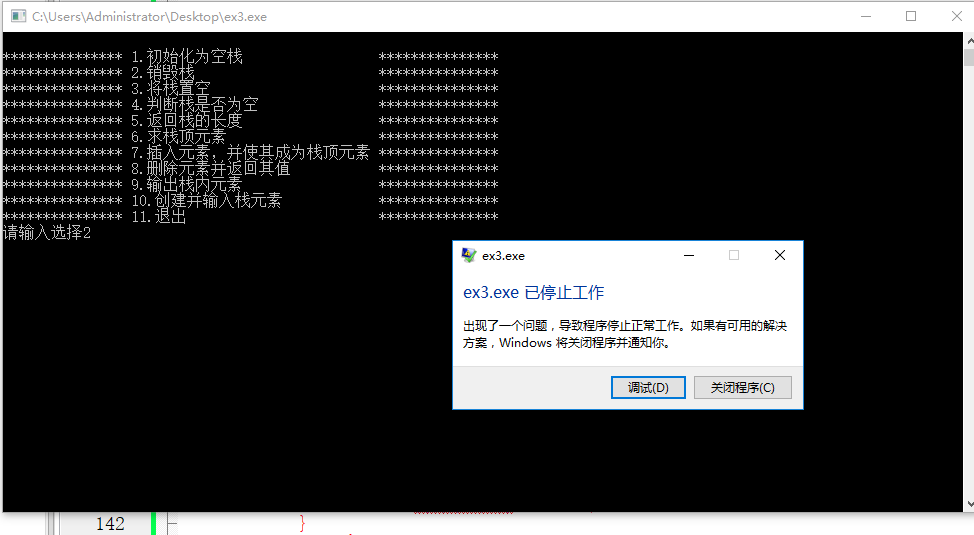
return 0;

}

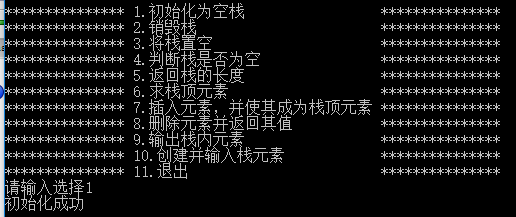
实验数据记录：

顺序栈：

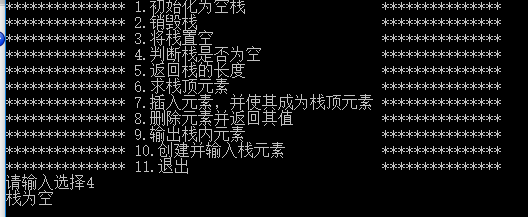
没有初始化前进行其他操作，程序是否能控制住；



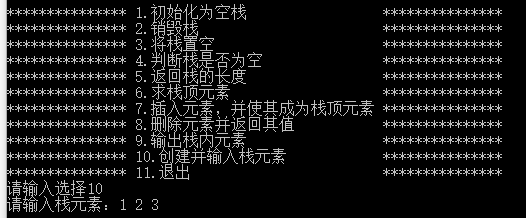
初始化一个栈；



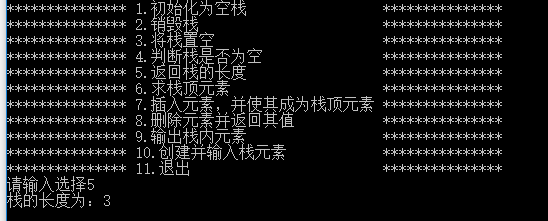
判栈空，屏幕显示栈为空；



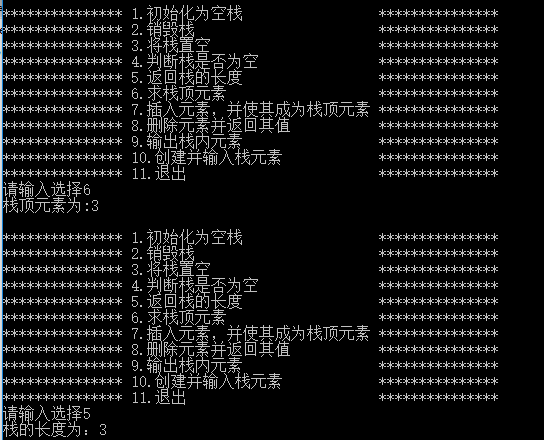
3个数入栈， 1、2、3；



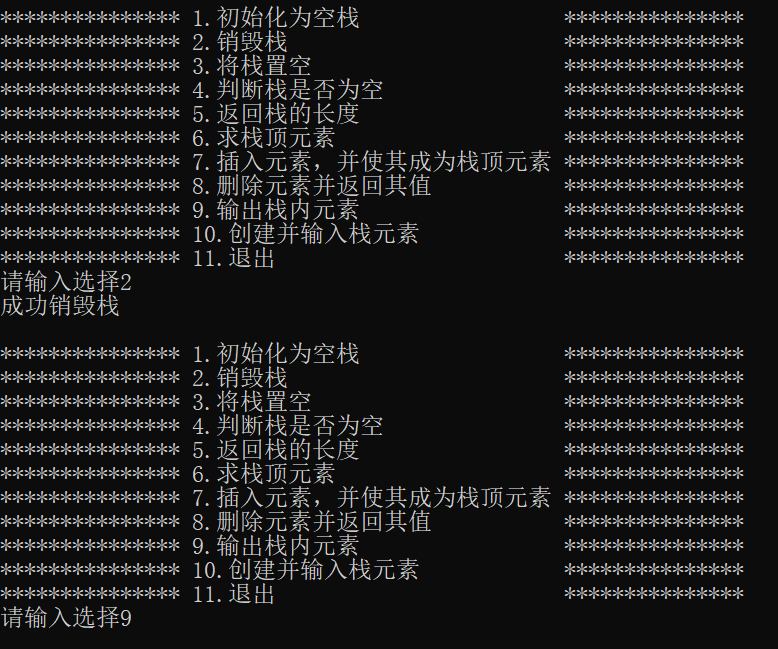
栈长度，屏幕输出3；



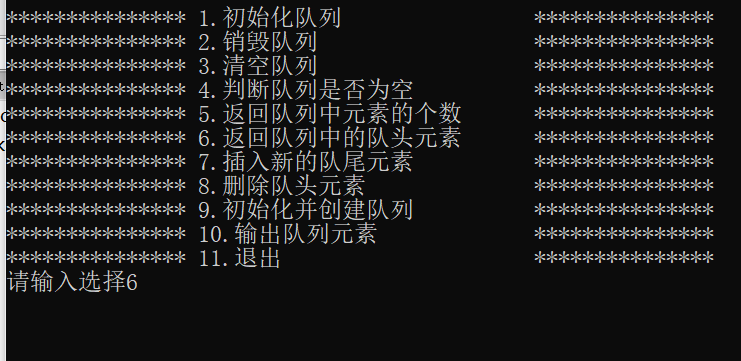
取栈顶元素，再判栈空，然后再判栈长度。让学生知道取栈顶元素不改变栈中的内容，栈顶指针不发生改变；



销毁栈，再做其他操作，判断程序是否能控制；

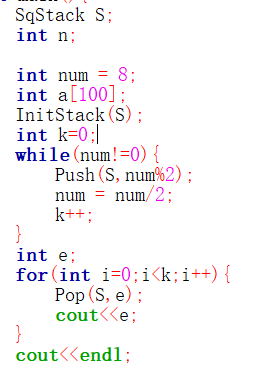


没有初始化前进行其他操作



向程序中加入

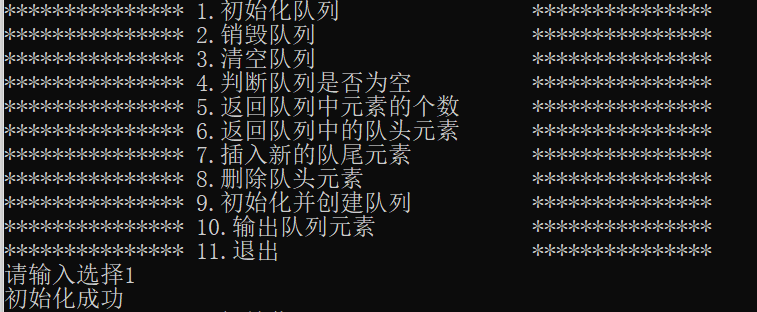
代码段



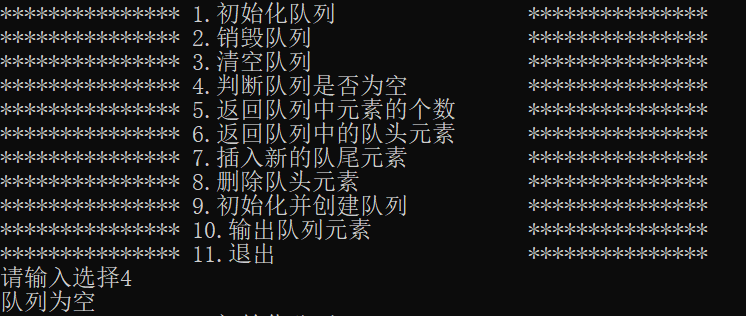
执行后



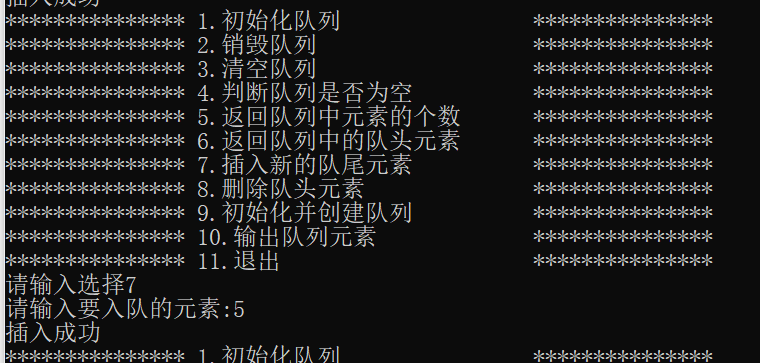
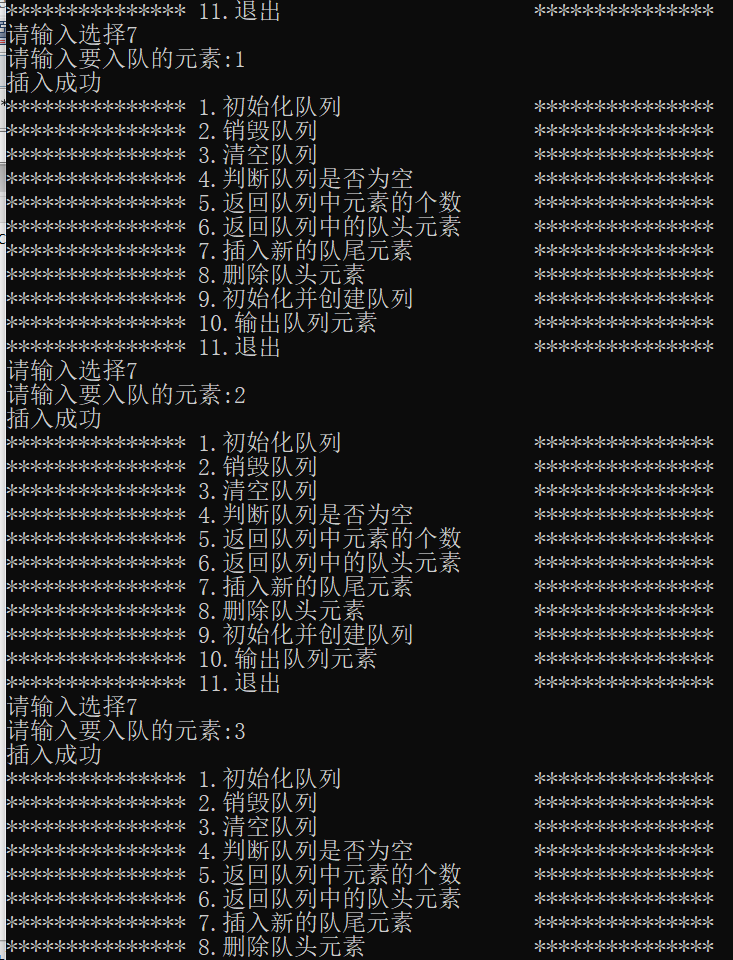
初始化一个队列



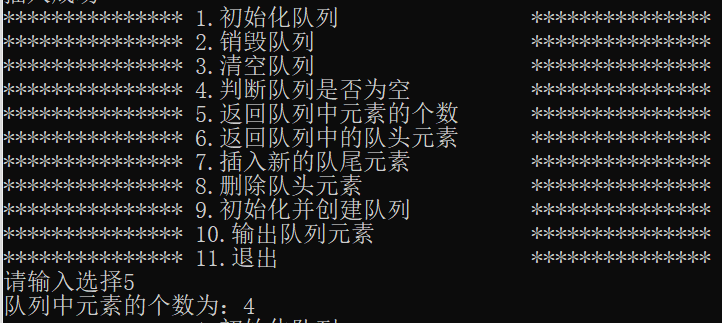
判队列空



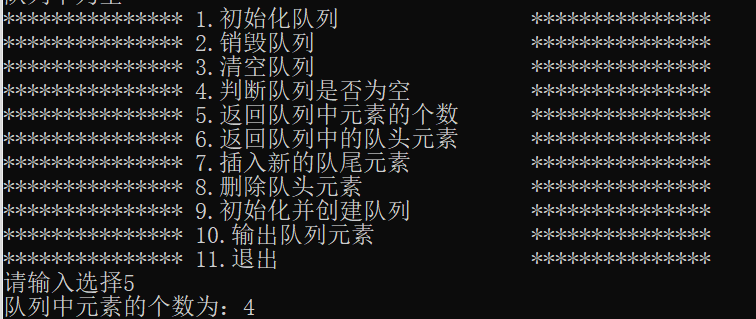
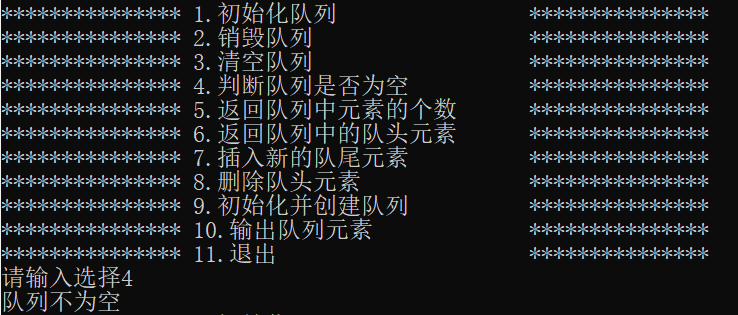
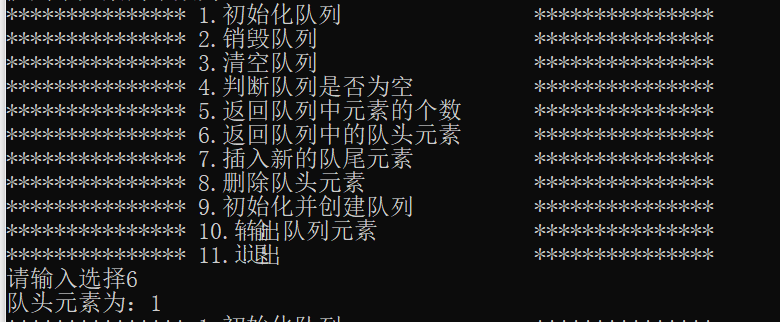
4个数入队1，2,3,5



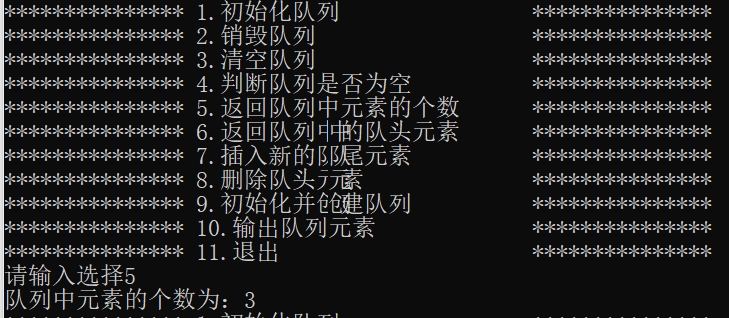
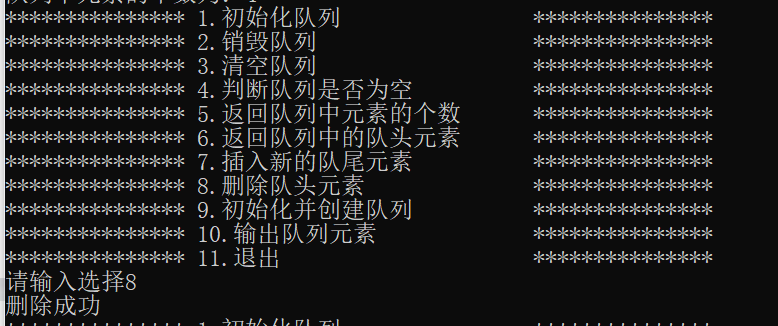
队列长度



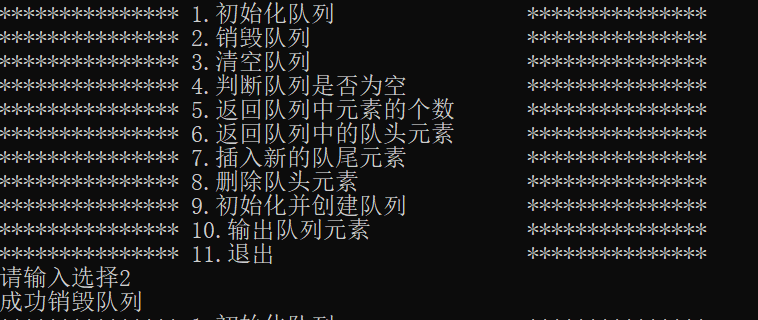
取队头元素，再判空，然后再判队的长度



出队，再判队的长度



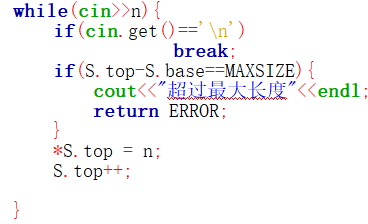
销毁队，再做其他操作，判断程序是否能控制



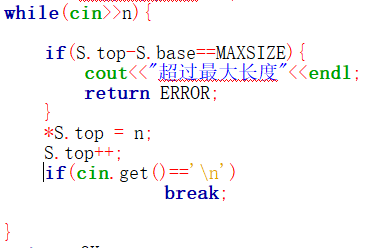
问题讨论：

问题：实验三在创建顺序栈时，想要实现输入的元素以空格分隔，输入回车结束的效果，循环代码：

结果创建爱你创建后的栈总是缺少最后一个元素



解决：将代码改为



初始化结果正确

问题：在做链队列初始化操作时，总是初始化失败

解决：原因是因为没有注意到front和head是指针，将front当做头结点，然后通过调用入队操作进行初始化