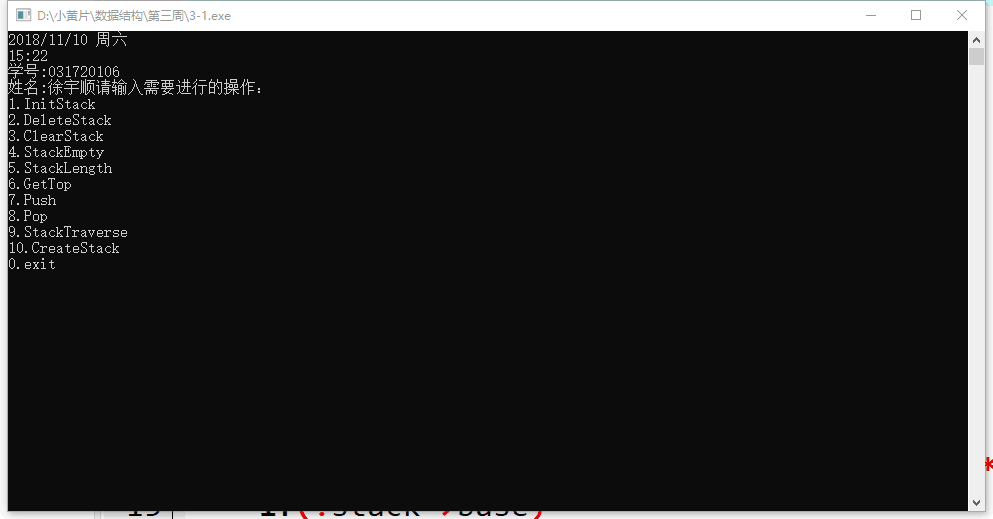
班号1617301 学号：031720106 姓名：徐宇顺

1、

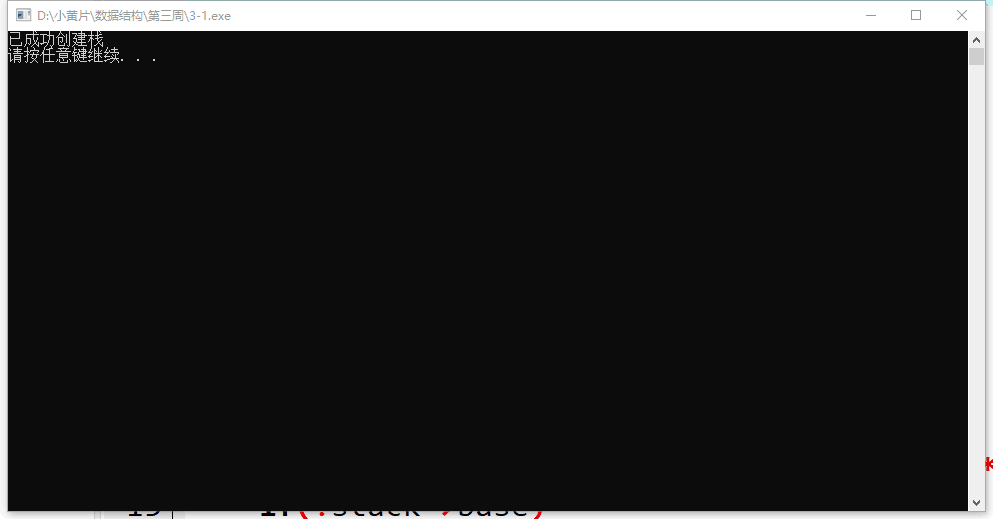
题目：编程实现书P45 ADT Stack 基本操作9个，用顺序存储结构实现；

算法思想：用顺序动态存储结构存储栈；

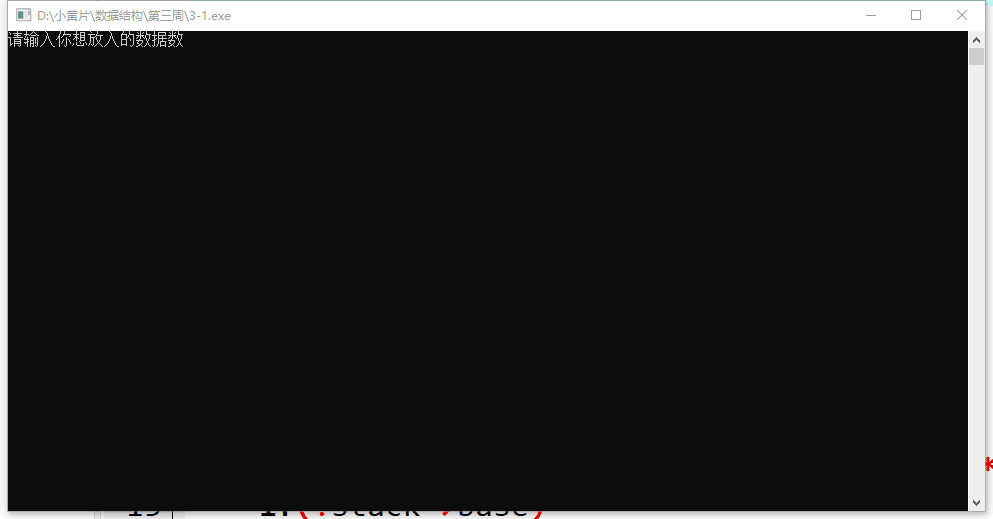
运行结果：



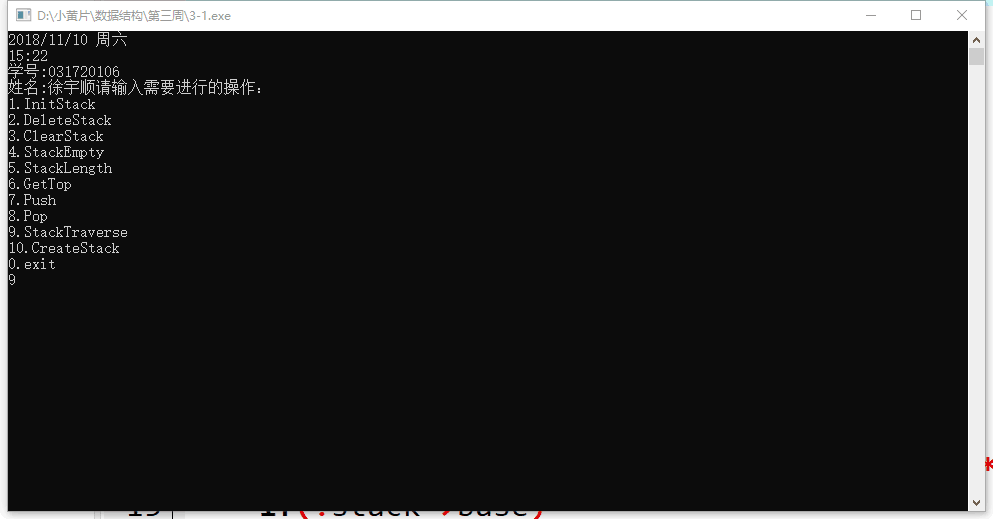
输入1，成功创建栈

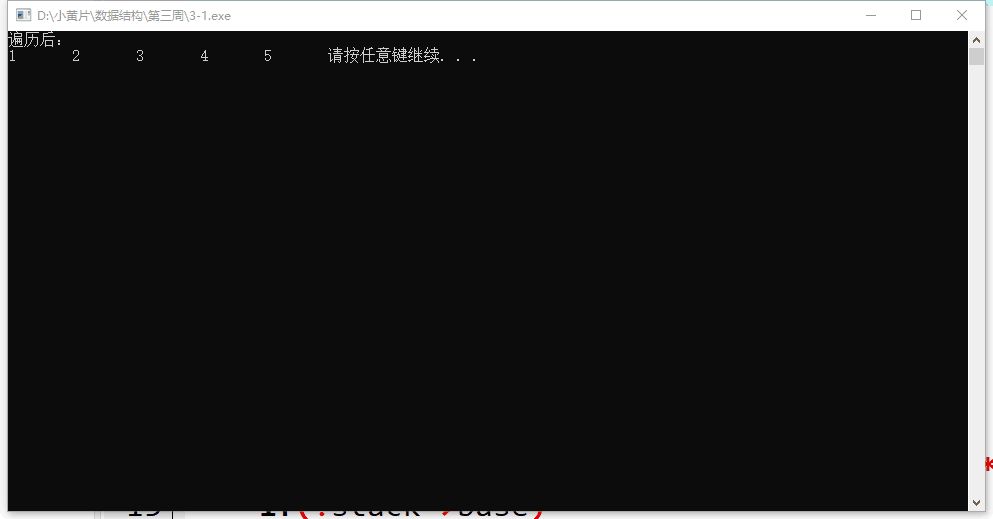


输入5，输入1,2,3,4,5

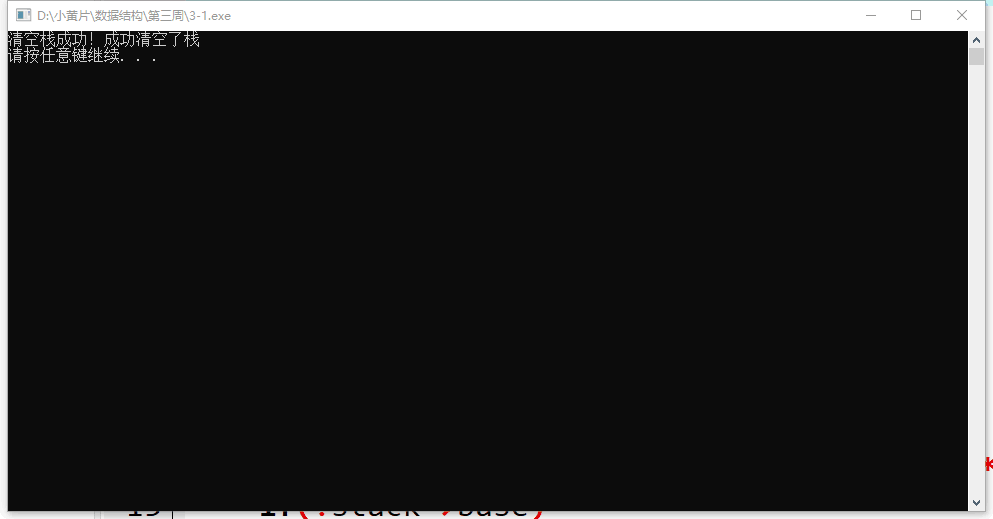


输入9，遍历整个栈

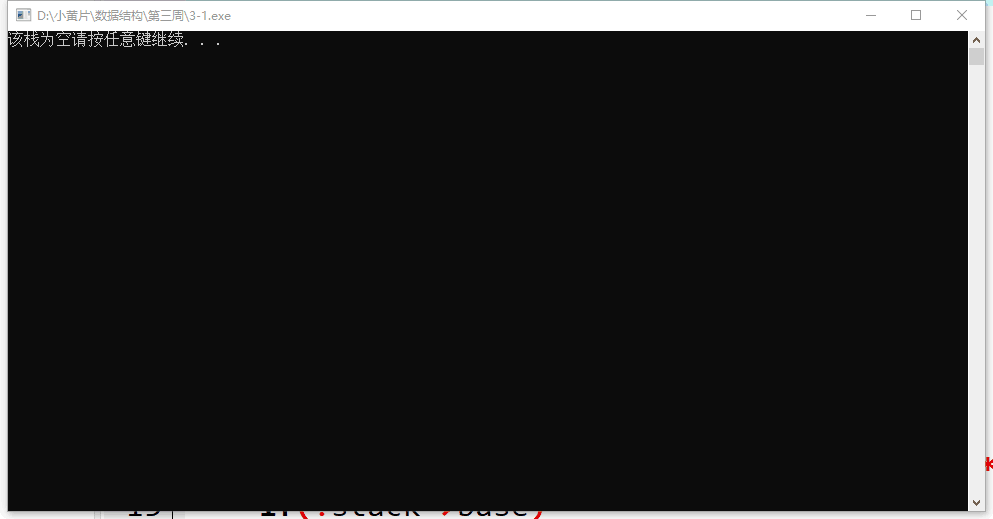




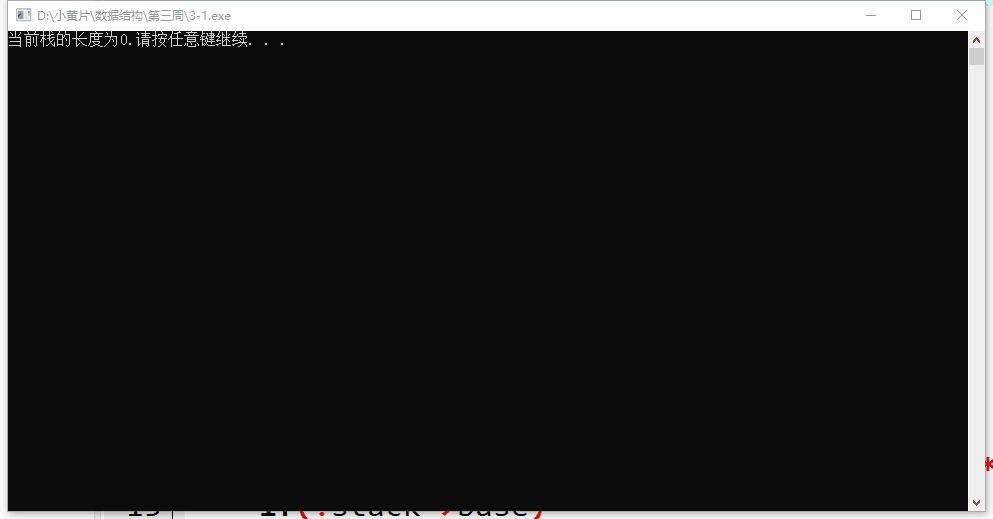
输入3，清空整个栈



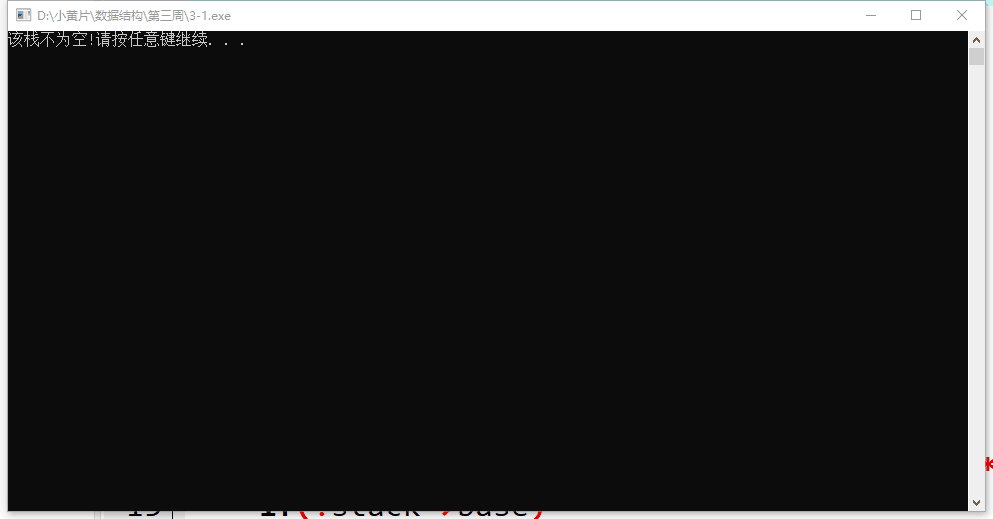
输入4，可知现在为空栈



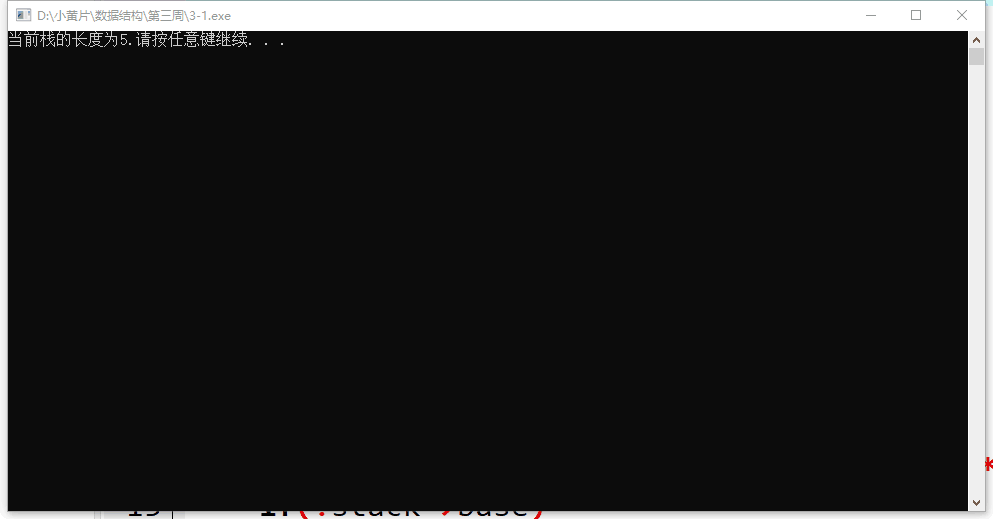
输入5，判断栈现在的长度为多少



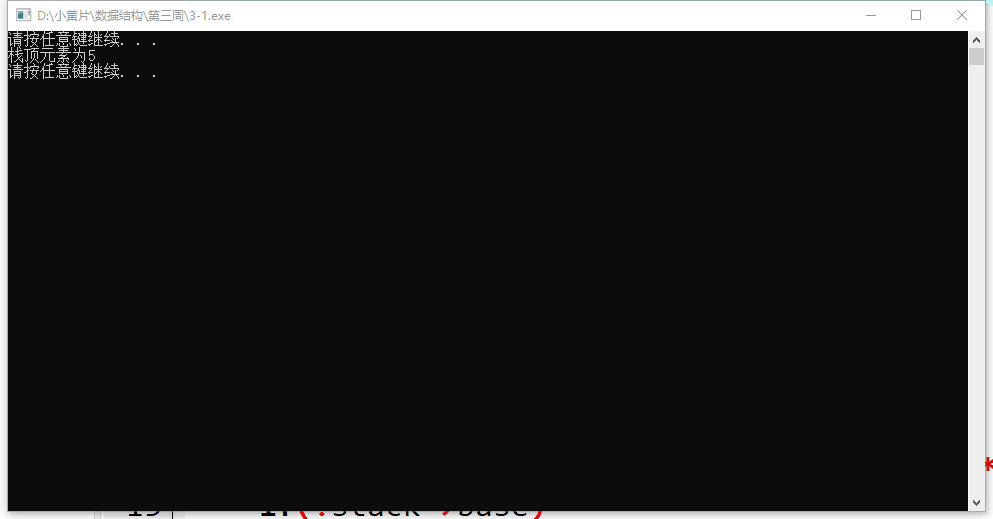
再次建立一次栈，此次栈不为空；



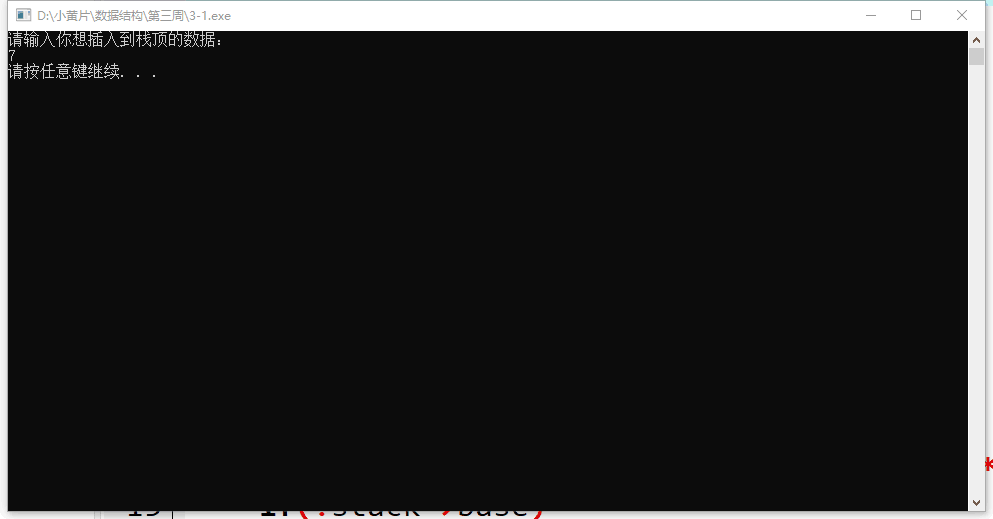
此时的栈的长度为5



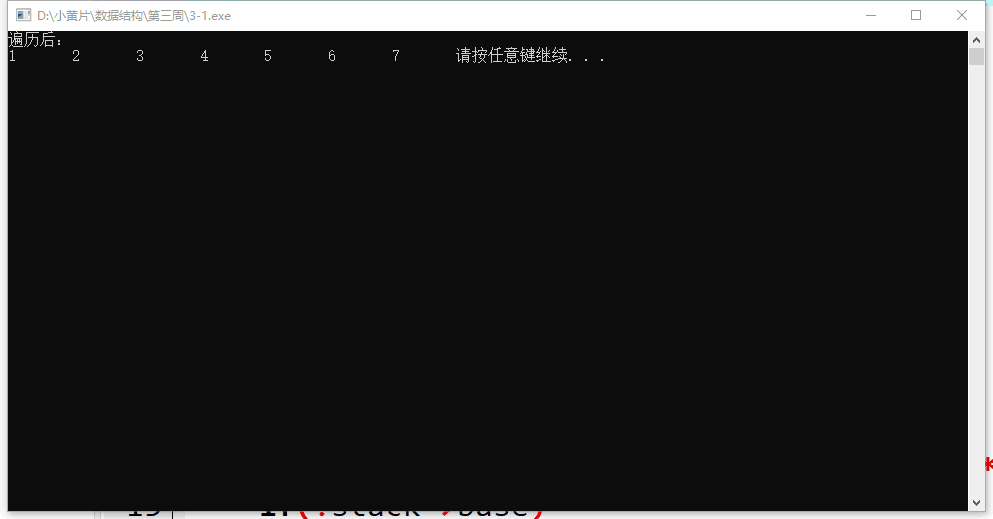
输入6，得到现在栈顶的元素是多少



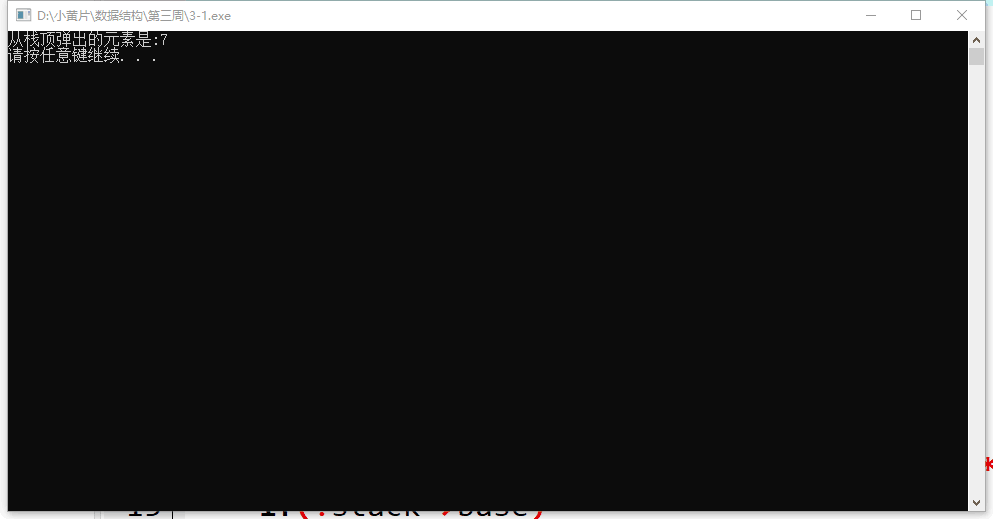
输入7，向栈顶放入6，7两个元素



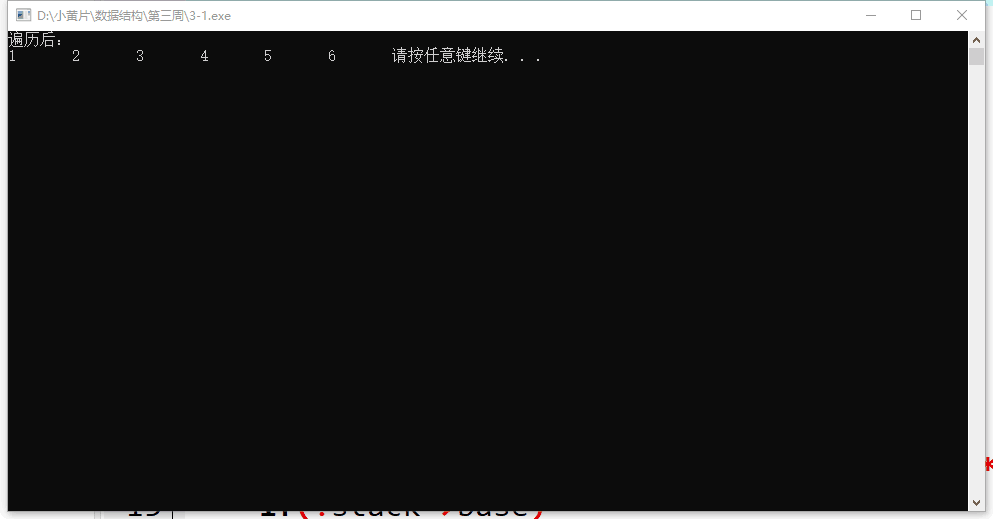
进行一次遍历



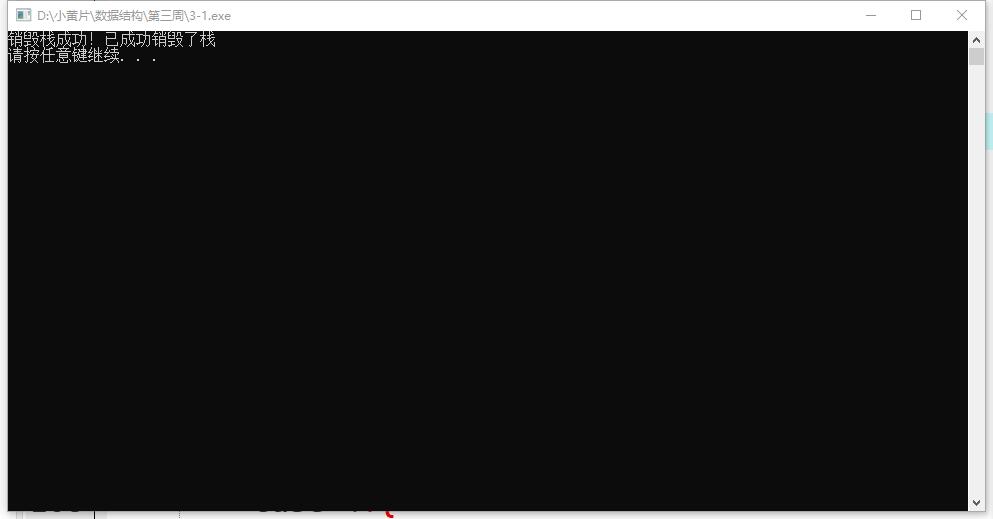
输入8，弹出栈顶的元素，即为7



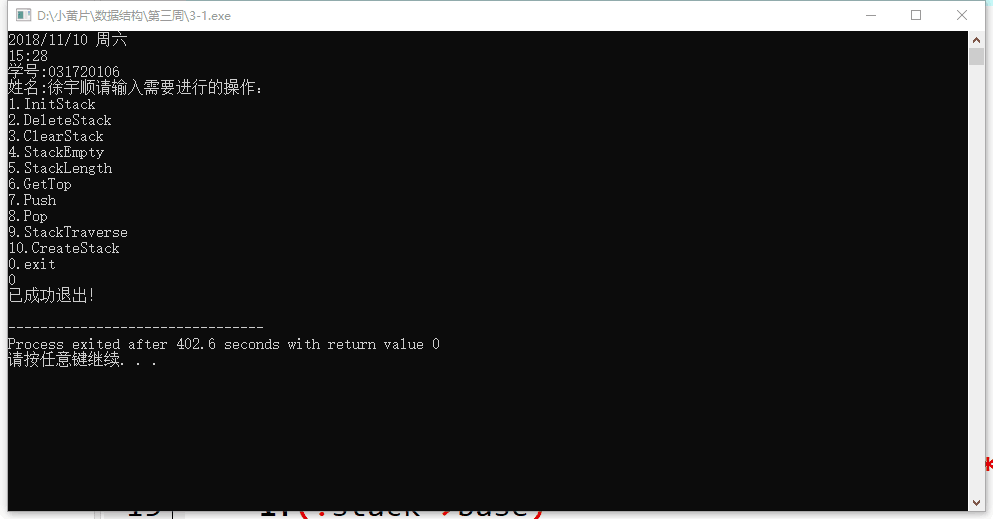
再次遍历；



输入2，销毁整个栈；



输入0，退出整个程序



运行结果：功能分别对应了数字

附源程序：#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#define STACK\_INIT\_SIZE 100

#define STACKINCREMENT 10

typedef struct sqstack{

int \*base;

int \*top;

int stacksize;

}SqStack;

void InitStack(SqStack \*stack)

{

stack->base=(int\*)malloc(STACK\_INIT\_SIZE\*sizeof(int));

if(!stack->base)

{

printf("创建栈失败！");

exit(0);

}

stack->top=stack->base;

stack->stacksize=STACK\_INIT\_SIZE;

}

void DestroyStack(SqStack \*stack)

{

free(stack->base);

stack->stacksize=0;

stack->base=NULL;

stack->top=NULL;

printf("销毁栈成功！");

}

void ClearStack(SqStack \*stack)

{

stack->top=stack->base;

printf("清空栈成功！");

}

void StackEmpty(SqStack stack)

{

if(stack.top-stack.base!=0)

{

printf("该栈不为空!");

}

else printf("该栈为空");

}

int StackLength(SqStack stack)

{

return (stack.top-stack.base);

}

int GetTop(SqStack \*stack,int \*e)

{

if(stack->top-stack->base==0){

printf("该栈为空!");

return 0;

}

\*e=\*(stack->top-1);

return 1;

}

void Push(SqStack \*stack,int e)

{

if(stack->top-stack->base==STACK\_INIT\_SIZE)

{

stack->base=(int \*)realloc(stack->base,(stack->stacksize+STACKINCREMENT)\*sizeof(int));

stack->top=stack->base+stack->stacksize;

stack->stacksize+=STACKINCREMENT;

}

if(!stack->base){

printf("插入元素时栈满，重新分配空间出错！");

return;

}

\*(stack->top)=e;

stack->top++;

}

void Pop(SqStack \*stack,int \*e)

{

if(stack->top-stack->base==0)

{

printf("该栈为空！");

return ;

}

\*e=\*(stack->top-1);

stack->top--;

}

void visit(int e)

{

printf("%d\t",e);

}

void StackTraverse(SqStack stack)

{

int \*p=stack.base;

if(stack.top-stack.base==0)

{

printf("该栈为空！");

return ;

}

while(p<stack.top)

{

visit(\*p);

p++;

}

}

void CreateStack(SqStack \*stack,int n)

{

int i;

stack->top=stack->base;

if(n>stack->stacksize)

{

stack->base=(int \*)realloc(stack->base,(stack->stacksize+STACKINCREMENT)\*sizeof(int));

stack->top=stack->base;

stack->stacksize+=STACKINCREMENT;

}

if(!stack->base){

printf("在向栈中给元素时，重新分配空间出错！");

return;

}

for(i=0;i<n;i++)

{

printf("请输入数:");

scanf("%d",stack->base+i);

stack->top++;

}

}

int main()

{

SqStack stack;

int m=-1,n;

int judge,length,ele,i;

while(m!=0){

system("cls");

system("date/t");

system("time/t");

printf("学号:031720106\n姓名:徐宇顺");

printf("请输入需要进行的操作：\n");

printf("1.InitStack\n");

printf("2.DeleteStack\n");

printf("3.ClearStack\n");

printf("4.StackEmpty\n");

printf("5.StackLength\n");

printf("6.GetTop\n");

printf("7.Push\n");

printf("8.Pop\n");

printf("9.StackTraverse\n");

printf("10.CreateStack\n");

printf("0.exit\n");

scanf("%d",&m);

switch(m)

{

case 1:{

system("cls");

InitStack(&stack);

printf("已成功创建栈\n");

system("pause");

break;

}

case 2:{

system("cls");

DestroyStack(&stack);

printf("已成功销毁了栈\n");

system("pause");

break;

}

case 3:{

system("cls");

ClearStack(&stack);

printf("成功清空了栈\n");

system("pause");

break;

}

case 4:{

system("cls");

StackEmpty(stack);

system("pause");

break;

}

case 5:{

system("cls");

length=StackLength(stack);

printf("当前栈的长度为%d.",length);

system("pause");

break;

}

case 6:{

system("cls");

judge=GetTop(&stack,&ele);

system("pause");

if(judge==0){

break;

}

printf("栈顶元素为%d\n",ele);

system("pause");

break;

}

case 7:{

system("cls");

printf("请输入你想插入到栈顶的数据：\n");

scanf("%d",&ele);

Push(&stack,ele);

system("pause");

break;

}

case 8:{

system("cls");

Pop(&stack,&ele);

printf("从栈顶弹出的元素是:%d\n",ele);

system("pause");

break;

}

case 9:{

system("cls");

printf("遍历后：\n");

StackTraverse(stack);

system("pause");

break;

}

case 10:{

system("cls");

printf("请输入你想放入的数据数\n");

scanf("%d",&n);

CreateStack(&stack,n);

system("pause");

break;

}

case 0:{

printf("已成功退出！\n");

exit(0);

break;

}

}

}

return 0;

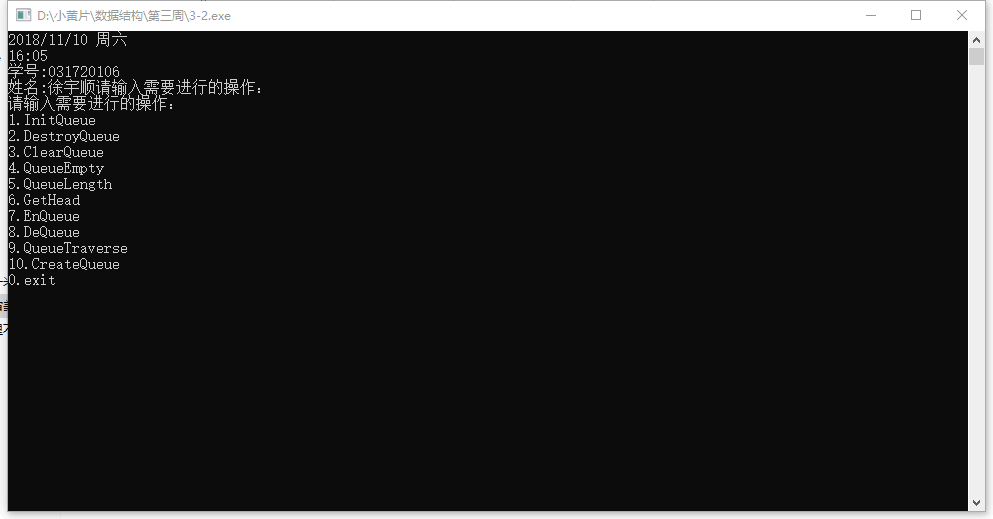
}

2、

题目：编程实现书P59 ADT Queue 基本操作9个，用链式存储结构实现；

算法思想：用链式存储结构实现队列

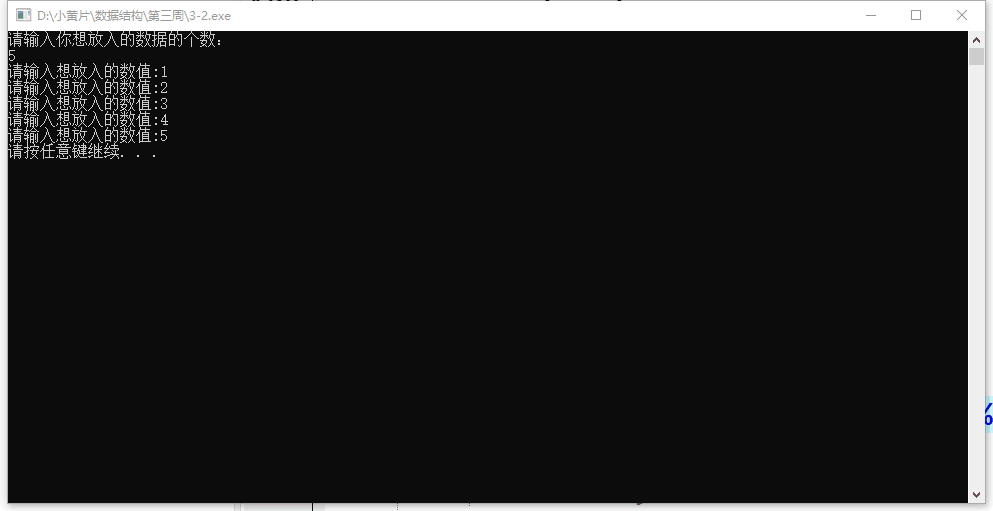
运行结果：



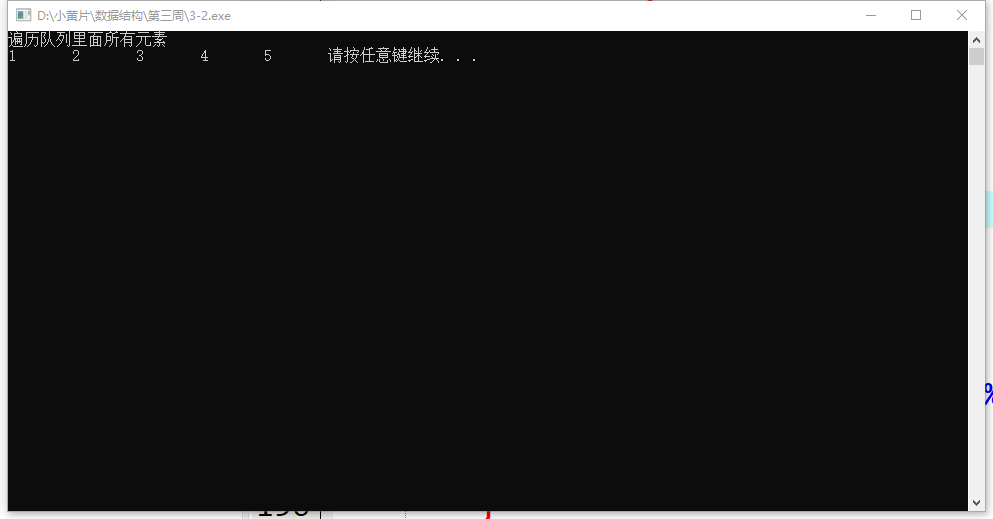
创建队列：



输入10创建队列，放入5个数据，1,2,3,4,5



输入9，遍历整个队列



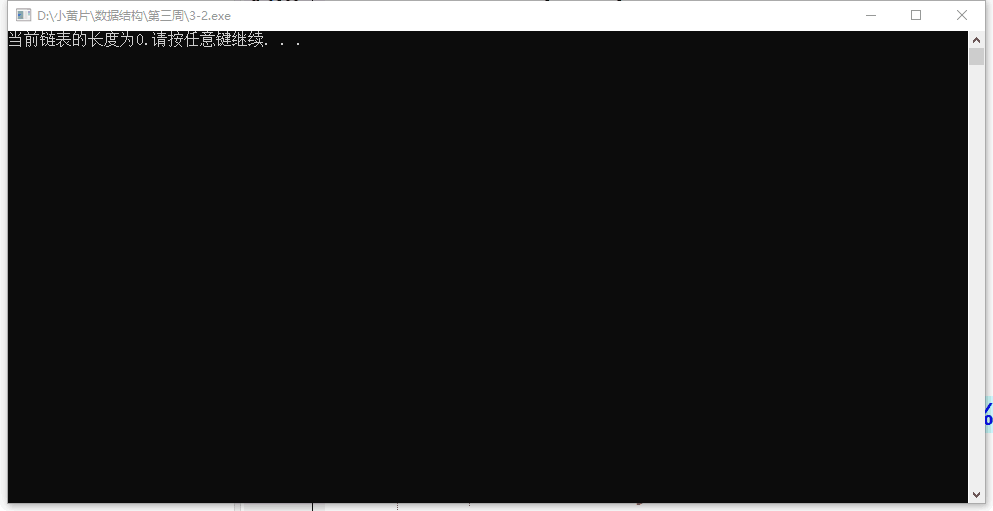
输入3，清空整个队列



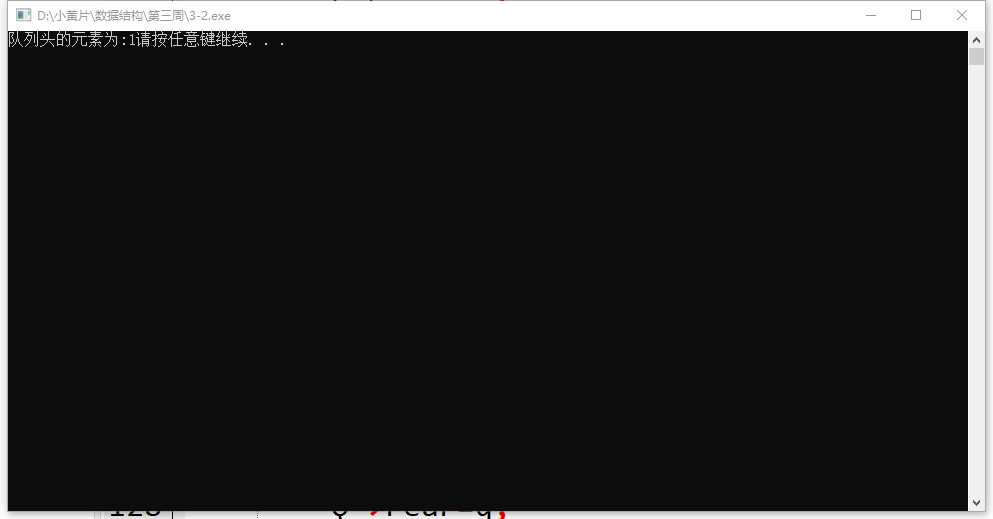
输入4，判断队列是否为空



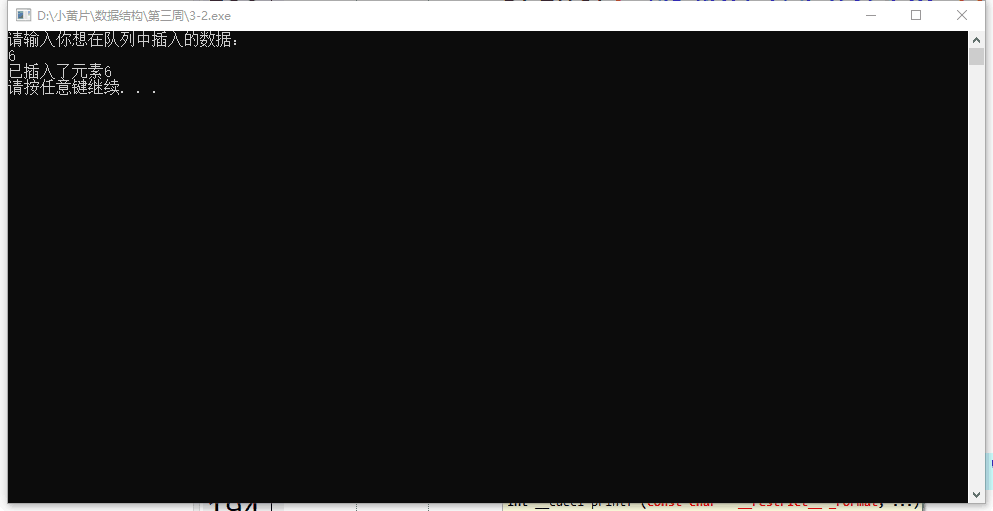
输入5，判断该队列的长度为多少



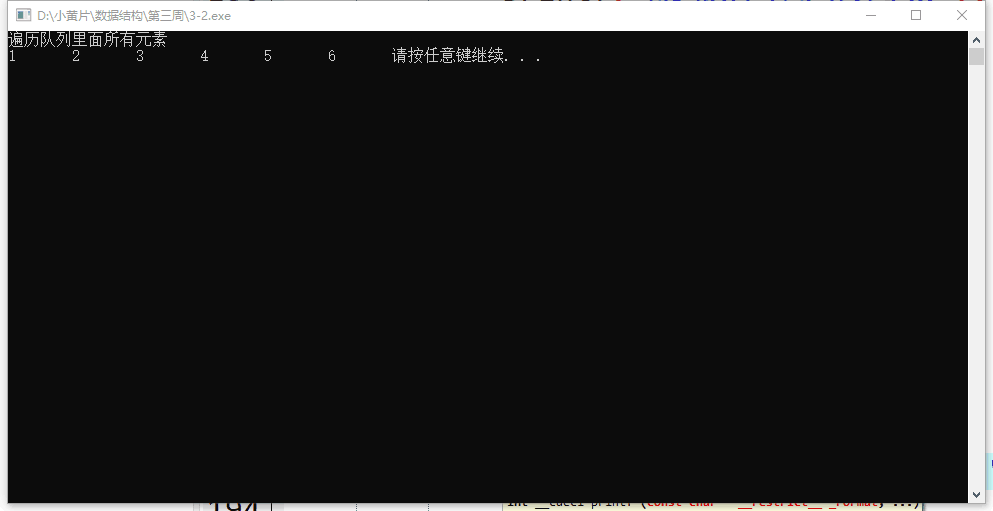
得到队列头的元素；



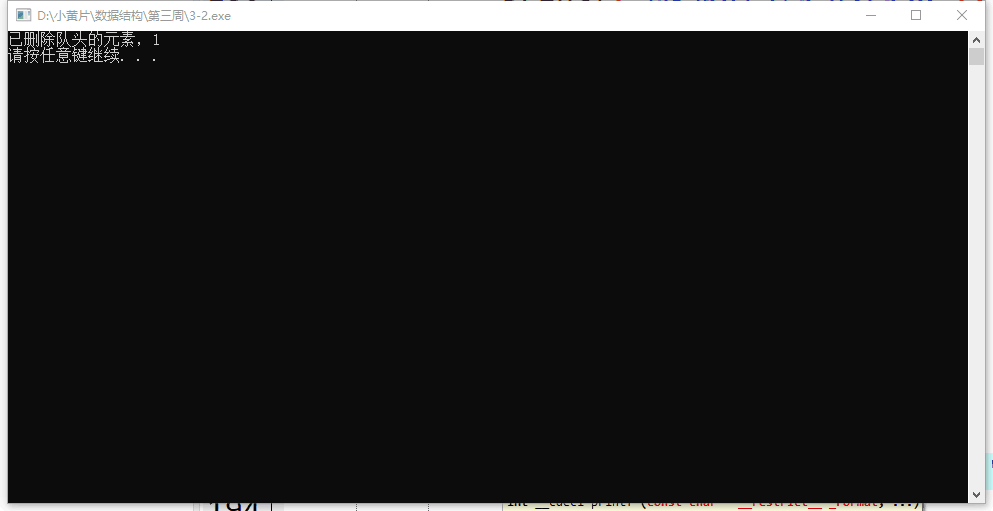
再次建立队列后，在队尾插入一个数据6；



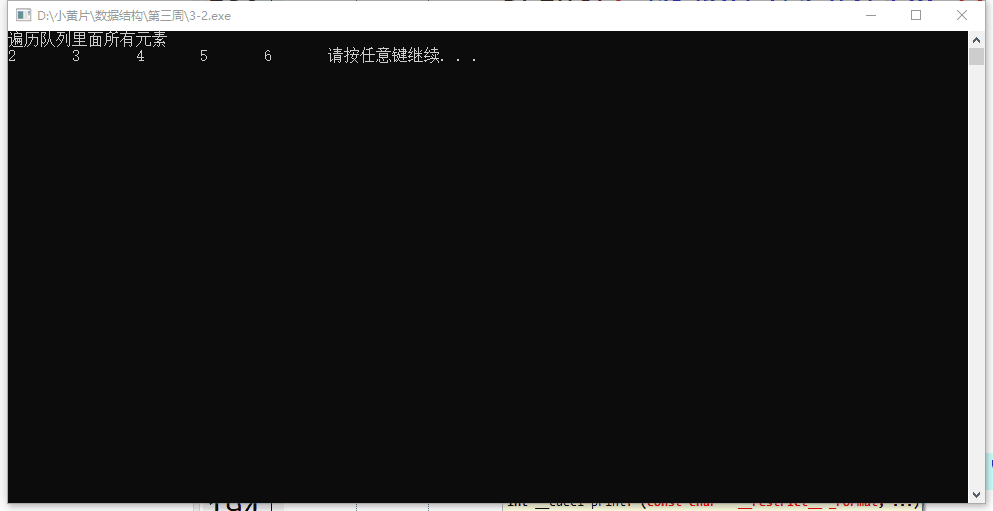
输入9，遍历整个队列



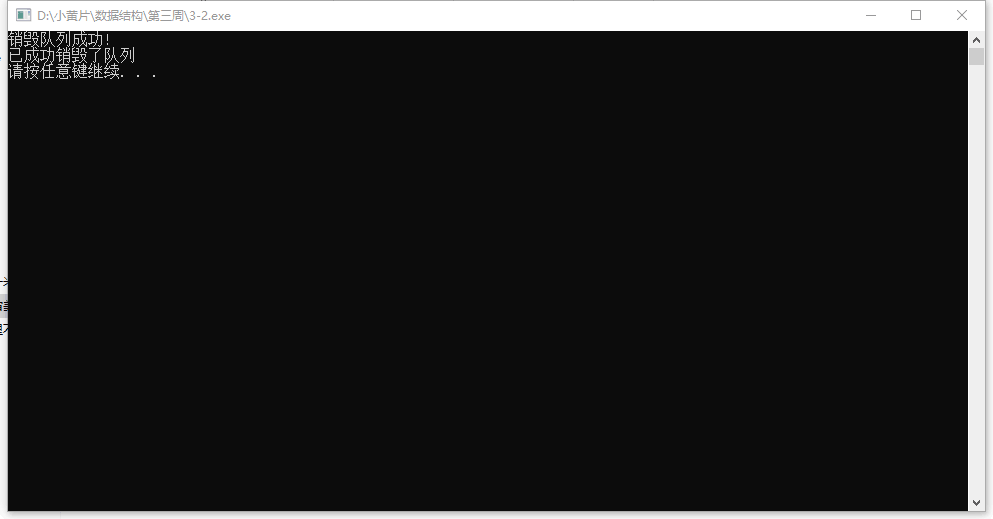
删除队头的元素1，



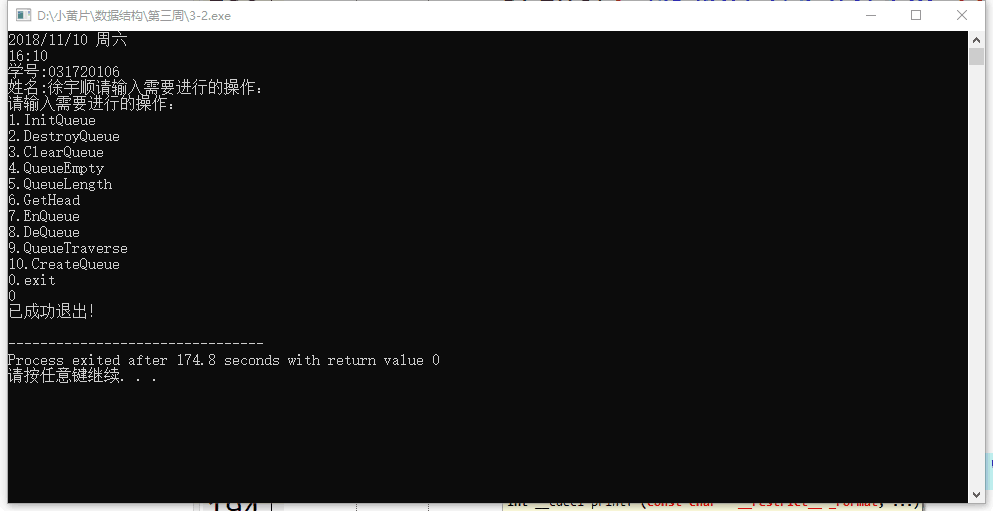
再次遍历：



销毁队列：



输入0后退出：



结果分析：每个数字对应一定的功能；

附源程序：

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

typedef struct QNode

{

int data;

struct QNode \*next;

}QNode,\*QueuePtr;

typedef struct Link

{

QueuePtr front;

QueuePtr rear;

}LinkQueue;

void InitQueue(LinkQueue \*Q)

{

Q->front=(QueuePtr)malloc(sizeof(QNode));

Q->rear=Q->front;

Q->front->next=NULL;

printf("创建队列成功!\n");

}

void DestroyQueue(LinkQueue \*Q)

{

QNode \*p,\*q;

p=Q->front;

q=p->next;

while(q)

{

free(p);

p=q;

q=p->next;

}

free(p);

printf("销毁队列成功!\n");

}

void ClearQueue(LinkQueue \*Q)

{

Q->rear=Q->front;

printf("清空列表成功！");

}

void QueueEmpty(LinkQueue Q)

{

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

printf("该队列为空！");

}

else printf("该队列不为空！");

}

int QueueLength(LinkQueue Q)

{

int length=0;

QNode \*p;

p=Q.front->next;

while(p!=NULL)

{

length++;

p=p->next;

}

return length;

}

void GetHead(LinkQueue \*Q,int \*e)

{

if(Q->front==Q->rear)

{

printf("该队列为空");

return;

}

QNode \*p;

p=Q->front->next;

\*e=p->data;

printf("队列头的元素为:%d",\*e);

}

void EnQueue(LinkQueue \*Q,int e)

{

QNode \*p;

p=(QNode\*)malloc(sizeof(QNode));

Q->rear->next=p;

p->data=e;

Q->rear=p;

p->next=NULL;

printf("已插入了元素%d\n",e);

}

void DeQueue(LinkQueue \*Q,int \*e)

{

if(Q->front==Q->rear)

{

printf("该队列为空");

return ;

}

QNode \*p,\*q;

p=Q->front->next;

\*e=p->data;

q=p->next;

Q->front->next=q;

if(Q->rear==p) Q->rear=Q->front;

free(p);

printf("已删除队头的元素，%d\n",\*e);

}

void visit(int e)

{

printf("%d\t",e);

}

void QueueTraverse(LinkQueue Q)

{

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

printf("该队列为空！");

return ;

}

QNode \*p;

p=Q.front->next;

printf("遍历队列里面所有元素\n");

while(p!=NULL)

{

visit(p->data);

p=p->next;

}

}

void CreateQueue(LinkQueue \*Q,int n)

{

int i;

QNode \*p,\*q;

p=Q->front;

for(i=0;i<n;i++)

{

q=(QNode\*)malloc(sizeof(QNode));

printf("请输入想放入的数值:");

scanf("%d",&q->data);

p->next=q;

Q->rear=q;

p=q;

}

Q->rear->next=NULL;

}

int main()

{

LinkQueue Q;

int m=-1,n;

int length;

bool judge;

int ele;

int i;

int pre\_e,next\_e;

while(m!=0){

system("cls");

system("date/t");

system("time/t");

printf("学号:031720106\n姓名:徐宇顺");

printf("请输入需要进行的操作：\n");

printf("请输入需要进行的操作：\n");

printf("1.InitQueue\n");

printf("2.DestroyQueue\n");

printf("3.ClearQueue\n");

printf("4.QueueEmpty\n");

printf("5.QueueLength\n");

printf("6.GetHead\n");

printf("7.EnQueue\n");

printf("8.DeQueue\n");

printf("9.QueueTraverse\n");

printf("10.CreateQueue\n");

printf("0.exit\n");

scanf("%d",&m);

switch(m)

{

case 1:{

system("cls");

InitQueue(&Q);

printf("已成功创建队列\n");

system("pause");

break;

}

case 2:{

system("cls");

DestroyQueue(&Q);

printf("已成功销毁了队列\n");

system("pause");

break;

}

case 3:{

system("cls");

ClearQueue(&Q);

printf("成功清空了队列\n");

system("pause");

break;

}

case 4:{

system("cls");

QueueEmpty(Q);

system("pause");

break;

}

case 5:{

system("cls");

length=QueueLength(Q);

printf("当前链表的长度为%d.",length);

system("pause");

break;

}

case 6:{

system("cls");

GetHead(&Q,&ele);

system("pause");

break;

}

case 7:{

system("cls");

printf("请输入你想在队列中插入的数据：\n");

scanf("%d",&ele);

EnQueue(&Q,ele);

system("pause");

break;

}

case 8:{

system("cls");

DeQueue(&Q,&ele);

system("pause");

break;

}

case 9:{

system("cls");

QueueTraverse(Q);

system("pause");

break;

}

case 10:{

system("cls");

printf("请输入你想放入的数据的个数：\n");

scanf("%d",&n);

CreateQueue(&Q,n);

system("pause");

break;

}

case 0:{

printf("已成功退出！\n");

exit(0);

break;

}

}

}

return 0;

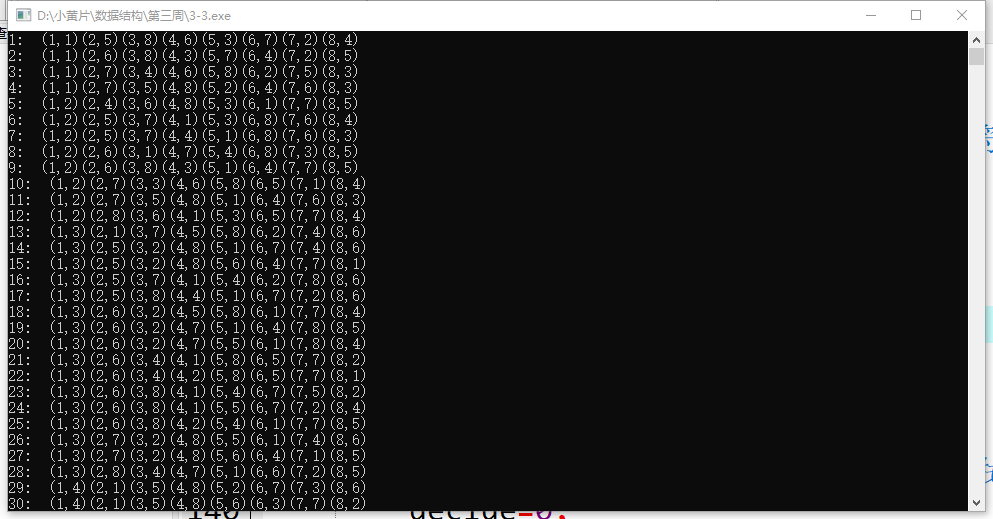
}

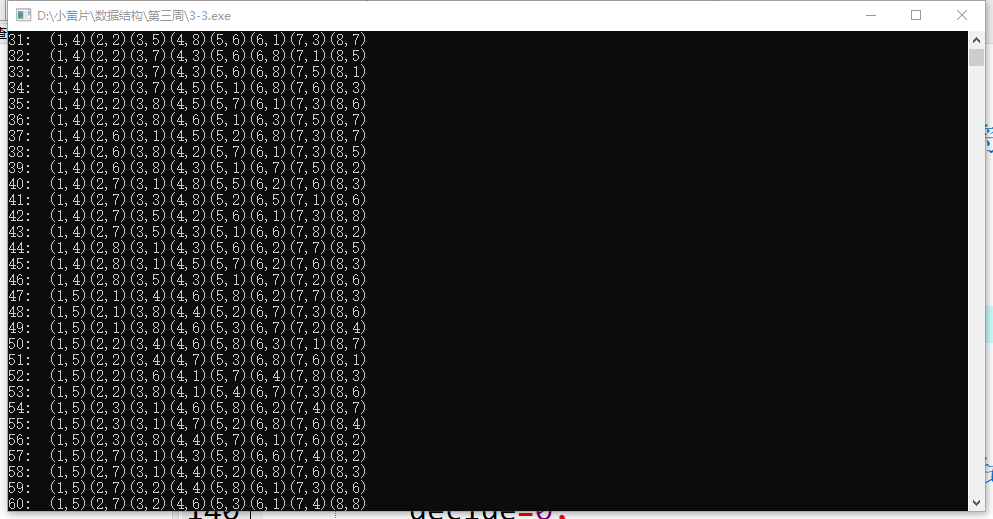
3、

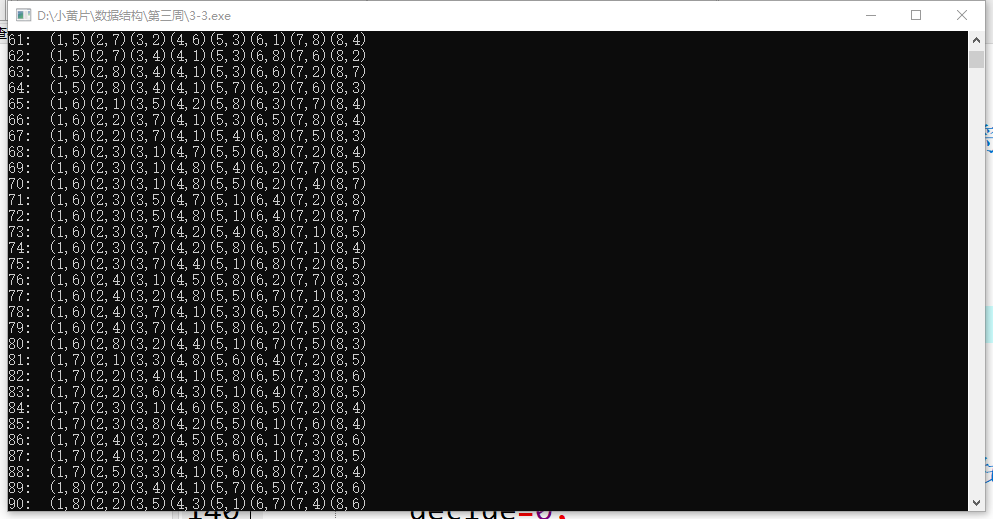
题目：利用栈操作实现八皇后问题求解。

算法思想：创建一个栈，放入坐标，再到下一行，下一行入栈的坐标和栈中坐标横坐标纵坐标不一样，并且横坐标纵坐标之差，横坐标纵坐标之和不等

运行结果：









结果分析：总共有92种，每一种有8个坐标

附源程序：

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#define STACK\_INIT\_SIZE 100

#define STACKINCREMENT 10

typedef struct site{

int x;

int y;

}Site;

typedef struct sqstack{

Site \*base;

Site \*top;

int stacksize;

}SqStack;

int CNT=0;

int count=0;

int startlist=0;//初始的列

void InitStack(SqStack \*stack)

{

stack->base=(Site\*)malloc(STACK\_INIT\_SIZE\*sizeof(Site));

if(!stack->base)

{

printf("创建栈失败！");

exit(0);

}

stack->top=stack->base;

stack->stacksize=STACK\_INIT\_SIZE;

}

void Pop(SqStack \*stack)

{

if(stack->top-stack->base==0)

{

printf("该栈为空！");

return ;

}

stack->top--;

}

void Push(SqStack \*stack,Site e)

{

if(stack->top-stack->base==STACK\_INIT\_SIZE)

{

stack->base=(Site \*)realloc(stack->base,(stack->stacksize+STACKINCREMENT)\*sizeof(Site));

stack->top=stack->base+stack->stacksize;

stack->stacksize+=STACKINCREMENT;

}

if(!stack->base){

printf("插入元素时栈满，重新分配空间出错！");

return;

}

\*(stack->top)=e;

stack->top++;

}

int Judge(SqStack \*stack,Site e)

{

Site\* p;

bool judge=1;

p=stack->base;

while(p<stack->top)

{

if((e.y==p->y)||(e.x==p->x)||(e.y-e.x==p->y-p->x)||(e.y+e.x==p->y+p->x)){

judge=0;

}

p++;

}

return judge;

}

void visit(Site e)

{

printf("(%d,%d)",e.x+1,e.y+1);

}

void StackTraverse(SqStack stack)

{

Site \*p=stack.base;

if(stack.top-stack.base==0)

{

printf("该栈为空！");

return ;

}

printf("%d: ",count);

while(p<stack.top)

{

visit(\*p);

p++;

}

}

int EmptyStack(SqStack \*stack)

{

if(stack->top-stack->base==0)

return 0;

else

return 1;

}//判断是否为空

void EightQueen(SqStack \*stack)

{

Site ele[10][10];

Site \*e;

int i,j,k=0;

int sign=1;

bool decide=0;

for(i=0;i<8;i++)//初始化整个矩阵

{

for(j=0;j<8;j++)

{

ele[i][j].x=i;

ele[i][j].y=j;

}

}

for(i=0;;){

if(CNT==8)

{

count++;

CNT--;

StackTraverse(\*stack);

k=(stack->top-1)->y+1;

Pop(stack);

i--;

printf("\n");

}//若有八个则输出一次

j=k;

for(;j<8;j++){

if(Judge(stack,ele[i][j]))

{

CNT++;

decide=1;

Push(stack,ele[i][j]);

i++;

k=0;

break;

}//判断该点是否适合条件，若符合条件，入栈，k=0即表示从下一行开始

}

if(decide==0){

sign=EmptyStack(stack);

if(!sign) break;

k=(stack->top-1)->y+1;

Pop(stack);

i--;

CNT--;

}//判断该行是否入栈，如果没有入栈，则弹出上一层，并从上一层的y值开始选择

decide=0;

}

}

int main()

{

SqStack stack;

InitStack(&stack);

EightQueen(&stack);

return 0;

}

4、

题目：问题描述

　　在某图形操作系统中,有 N 个窗口,每个窗口都是一个两边与坐标轴分别平行的矩形区域。窗口的边界上的点也属于该窗口。窗口之间有层次的区别,在多于一个窗口重叠的区域里,只会显示位于顶层的窗口里的内容。  
　　当你点击屏幕上一个点的时候,你就选择了处于被点击位置的最顶层窗口,并且这个窗口就会被移到所有窗口的最顶层,而剩余的窗口的层次顺序不变。如果你点击的位置不属于任何窗口,则系统会忽略你这次点击。  
　　现在我们希望你写一个程序模拟点击窗口的过程。

输入格式

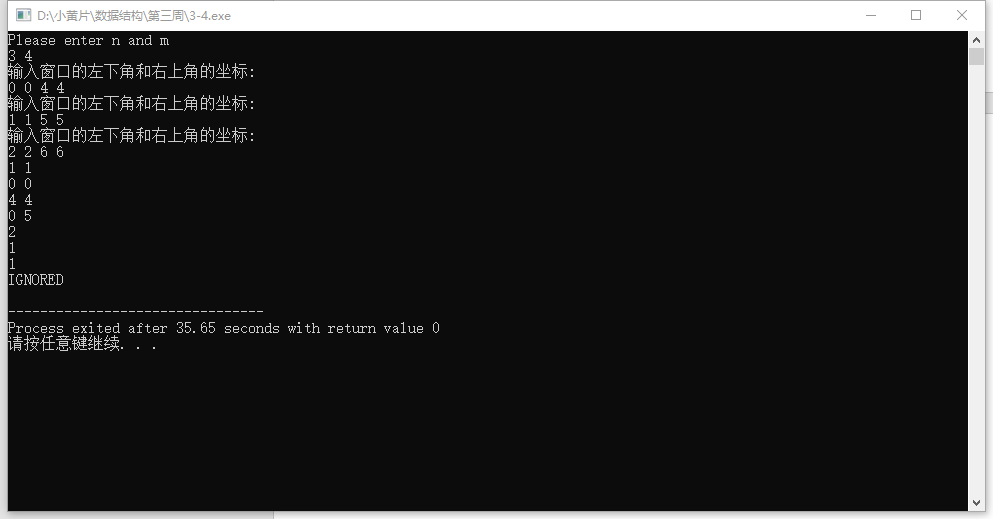
输入的第一行有两个正整数,即 N 和 M。(1 ≤ N ≤ 10,1 ≤ M ≤ 10)  
　　接下来 N 行按照从最下层到最顶层的顺序给出 N 个窗口的位置。 每行包含四个非负整数 x1, y1, x2, y2,表示该窗口的一对顶点坐标分别为 (x1, y1) 和 (x2, y2)。保证 x1 < x2, y1 < y2。  
　　接下来 M 行每行包含两个非负整数 x, y,表示一次鼠标点击的坐标。  
　　题目中涉及到的所有点和矩形的顶点的 x, y 坐标分别不超过 2559 和　　1439。

**问题分析：**这个问题可以用链式线性表来实现。

输出格式：输出包括 M 行,每一行表示一次鼠标点击的结果。如果该次鼠标点击选择了一个窗口,则输出这个窗口的编号(窗口按照输入中的顺序从 1 编号到 N);如果没有,则输出"IGNORED"(不含双引号)。

算法思想：用一个单链表存储，用头插法插入每个框的数据，判断点击的位置从头开始判断，判断到的那一个窗口，将其转到头指针之后。

运行结果：



结果分析：用一个单链表存储，用头插法插入每个框的数据，判断点击的位置从头开始判断，判断到的那一个窗口，将其转到头指针之后。

附源程序：

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

typedef struct point

{

int x;

int y;

}Point;

typedef struct lnode

{

int num;

Point low;

Point high;

struct lnode \*next;

}LNode,\*LinkList;

LinkList InitList()

{

LinkList L;

L=(LinkList)malloc(sizeof(LNode));

L->next=NULL;

return L;

}

void createList(LinkList list,int n)

{

LNode \*p,\*q;

p=list;

int i;

for(i=0;i<n;i++)

{

q=(LNode \*)malloc(sizeof(LNode));

q->num=i+1;

printf("输入窗口的左下角和右上角的坐标:\n");

scanf("%d%d%d%d",&q->low.x,&q->low.y,&q->high.x,&q->high.y);

q->next=p->next;

p->next=q;

}

}

void Click(Point points[],int m)

{

int i;

for(i=0;i<m;i++)

{

scanf("%d%d",&points[i].x,&points[i].y);

}

}

int Change(LinkList L,Point click)

{

LNode \*p,\*q;

p=L;

q=p->next;

if(q==NULL)

{

printf("该链表为空！");

return 0;

}

while(q)

{

if(click.x>=q->low.x&&click.x<=q->high.x&&click.y>=q->low.y&&click.y<=q->high.y)

{

p->next=q->next;

q->next=L->next;

L->next=q;

return q->num;

}

p=q;

q=p->next;

}

if(q==NULL){

return -1;

}

}

void Output(LinkList L,Point points[],int m)

{

int i;

int number;

for(i=0;i<m;i++)

{

number=Change(L,points[i]);

if(number==0){

continue;

}

else if(number==-1){

printf("IGNORED\n");

}

else printf("%d\n",number);

}

}

int main()

{

LinkList L;

int n,m;

Point points[0x100];

L=InitList();

printf("Please enter n and m\n");

scanf("%d%d",&n,&m);

createList(L,n);

Click(points,m);

Output(L,points,m);

return 0;

}