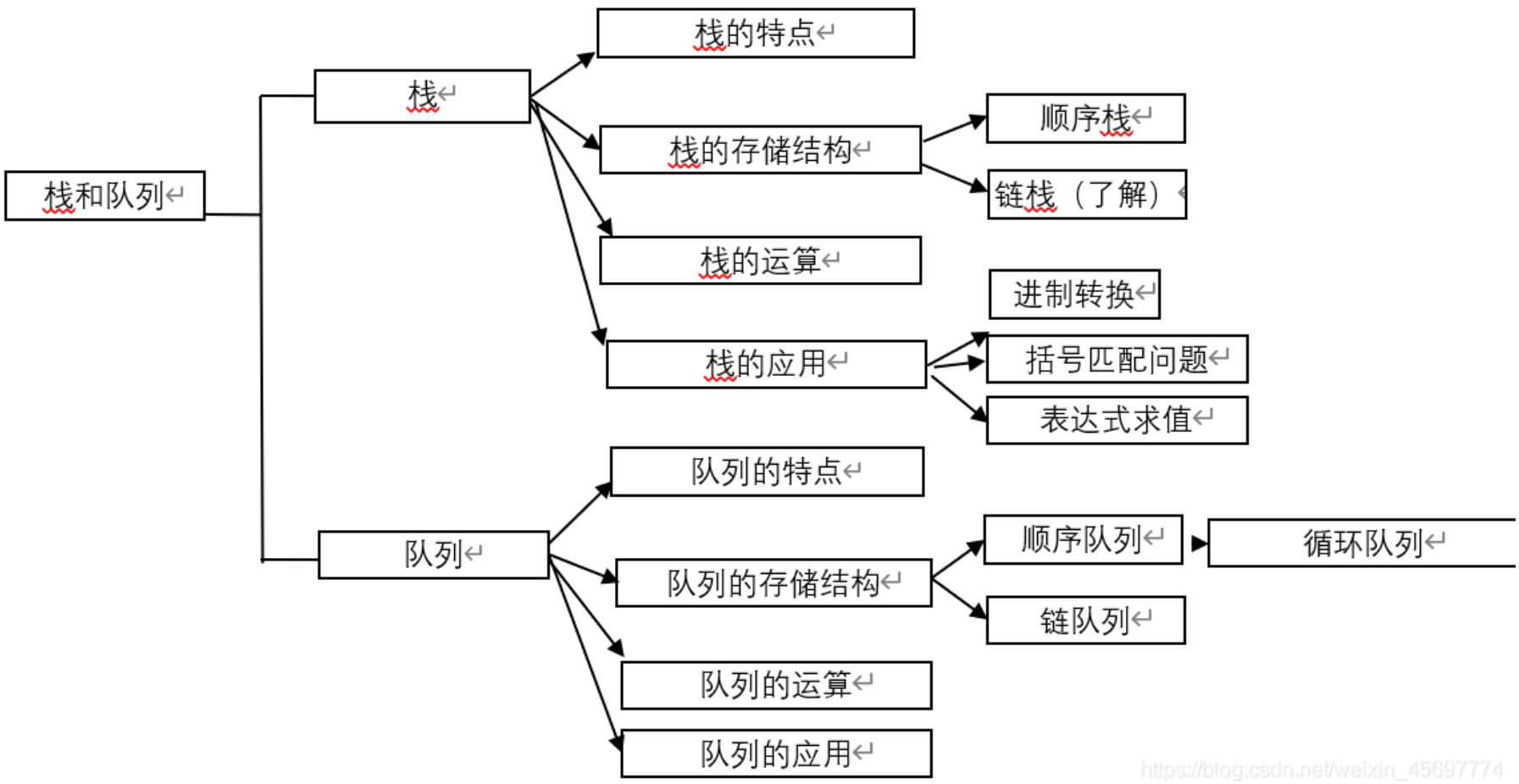
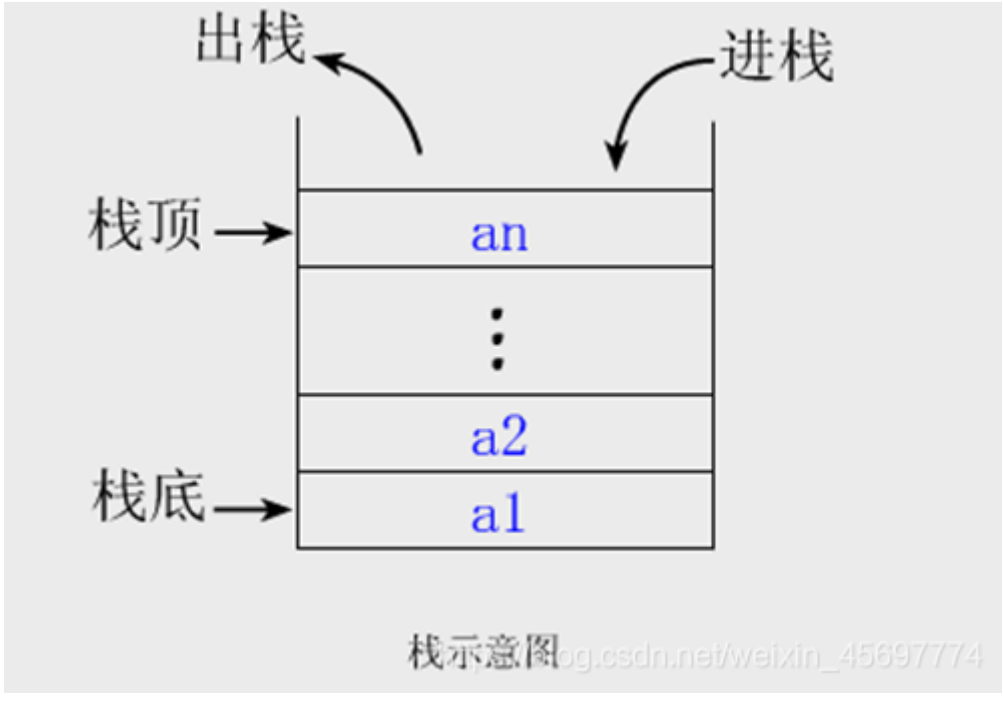
> **本系列博客为《数据结构》（C语言版）的学习笔记（上课笔记），仅用于学习交流和自我复习**

数据结构合集链接： 《数据结构》C语言版（严蔚敏版） 全书知识梳理（超详细清晰易懂）



一、栈

栈（$stack$）$(last\ in first\ out)$后进先出



栈示意图

0.栈的基本概念

- **定义**
  只能在表的一端（栈顶）进行插入和删除运算的线性表
- **逻辑结构**
  与线性表相同，仍为一对一关系
- **存储结构**
  用顺序栈或链栈存储均可，但以顺序栈更常见
- **运算规则**
  只能在栈顶运算，且访问结点时依照后进先出（LIFO）或先进后出（FILO）的原则
- **实现方式**
  关键是编写入栈和出栈函数，具体实现依顺序栈或链栈的不同而不同基本操作有入栈、出栈、读栈顶元素值、建栈、判断栈满、栈空等

1.栈的实现

可以用一个数组和一个变量（记录栈顶位置）来实现栈结构。

~~这里给出一套超级麻烦的方法，建议不看~~

2.栈与递归

- **优点**：结构清晰，程序易读
- **缺点**：每次调用要生成工作记录，保存状态信息，入栈；返回时要出栈，恢复状态信息。时间开销大。

设有一个递归算法如下:

```
int X(int n)
{ if(n≤3) return 1;
  else return X(n-2)+X(n-4)+1
}
```

则计算X(X(8))时需要计算X函数 多少次.
A. 8　　B.9　　C.16　　D.18

答案：D

3.Hanoi塔问题

标程：

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll N=1e5+7;
const ll mod=1e9+7;
ll cnt,n;
void move(ll id,char from,char to)
{
    printf ("step %lld: move %lld from %c→%c\n", ++cnt, id, from, to);
}
void hanoi(ll n,char x,char y,char z)
{
    if(n==0)
        return;
    hanoi(n-1,x,z,y);//把n-1个盘子全部从X经过z移动到y柱上
    move(n,x,z);//偷偷把第n个盘子从x移动到z上
    hanoi(n-1,y,x,z);//把n-1个盘子从y经过x移动到z柱上，结束
}

int main()
{
    cin>>n;
    hanoi(n,'A','B','C');
    return 0;
}
```

不懂的话建议看一下下面这篇我写的博客：
汉诺塔原理超详细讲解+变式例题

二、队列

队列是一种先进先出 $(FIFO)$ 的线性表. 在表一端插入,在另一端删除。

0.队列的基本概念

- **定义**
  只能在表的一端（队尾）进行插入，在另一端（队头）进行删除运算的线性表

- **逻辑结构**
  与线性表相同，仍为一对一关系
- **存储结构**
  用顺序队列或链队存储均可
- **运算规则**
  先进先出（FIFO）
- **实现方式**
  关键是编写入队和出队函数，具体实现依顺序队或链队的不同而不同

1.队列的实现

可用一个数组和两个变量优化为循环队列或者STL实现。比如循环队列 `queue` ， 双端队列 `deque`

2.循环队列

~~懒得敲了，偷个懒贴一些幻灯片吧~~

队列的抽象数据类型

ADT Queue {

数据对象: $D=\{a_i \mid a_i \in ElemSet, i=1,2,\cdots,n,\ n\geq 0\}$

数据关系: $R_1=\{<a_{i-1},a_i> \mid a_{i-1},a_i \in D, i=1,2,\cdots,n\}$

约定a_1端为队列头,a_n端为队列尾

基本操作:

(1) InitQueue (&Q) //构造空队列
(2) DestroyQueue (&Q) //销毁队列
(3) ClearQueue (&S) //清空队列
(4) QueueEmpty(S) //判空. 空--TRUE,

队列的抽象数据类型

(5) QueueLength(Q) //取队列长度
(6) GetHead (Q,&e) //取队头元素,
(7) EnQueue (&Q,e) //入队列
(8) DeQueue (&Q,&e) //出队列
(9) QueueTraverse(Q,visit()) //遍历

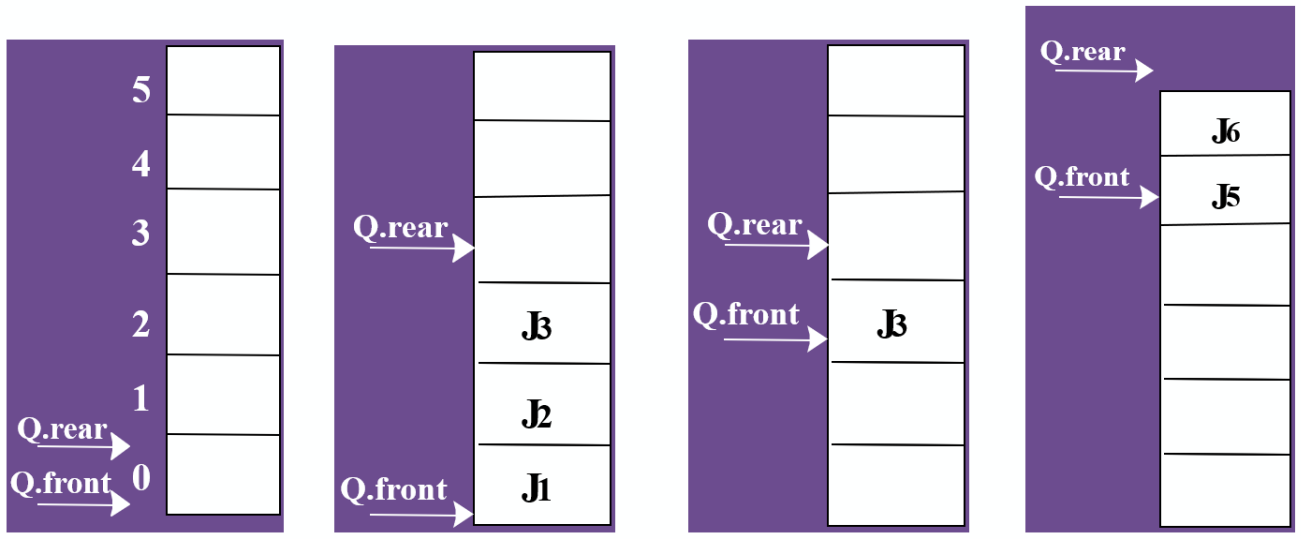
}ADT Queue

https://blog.csdn.net/weixin_45697774

队列的顺序表示 - - 用一维数组base[M]

```
#define M  100   //最大队列长度
Typedef struct {
   QElemType *base;  //初始化的动态分配存储空间
   int  front;          //头指针
   int  rear;           //尾指针
}SqQueue;
```

队列的顺序表示 - - 用一维数组base[M]



front=rear=0

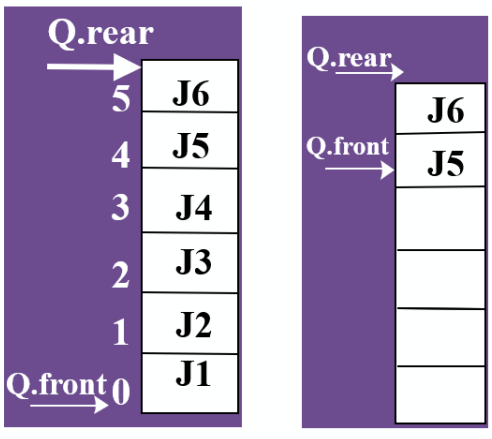
空队标志：front= =rear

入队：base[rear++]=x;

出队：x=base[front++];

存在的问题

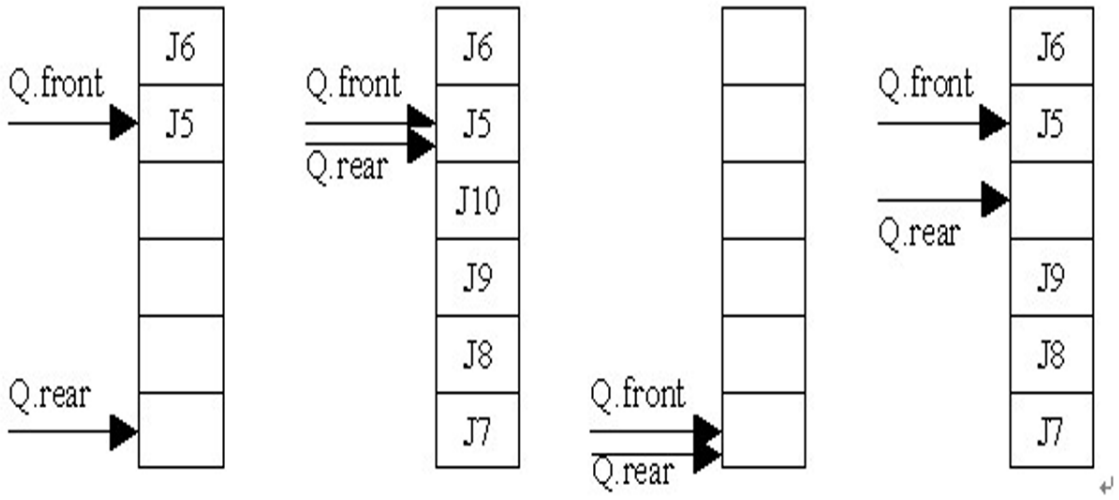
设大小为M



front=0
rear=M时
再入队—真溢出

front□ 0
rear=M时
再入队—假溢出

存在的问题

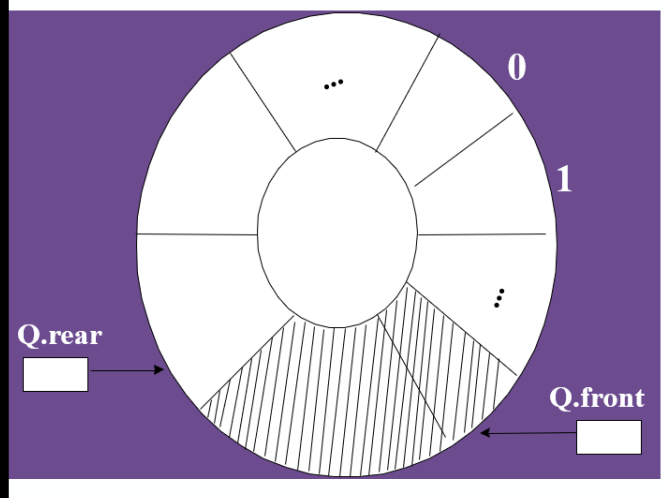


（a）一般情况　（b）队列空间被占满　（c）空队　（d）呈“满”状态的循环队列

图 3.12　循环队列中头、尾指针和元素之间的关系

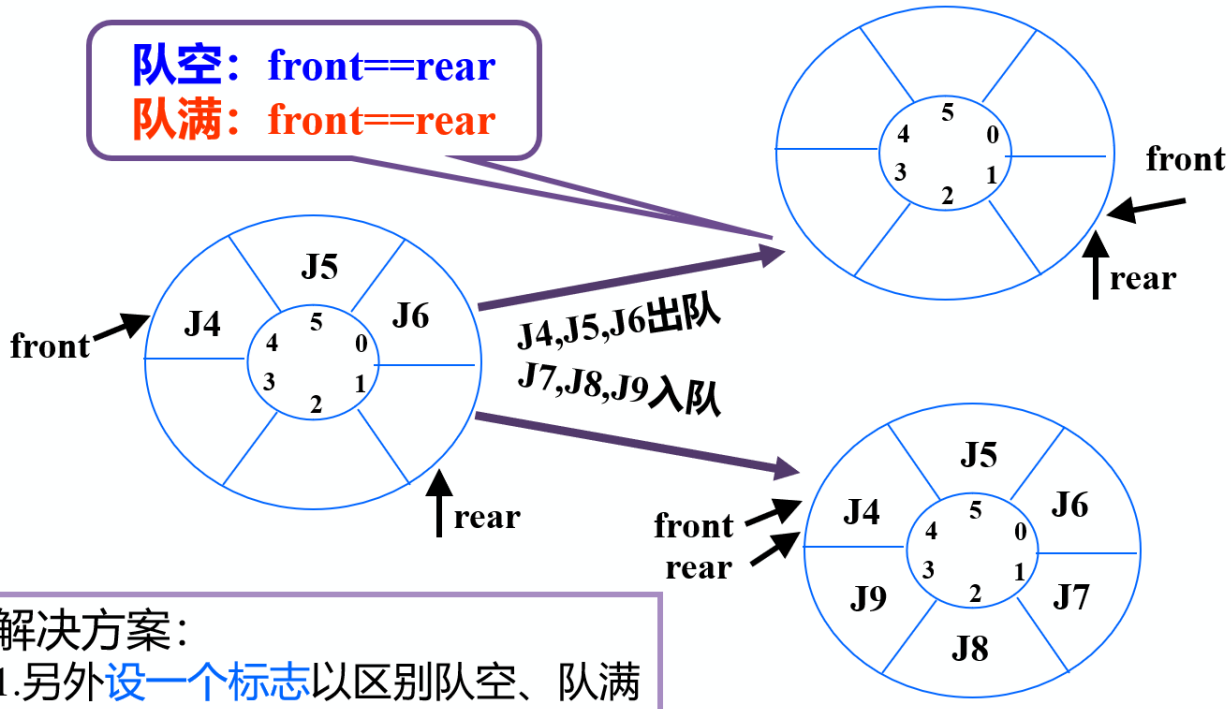
解决的方法 - - 循环队列

base[0]接在base[M-1]之后
若rear+1==M
则令rear=0;



实现：利用“模”运算
入队：
```
base[rear]=x;
rear=(rear+1)%M;
```
出队：
```
x=base[front];
front=(front+1)%M;
```

解决的方法 - - 循环队列



解决方案：
1.另外设一个标志以区别队空、队满
2.少用一个元素空间：
队空：front==rear
队满：(rear+1)%M==front

循环队列

```
#define MAXQSIZE  100  //最大长度
Typedef struct {
   QElemType *base;  //初始化的动态分配存储空间
   int  front;          //头指针
   int  rear;           //尾指针
}SqQueue;
```

循环队列初始化

```
Status InitQueue (SqQueue &Q){
    Q.base =new QElemType[MAXQSIZE]
   if(!Q.base) exit(OVERFLOW);
    Q.front=Q.rear=0;
    return OK;
}
```


求循环队列的长度

```
int  QueueLength (SqQueue Q){
   return (Q.rear-Q.front+MAXQSIZE)%MAXQSIZE;
 }
```

循环队列入队

```
Status EnQueue(SqQueue &Q,QElemType e){
    if((Q.rear+1)%MAXQSIZE==Q.front)  return ERROR;
    Q.base[Q.rear]=e;
    Q.rear=(Q.rear+1)%MAXQSIZE;
     return OK;
}
```


循环队列出队

```
Status DeQueue (LinkQueue &Q,QElemType &e){
   if(Q.front==Q.rear) return ERROR;
   e=Q.base[Q.front];
   Q.front=(Q.front+1)%MAXQSIZE;
   return OK;
}
```


2.1循环队列的相关条件和公式：

1. 队空条件：`rear==front`
2. 队满条件：`(rear+1) %QueueSIze==front`，其中QueueSize为循环队列的最大长度
3. 计算队列长度：`（rear-front+QueueSize）%QueueSize`
4. 入队：`（rear+1）%QueueSize`
5. 出队：`（front+1）%QueueSize`

3.链队列

```
typedef struct QNode{
    QElemType   data;
    struct Qnode  *next;
}Qnode, *QueuePtr;
typedef struct {
    QueuePtr  front;              //队头指针
    QueuePtr  rear;               //队尾指针
}LinkQueue;


```

链队列初始化

```
Status InitQueue (LinkQueue &Q){
   Q.front=Q.rear=(QueuePtr) malloc(sizeof(QNode));
    if(!Q.front) exit(OVERFLOW);
    Q.front->next=NULL;
     return OK;
}

```

销毁链队列

```
Status DestroyQueue (LinkQueue &Q){
   while(Q.front){
      Q.rear=Q.front->next;
      free(Q.front);
      Q.front=Q.rear;    }
   return OK;
}

```

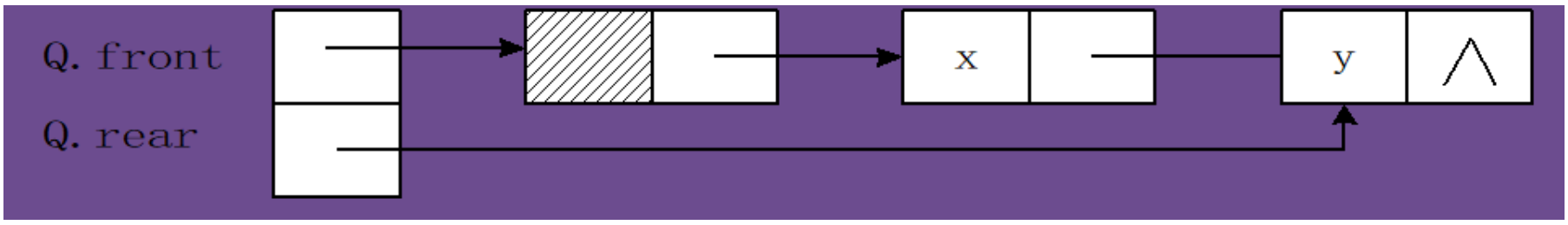
判断链队列是否为空

```
 Status QueueEmpty (LinkQueue Q)
{
    return (Q.front==Q.rear);
 }

```

求链队列的队头元素

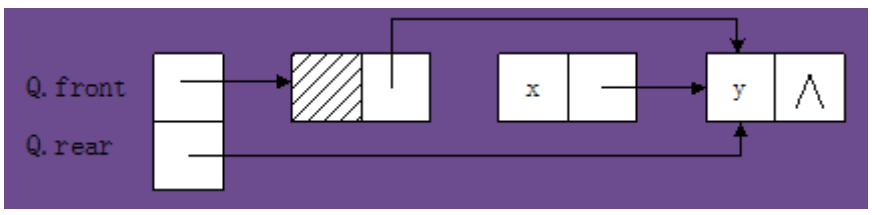
```
Status GetHead (LinkQueue Q, QElemType &e){
    if(Q.front==Q.rear) return ERROR;
    e=Q.front->next->data;//有头结点
    return OK;
}


```

链队列入队




```
Status EnQueue(LinkQueue &Q,QElemType e){
    p=(QueuePtr)malloc(sizeof(QNode));
    if(!p) exit(OVERFLOW);
    p->data=e; p->next=NULL;
    Q.rear->next=p;
    Q.rear=p;
    return OK;
}

```

链队列出队



4.链队列完整代码

```
Status DeQueue (LinkQueue &Q,QElemType &e){
    if(Q.front==Q.rear) return ERROR;
    p=Q.front->next;
    e=p->data;
    Q.front->next=p->next;
    if(Q.rear==p) Q.rear=Q.front;
    delete p;
    return OK;
}

```

```
#include "stdio.h"
#include "stdlib.h"
#include "io.h"
#include "math.h"
#include "time.h"
```

```c

#define OK 1
#define ERROR 0
#define TRUE 1
#define FALSE 0
#define MAXSIZE 20 /* 存储空间初始分配量 */

typedef int Status;

typedef int QElemType; /* QElemType类型根据实际情况而定，这里假设为int */

typedef struct QNode    /* 结点结构 */
{
    QElemType data;
    struct QNode *next;
} QNode, *QueuePtr;

typedef struct          /* 队列的链表结构 */
{
    QueuePtr front, rear; /* 队头、队尾指针 */
} LinkQueue;


Status visit(QElemType c)
{
    printf("%d ", c);
    return OK;
}


/* 构造一个空队列Q */
Status InitQueue(LinkQueue *Q)
{
    Q->front = Q->rear = (QueuePtr)malloc(sizeof(QNode));
    if (!Q->front)
        exit(OVERFLOW);
    Q->front->next = NULL;
    return OK;
}

/* 销毁队列Q */
Status DestroyQueue(LinkQueue *Q)
{
    while (Q->front)
    {
        Q->rear = Q->front->next;
        free(Q->front);
```

```
53          Q->front = Q->rear;
54      }
55      return OK;
56  }
57
58  /* 将Q清为空队列 */
59  Status ClearQueue(LinkQueue *Q)
60  {
61      QueuePtr p, q;
62      Q->rear = Q->front;
63      p = Q->front->next;
64      Q->front->next = NULL;
65      while (p)
66      {
67          q = p;
68          p = p->next;
69          free(q);
70      }
71      return OK;
72  }
73
74
75  /* 若Q为空队列,则返回TRUE,否则返回FALSE */
76  Status QueueEmpty(LinkQueue Q)
77  {
78      if (Q.front == Q.rear)
79          return TRUE;
80      else
81          return FALSE;
82  }
83
84  /* 求队列的长度 */
85  int QueueLength(LinkQueue Q)
86  {
87      int i = 0;
88      QueuePtr p;
89      p = Q.front;
90      while (Q.rear != p)
91      {
92          i++;
93          p = p->next;
94      }
95      return i;
96  }
97
98  /* 若队列不空,则用e返回Q的队头元素,并返回OK,否则返回ERROR */
99  Status GetHead(LinkQueue Q, QElemType *e)
```

```
{
    QueuePtr p;
    if (Q.front == Q.rear)
        return ERROR;
    p = Q.front->next;
    *e = p->data;
    return OK;
}


/* 插入元素e为Q的新的队尾元素 */
Status EnQueue(LinkQueue *Q, QElemType e)
{
    QueuePtr s = (QueuePtr)malloc(sizeof(QNode));
    if (!s) /* 存储分配失败 */
        exit(OVERFLOW);
    s->data = e;
    s->next = NULL;
    Q->rear->next = s;  /* 把拥有元素e的新结点s赋值给原队尾结点的后继，见图中① */
    Q->rear = s;    /* 把当前的s设置为队尾结点，rear指向s，见图中② */
    return OK;
}

/* 若队列不空,删除Q的队头元素,用e返回其值,并返回OK,否则返回ERROR */
Status DeQueue(LinkQueue *Q, QElemType *e)
{
    QueuePtr p;
    if (Q->front == Q->rear)
        return ERROR;
    p = Q->front->next;     /* 将欲删除的队头结点暂存给p，见图中① */
    *e = p->data;           /* 将欲删除的队头结点的值赋值给e */
    Q->front->next = p->next; /* 将原队头结点的后继p->next赋值给头结点后继，见图中②
*/
    if (Q->rear == p)   /* 若队头就是队尾，则删除后将rear指向头结点，见图中③ */
        Q->rear = Q->front;
    free(p);
    return OK;
}

/* 从队头到队尾依次对队列Q中每个元素输出 */
Status QueueTraverse(LinkQueue Q)
{
    QueuePtr p;
    p = Q.front->next;
    while (p)
    {
        visit(p->data);
```

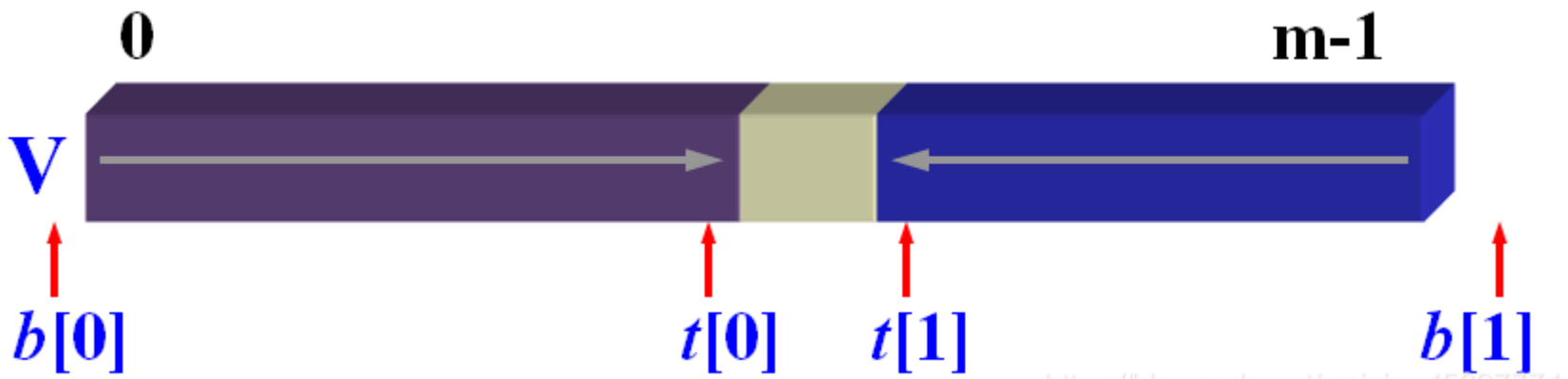
```
        p = p->next;
    }
    printf("\n");
    return OK;
}

int main()
{
    int i;
    QElemType d;
    LinkQueue q;
    i = InitQueue(&q);
    if (i)
        printf("成功地构造了一个空队列!\n");
    printf("是否空队列? %d(1:空 0:否)  ", QueueEmpty(q));
    printf("队列的长度为%d\n", QueueLength(q));
    EnQueue(&q, -5);
    EnQueue(&q, 5);
    EnQueue(&q, 10);
    printf("插入3个元素(-5,5,10)后,队列的长度为%d\n", QueueLength(q));
    printf("是否空队列? %d(1:空 0:否)  ", QueueEmpty(q));
    printf("队列的元素依次为: ");
    QueueTraverse(q);
    i = GetHead(q, &d);
    if (i == OK)
        printf("队头元素是: %d\n", d);
    DeQueue(&q, &d);
    printf("删除了队头元素%d\n", d);
    i = GetHead(q, &d);
    if (i == OK)
        printf("新的队头元素是: %d\n", d);
    ClearQueue(&q);
    printf("清空队列后,q.front=%u q.rear=%u q.front->next=%u\n", q.front,
q.rear, q.front->next);
    DestroyQueue(&q);
    printf("销毁队列后,q.front=%u q.rear=%u\n", q.front, q.rear);
    return 0;

}
```

三、考研试题（算法设计题）

1.

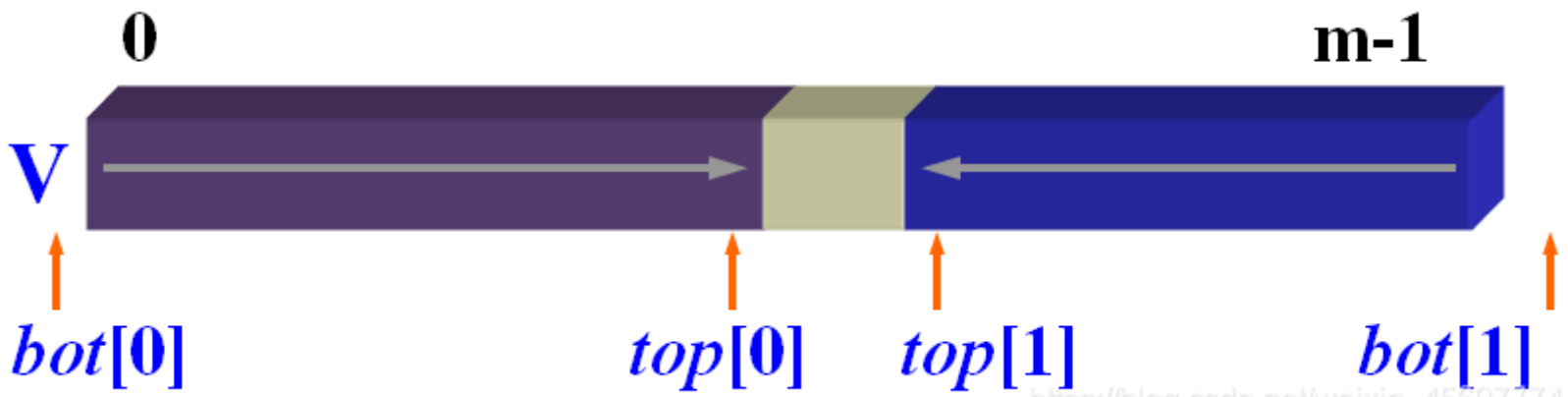
（对顶栈）

双栈共享一个栈空间



优点：互相调剂，灵活性强，减少溢出机会

将编号为0和1的两个栈存放于一个数组空间V[m]中，栈底分别处于数组的两端。当第0号栈的栈顶指针top[0]等于-1时该栈为空，当第1号栈的栈顶指针top[1]等于m时该栈为空。两个栈均从两端向中间增长（如下图所示） 。



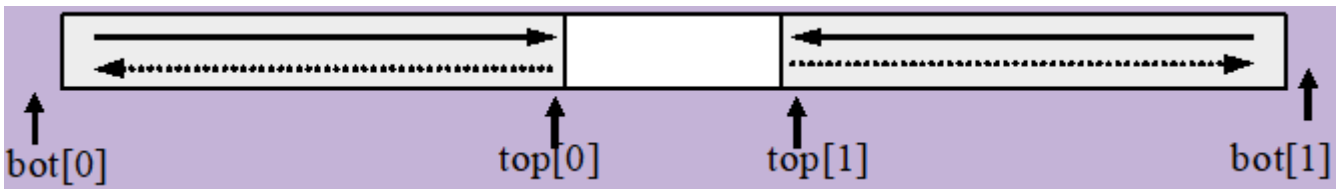

```
typedef struct
{
    int top[2], bot[2];      //栈顶和栈底指针
  SElemType *V; //栈数组
    int m;           //栈最大可容纳元素个数
}DblStack;

```

试编写判断栈空、栈满、进栈和出栈四个算法的函数(函数定义方式如下)

```
void Dblpush(DblStack &s,SElemType x,int i) ;
//把x插入到栈i的栈
int Dblpop(DblStack &s,int i,SElemType &x) ;
 //退掉位于栈i栈顶的元素
int IsEmpty(DblStack s,int i) ;
//判栈i空否，空返回1，否则返回0
int IsFull(DblStack s) ;
//判栈满否，满返回1，否则返回0
```

2.将编号为0和1的两个栈存放于一个数组空间V[m]中，栈底分别处于数组的两端。当第0号栈的栈顶指针top[0]等于-1时该栈为空；当第1号栈的栈顶指针top[1]等于m时，该栈为空。两个栈均从两端向中间增长。试编写双栈初始化，判断栈空、栈满、进栈和出栈等算法的函数。双栈数据结构的定义如下：

```
typedef struct{
  int top[2], bot[2];   //栈顶和栈底指针
  SElemType *V;         //栈数组
  int m;                //栈最大可容纳元素个数
}DblStack;
```



答案:

```
//初始化一个大小为m的双向栈s
Status Init_Stack(DblStack &s,int m)
{
  s.V=new SElemType[m];
  s.bot[0]=-1;
  s.bot[1]=m;
  s.top[0]=-1;
  s.top[1]=m;
  return OK;
}

```

```
//判栈i空否，空返回1，否则返回0
int IsEmpty(DblStack s,int i)
{
    return s.top[i] == s.bot[i];
}

```

```
//判栈满否，满返回1，否则返回0
int IsFull(DblStack s)
{
    if(s.top[0]+1==s.top[1])
        return 1;
    else return 0;
}

```

```
void Dblpush(DblStack &s,SElemType x,int i)
{
    if( IsFull (s ) )  exit(1);
        // 栈满则停止执行
    if ( i == 0 ) s.V[ ++s.top[0] ] = x;
   //栈0情形：栈顶指针先加1，然后按此地址进栈
   else s.V[--s.top[1]]=x;
   //栈1情形：栈顶指针先减1，然后按此地址进栈
}

```

```
int Dblpop(DblStack &s,int i,SElemType &x)
    {
    if ( IsEmpty ( s,i ) ) return 0;
        //判栈空否，若栈空则函数返回0
    if ( i == 0 ) s.top[0]--;    //栈0情形：栈顶指针减1
    else s.top[1]++; //栈1情形：栈顶指针加1
    return 1;
}

```

3.已知f为单链表的表头指针，链表中存储的都是整型数据，试写出实现下列运算的递归算法：
① 求链表中的最大整数；
② 求链表的结点个数；
③ 求所有整数的平均值。

```
int GetMax(LinkList p){//求链表中的最大整数
    if(!p→next)    return p→data;
    else {
        int max=GetMax(p→next);
        return  p→data≥max ? p→data:max;
    }
}

```

```
void main( ){
     LinkList L;
    CreatList(L);
    cout<<"链表中的最大整数为："<<GetMax(L→next)<<endl;
    ……}

```

四、作业习题

1.在一个具有n个单元的顺序栈中，假设栈底是存储地址的**高端**，现在我们以top作为栈顶指针，则作退栈操作时，top的变化是( )

A．top=top-1
B．top=top+1
C．top不变
D．top不确定

```
答案：B
```

这里跟正常的数组栈不一样，`top`是指针，**栈底是地址高的那一边**，所以最开始的时候top指向栈底，执行入栈操作的时候，`top--`，地址减小，执行出栈操作，`top++`，地址增加，向栈底方向移动。（**跟正常的正好相反**）

代码示例：

```
int a[5];
//栈底是高端地址
int base=4,top=4;
//top==base→空栈
if(top≥0){
    a[top]=20;
    top--;
}
```

若栈底是底端地址则就是正常的，跟上面相反

2.数组A[1..n]作为栈的存储空间，栈顶top的初值为n+1,在未溢出的情况表，以下（ ）完成入栈X操作。(2分)
A.top++; A[top]=X;
B.A[top]=X; top++;
C.top--; A[top]=X;
D.A[top]=X; top--;

答案：C

3.用链接方式存储的队列，在进行删除运算时( )

A．仅修改头指针
B．仅修改尾指针
C．头、尾指针都要修改
D．头、尾指针可能都要修改

答案：D

4*.数组Q［n］用来表示一个循环队列,f为当前队列头元素的前一位置,r为队尾元素的位置,假定队列中元素的个数小于n,计算队列中元素个数的公式为（ ）.
A．r-f
B．(n+f-r)%n
C．n+r-f
D．(n+r-f)%n

答案：D

循环队列,r可能小于f,例如n为4时,元素个数有0、1、2、3,r可以为0,f为2,这样实际上有两个元素,但是以r-f得出来的是-2.

5.循环队列的队满条件为( )

A．$(CQ.rear+1)\%maxsize == (CQ.front+1)\%maxsize$

B．$(CQ.rear+1)\%maxsize == CQ.front+1$

C．$(CQ.rear+1)\%maxsize == CQ.front$

D．$CQ.rear == CQ.front$

答案：C

约定循环队列的队头指针指示队头元素在数组中实际位置的前一个位置，队尾指针指示队尾元素在数组中的实际位置。当队尾指针“绕一圈”后赶上队头指针时，视为队满。

6.C语言数组Data[m+1]作为循环队列SQ的存储空间，front为队头指针，rear为队尾指针，则执行出队操作的语句为( )

A．front=front+1
B．front=(front+1)%m
C．rear=(rear+1)%m
D．front=(front+1)%(m+1)

答案 ： D

循环队列嘛，又不是双端队列，从头出。

7.循环队列用数组A[0..m-1]存放其元素值，已知其头尾指针分别是front和rear，则当前队列的元素个数是（）；该循环队列最多可放下（）个元素。

答案：(rear-front+m)%m，m-1

8.以下运算实现在链栈上的初始化，请在空白处用请适当句子予以填充。

```
typedef struct Node{
  DataType data;
  struct Node *next;
}StackNode,*LStackTp;
void InitStack(LStackTp &ls){
    ls=NULL(3分);
}
```

9.以下运算实现在链栈上的进栈，请在空白处用请适当句子予以填充。

```
1
2
3 void Push（LStackTp &ls,DataType x）{
4       LStackTp p;
5       p=(LStackTp)malloc(sizeof(StackNode));
6       p→data=x(2分);
7       p→next=ls;
8       ls=p(2分);
9 }
```

10.以下运算实现在链栈上的退栈，请在空白处用请适当句子予以填充。

```
1
2 int pop(LStackTp &ls,DataType &x){
3       LStackTp p;
4       if(ls≠NULL){
5           p=ls;
6           x=p→data(2分);
7           ls=ls→next;
8           free(p)(2分);
9           return(1);
10      }else
11              return(0);
12 }
```

11.队列结构的顺序存储会产生假溢出现象

12.

```
1  /* 阅读下面关于循环队列的程序,实现循环队列的入队和出队操作 */
2  /* 熟悉循环队列的结点类型，掌握循环队列在插入和删除元素在操作上的特点*/
3  /* 加深对循环队列的理解，逐步培养解决实际问题的编程能力*/
4  /* 程序填空，运行程序*/
5  #include
6  #include
7  #include
8  #include
9  #include
10 #include
11 #include
12 #include
13 #include /* exit() */
14
15 #define TRUE 1
16 #define FALSE 0
17 #define OK 1
18 #define ERROR 0
19 #define INFEASIBLE -1
20
```

```c
typedef int Status;
typedef int Boolean;
typedef int QElemType;

/* 队列的顺序存储结构(可用于循环队列和非循环队列) */
#define MAXQSIZE 5
typedef struct {
    QElemType *base;
    int front;
    int rear; //填写语句
} SqQueue;


/* 循环队列的基本操作(9个) */
Status InitQueue(SqQueue *Q) {
    (*Q).base = (QElemType *)malloc(MAXQSIZE * sizeof(QElemType)); //填写一条语句

    if (!(*Q).base)
        exit(OVERFLOW);

    (*Q).front = (*Q).rear = 0; //填写一条语句
    return OK;
}

Status DestroyQueue(SqQueue *Q) {
    if ((*Q).base)
        free((*Q).base);

    (*Q).base = NULL;
    (*Q).front = (*Q).rear = 0;
    return OK;
}

Status ClearQueue(SqQueue *Q) {
    (*Q).front = (*Q).rear = 0;
    return OK;
}

Status QueueEmpty(SqQueue Q) {
    if (Q.front == Q.rear) //括号内补充完整
        return TRUE;
    else
        return FALSE;
}

int QueueLength(SqQueue Q) {
```

```
67      return (Q.rear - Q.front + MAXQSIZE) % MAXQSIZE; //填写语句
68  }
69
70  Status GetHead(SqQueue Q, QElemType *e) {
71      if (Q.front == Q.rear)
72          return ERROR;
73
74      *e = *(Q.base + Q.front);
75      return OK;
76  }
77
78  Status EnQueue(SqQueue *Q, QElemType e) {
79      if (((*Q).rear + 1) % MAXQSIZE == (*Q).front) //此处补充完整判断条件表达式
80          return ERROR;
81
82      (*Q).base[(*Q).rear] = e;
83      (*Q).rear = ((*Q).rear + 1) % MAXQSIZE; //填写两条语句
84      return OK;
85  }
86
87  Status DeQueue(SqQueue *Q, QElemType *e) {
88      if ((*Q).front == (*Q).rear) //此处补充完整判断条件表达式
89          return ERROR;
90
91      *e = (*Q).base[(*Q).front];
92      (*Q).front = ((*Q).front + 1) % MAXQSIZE; //填写两条语句
93      return OK;
94  }
95
96  Status QueueTraverse(SqQueue Q, void(*vi)(QElemType)) {
97      int i;
98      i = Q.front;
99
100     while (i != Q.rear) {
101         vi(*(Q.base + i));
102         i = (i + 1) % MAXQSIZE;
103     }
104
105     printf("\n");
106     return OK;
107 }
108
109
110 void visit(QElemType i) {
111     printf("%d ", i);
112 }
113
```

```
void main() {
    Status j;
    int i = 0, l;
    QElemType d;
    SqQueue Q;
    InitQueue(&Q);
    printf("初始化队列后，队列空否？%u(1:空 0:否)\n", QueueEmpty(Q));
    printf("请输入整型队列元素(不超过%d个),-1为提前结束符: ", MAXQSIZE - 1);

    do {
        scanf("%d", &d);

        if (d == -1)
            break;

        i++;
        EnQueue(&Q,
                d); //循环体内填写语句，实现依次有4个（MAXQSIZE-1）元素入队，-1作为提前结束符。
    } while (iprintf("队列长度为: %d\n", QueueLength(Q));

            printf("现在队列空否？%u(1:空 0:否)\n", QueueEmpty(Q));
            printf("连续%d次由队头删除元素,队尾插入元素:\n", MAXQSIZE);
    for (l = 1; l ≤ MAXQSIZE; l++) {
        DeQueue(&Q, &d);
            printf("删除的元素是%d,请输入待插入的元素: ", d);
            scanf("%d", &d);
            EnQueue(&Q, d);
        }
    l = QueueLength(Q);
        printf("现在队列中的元素为: \n");
        QueueTraverse(Q, visit);
        printf("共向队尾插入了%d个元素\n", i + MAXQSIZE);

        if (l - 2 > 0)
        printf("现在由队头删除%d个元素:\n", l - 2);
    while (QueueLength(Q) > 2) {
        DeQueue(&Q, &d);
            printf("删除的元素值为%d\n", d);
        }
    j = GetHead(Q, &d);

        if (j)
        printf("现在队头元素为: %d\n", d);
        ClearQueue(&Q);
        printf("清空队列后, 队列空否？%u(1:空 0:否)\n", QueueEmpty(Q));
        DestroyQueue(&Q);
```

```
160 }
```

五、数据结构进阶

如果想要了解进阶数据结构，可以点击下方链接（涉及到竞赛方面内容）

> **0x11.基本数据结构 —栈与单调栈**

> **0x12.基本数据结构 — 队列与单调队列**