**栈的基本概念**

**1.栈的定义**

栈是一种只能在一端进行插入或删除操作的**线性表**。其中允许进行插入或删除操作的-端称为**栈顶**

**(Top)。**栈顶由一个称为栈顶指针的位置指示器(其实就是一个变量，对于顺序栈，就是记录栈项元素所在数组位置标号的一个整型变量:对于链式栈，就是记录栈顶元素所在结点地址的指针)来指示，它是动态变化的。表的另一端称为**栈底**，栈底是固定不变的。栈的插入和删除操作一般称为**入栈和出栈**。

栈可以依照存储结构分为两种:顺序栈和链式栈。

1. 顺序栈定义

**typedef struct**

**{**

**int data [maxSize] ; //存放栈中元素，maxSize是已定义的常量**

**int top; //栈顶指针**

**} SqStack; //顺序栈类型定义**

1. 链栈结点定义

**typedef struct LNode**

**{**

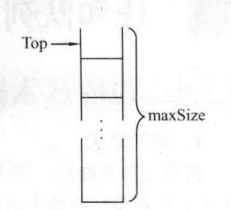
**int data;//数据域**

**struct LNode \*next; //指针域**

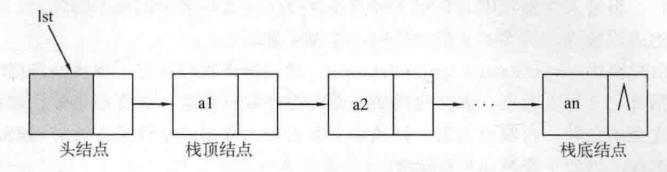
**} LNode; //链栈结点定义**

链栈就是采用链表来存储栈。这里用带头结点的单链表来作为存储体，

顺序栈示意图:



链栈示意图:



**队列的基本概念**

1.队列的定义

队列简称队，它也是一种操作受限的线性表，其限制为仅允许在表的一端进行插入， 在表的另一端进行删除。可进行插入的一端称为队尾(Rear), 可进行删除的一端称为队头(Front)。 向队列中插入新的元素称为进队，新元素进队后就成为新的队尾元素;从队列中删除元素称为出队，元素出队后，其后继元素就成为新的队头元素。

2.队列的特点

队列的特点概括起来就是先进先出(FIFO)。 打个比方，队列就好像开进隧道的一列火车，各节车

厢就是队中的元素，最先开进隧道的车厢总是最先驶出隧道。

3.队列的存储结构

可用**顺序表**和**链表**来存储队列，队列按存储结构可分为**顺序队**和**链队**两种。

3.顺序队列定义

**typedef struct**

**{**

**int data [maxSize] ;**

**int front; //队首指针**

**int rear; / /队尾指针**

**} SqQueue ; / /顺序队类型定义**

**4.链队定义**

(1)队结点类型定义

**typedef struct QNode**

**{**

**int data; / /数据域**

**struct QNode \*next; / /指针域**

**}QNode; / /队结点类型定义**

(2) 链队类型定义

**typedef struct**

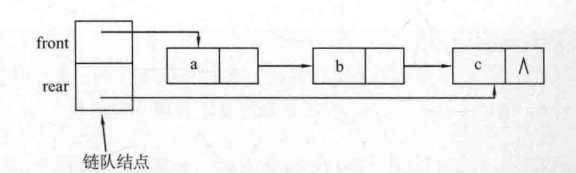
**{**

**QNode \* front;//队头指针**

**QNode \*rear;//队尾指针**

**} LiQueue;/ /链队类型定义**

链队示意图：



1.顺序栈的要素

对于顺序栈st,一共有4个要素，包括两个特殊状态和两个操作。

(1)几个状态

**1)栈空状态。**

栈空状态为st.top==-1。考试中有时会出现其他规定，其实大同小异，稍加注意即可。

**2)栈满状态。**

st.top==maxSize-1. maxSize为栈中最大元素的个数，则maxSize-1为栈满时栈顶元素在数组中的位

置，因为数组下标从0号开始。本书规定栈顶指针top为-1时栈空，即top==0的数组位置也可以存有数据元素。

**3)非法状态(上溢和下溢)。**

栈满就是一种继续入栈就会上溢的状态，对应的栈下溢就是栈空的时候继续出栈所造成的结果。

**(2)两个操作**

**1)元素x进栈操作: ++(st.top); st.data[st.top]=x;**

1. **元素x出栈操作: x=st.data[st.top]; --(st.top);**

**2.初始化栈的代码**

初始化一个栈，只需将栈顶指针置为-1即可，其对应代码如下:

**void initStack(SqStack &st) / /初始化栈**

**{**

**st. top=-1;//只需将栈顶指针设置为-1**

**}**

**3.判断栈空代码**

栈st为空的时候返回1,否则返回0，其对应代码如下:

**int isEmpty (SqStack st)**

**{**

**if(st. top==-1)**

**return 1;**

**else**

**return 0;**

**}**

**4.进栈代码**

**int push (SqStack &st, int x)**

**{**

**if (st. top==maxSize-1) / /这里要注意，栈满不能进栈**

**return 0;**

**++ (st. top) ; //先移动指针，再进栈**

**st.data[st.top] = x;**

**return 1 ;**

**}**

**5.出栈代码**

**int pop (SqStack &st, int &x)**

**{**

**if (st. top==-1)//注意，如果栈空，则不能出栈**

**return 0;**

**x=st.data[st.top] ;//先取出元素，再移动指针**

**--(st.top) ;**

**return 1 ;**

**}**

(1)定义一个栈并且初始化

假设元素是int型，可以这么写:

//这两句话连定义带初始化都有了，当然，栈顶指针和元素的类型要根据题目而定

**int stack[maxSize]; int top= 1;**

(2)元素x进栈

**stack[++top]=x;**  //仅一句即实现进栈操作

(3)元素x出栈

**x=stack[top - 1];**  //仅一句即实现出栈操作

**链栈**

**1.链栈的要素**

和顺序栈对应，链栈也有4个要素，包括两个特殊状态和两个操作。

**(1)两个状态**

1)栈空状态。

**lst->next==NULL**

2)栈满状态。

不存在栈满的情况(假设内存无限大的情况下不存在。一般题目要求不太严格，可以这么认为)。

**(2)两个操作**

1)元素(由指针p所指)进栈操作。

**p->next=lst->next; 1st->next=p;** //其 实就是头插法建立链表中的插入操作

2)出栈操作(出栈元素保存在x中)。

**p=lst->next; x=p->data; lst->next=p->next; free(p);** //其实就是单链表的删除操作

2.链栈的初始化代码

**void initStack (LNode\*&lst)**  //lst 要改变，用引用型

**{**

**lst= (LNode\*) malloc (sizeof (LNode));**  //制造一个头结点

**lst->next=NULL;**

**}**

**3.判断栈空代码**

当栈空时返回1，否则返回0,代码如下:

**int isEmpty (LNode \*lst)** //判断是否为空

**{**

**if (1st->next==NULL)**

**return 1;**

**else**

**return 0;**

**}**

**4.进栈代码**

void push (LNode \*lst, int x)

{

**LNode \*p;**

**p= (LNode\*) malloc (sizeof (LNode)) ; //为进栈元素申请结点空间**

**p->next=NULL;**

**/\*以下3句就是链表的头插法\*/**

**p->data=x;**

**P->next=lst->next;**

**lst->next=p;**

}

**5.出栈代码**

在栈不空的情况下可以执行，返回1，否则返回0，代码如下:

**int pop(LNode \*lst, int &x)//需要改变的变量要用引用型**

**{**

**LNode \*p;**

**if (1st->next==NULL) //栈空则不能出栈，返回0**

**return 0;**

**/\*以下就是单链表的删除操作\*/**

**p=lst->next;**

**x=p->data;**

**lst->next=p->next;**

**free(p) ;**

**return 1;**

**}**

顺序队

1.循环队列

在顺序队中，通常让队尾指针rear指向刚进队的元素位置,让队首指针front指向刚出队的元素位置。因此，元素进队的时候，rear要向后移动;元素出队的时候，front也要向后移动。这样经过一系列的出队和进队操作以后，两个指针最终会达到数组末端maxSize-1处。虽然队中已经没有元素，但仍然无法让元素进队，这就是所谓的“**假溢出**”。要解决这个问题，可以把数组弄成一一个环，让rear和front沿着环走，这样就永远不会出现两者来到数组尽头无法继续往下走的情况，这样就产生了循环队列。循环队列是改进的顺序队列。所示为循环队列元素的进队/出队示意图。

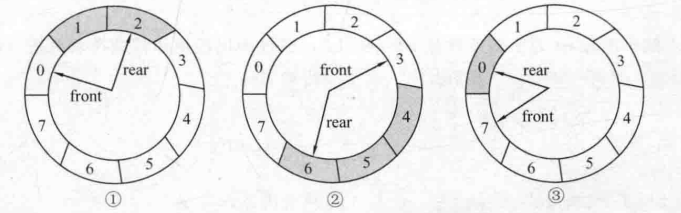


图3-4中进队/出队的变化情况如下:

①由空队进队两个元素，此时front指向0，rear指向2。

②进队4个元素，出队3个元素，此时front指向3，rear 指向6。

③进队2个元素，出队4个元素，此时front指向7，rear指向0。

从图3-4中由①到③的变化过程可以看出，经过元素的进进出出，即便是rear和front都到了数组尾端(图3-4中③所示)，依然可以让元素继续入队，因为两指针不是沿着数组下标递增地直线行走，而是沿着一个环行走，走到数组尽头的时候自动返回数组的起始位置。

要实现指针在递增的过程中沿着环形道路行走，有一个方法，就图3-4中的例子，拿front指针来说，可以循环执行语句front=(front+1)%8，若front的初值为0，在一- 个无限循环中，则front 的取值为0, 1, 2,3, 4，5，6, 7, 0, 1, 2...即以0~7为周期的无限循环数，也就是front沿着图3-4所示的环在行走。对于一.般情况，上述语句可写为**front=(front+1)%maxSize** ( maxSize是数组长度)。图3-5所示为循环队列两个特殊的状态：**队满和队空**(rear 的情况和front类似)。

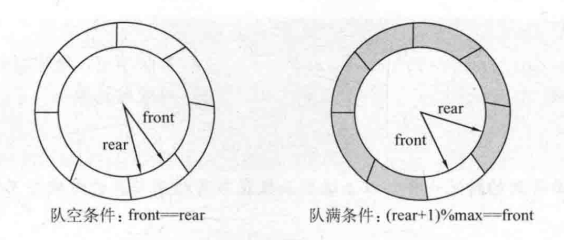
由图可以看出，循环队列必须损失一个存储空间，如果右图中的空白处也存入元素，则队满的条件也成了front==rear, 即和队空条件相同，那么就无法区分队空和队满了。

**2.循环队列的要素**

通过以上讲述可以总结出循环队列的4个要素。

**(1)两个状态**

1)队空状态: **qu.rear==qu.front。**



2)队满状态: **(qu.rear+ 1)%maxSize== qu.front。**

**(2)两个操作**

1)元素x进队操作(移动队尾指针)。

**qu.rear= (qu. rear+1) % maxSize;qu.data[qu.rear]=x;**

2)元素x出队操作(移动队首指针)。

**qu. front= (qu. front+1)% maxSize; x=qu.data[qu. front] ;**

3.初始化队列算法

**void ini tQueue (SqQueue &qu)**

**{**

**qu. front=qu.rear=0; //队首和队尾指针重合，并且指向0**

**}**

4.判断队空

**int isQueueEmpty (SqQueue qu)**

**{**

**if(qu. front==qu.rear) //不论队首、队尾指针指向数组中的哪个位置**

**return 1; //只要两者重合，即为队空**

**else**

**return 0;**

**}**

5.进队算法

**int enQueue (SqQueue &qu, int x)**

**{**

**if( (qu. rear+1) maxSize==qu. front)//队满的判断条件，队满则不能入队**

**return 0;**

**qu.rear= (qu. rear+1) maxSize;/ /若队未满，则先移动指针**

**qu.data[qu.rearl=x;/ /再存入元素**

**return 1;**

**}**

6.出队算法

**int deQueue (SqQueue &qu, int &x)**

**{**

**if (qu. front==qu. rear)//若队空，则不能出队**

**return 0;**

**qu. front= (qu. front+1) &maxSize;//若队不空，则先移动指针**

**x=qu.data[qu. front] ;//再取出元素**

**return 1;**

**}**

**链队**

链队就是采用链式存储结构存储队列，这里采用单链表来实现。链队的特点就是不存在队列满上溢的情况(其实这里也不太严格，内存满了就上溢了)。

**1.链队的要素**

链队也有两个特殊状态和两个操作。

**(1)两个状态**

1)队空状态。

lqu->rear==NULL或者1qu->front==NULL (为什么有两个，后边解释)

2)队满状态。

不存在队列满的情况(假设内存无限大的情况下不存在)。

**(2)两个操作**

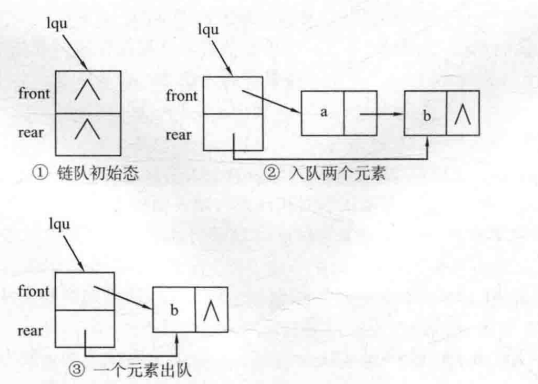
1)元素进队操作(假设p指向进队元素)。

**lqu->rear->next=p; lqu->rear=p;**

2)元素出队操作(假设x存储出队元素)。

**p=lqu->front; 1qu->front=p->next;x=p->data; free(p) ;**

图显示了一个链队的动态变化过程。由图3-6可以看出，front 和rear任何-一个为空都可以用来判定链队为空。



2.初始化链队算法

**void initQueue (LiQueue \*&lqu) / /初始化队列**

**{**

**lqu= (LiQueue\* )malloc (sizeof (LiQueue)) ;**

**lqu->front=lqu->rear=NULL;**

**}**

3.判断队空的算法

**int isQueueEmpty (LiQueue \*1qu)/ /判断队空**

**{**

**if (1qu->rear==NULLI 11qu->front==NULL)**

**return 1;**

**else**

**return 0;**

**}**

4.入队算法

**void enQueue (LiQueue \*1qu,int x)**

**{**

**QNode \*p;**

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

**p->data=x;**

**p->next=NULL;**

**if (lqu->rear==NULL)/ /若队列为空，则新结点是队首结点，也是队尾结点**

**lqu->front=lqu->rear=p;**

**else**

**{**

**lqu->rear->next=p;//将新结点链接到队尾，rear指向它**

**lqu->rear=p;**

**}**

**}**

**5.出队代码**

**int deQueue (LiQueue \*1qu, int &x)**

**{**

**QNode \*p;**

**if (1qu->rear=-NULL) //队空不能出队**

**return 0;**

**else**

**p=lqu->front;**

**if(1qu->front==lqu->rear) //队列中只有一个结点时的出队操作需特殊处理**

**lqu->front=lqu->rear=NULL;**

**else**

**lqu->front=lqu-> front->next;**

**x=p->data;**

**free (p) ;**

**return 1;**

**}**

(一)选择题

**1.栈操作数据的原则是( )。**

A.先进先出

B.后进先出

C.后进后出

D.不分顺序

2.在做进栈运算时，应先判别栈是否(①);在做退栈运算时，应先判别栈是否(②)。当栈中

元素为n个，做进栈运算时发生上溢，则说明该栈的最大容量为(③)。

①，②:A.空 B.满 C.，上溢 D.下溢

③: A. n-1 B. n C. n+1 D. n/2

**3.一个栈的输入序列为1, 2, 3，.. n,若输出序列的第一个元素是n,则输出序列的第i(1≤i≤n)个元素是( )。**

A.不确定

B. n-i+1

C. i

D. n-i

**4.若一个栈的输入序列为1, 2, 3，., n,输出序列的第-一个元素是i,则第j个输出元素是( )。**

A. i-j-1

B. i-j

C. j-i+1

D.不确定的

**5.有6个元素以6,5，4,3，2,1的顺序进栈，()不是合法的出栈序列。**

A.5，4，3，6，1，2

B.4，5，3，1，2，6

C.3，4，6，5，2，1

D.2，3，4，1，5，6

**6.输入序列为A，B，C，当可以变为C，B, A时，经过的栈操作为( )。**

A. push,pop,push,pop,push.pop

B. push,push.push.pop.pop.pop

C. push,push,pop,pop,push,pop

D. push,pop,push,push,pop,pop

**7.若一个栈以向量V[1， .... n]存储，初始栈顶指针top为n+1,则x进栈的正确操作是( )。**

A. top-top+1; V[top]=x;

B. V[top]=x; top=top+1;

C. top-top - 1;V[top]=x;

D. V[top]=x; top=top -1;

**8.若栈采用顺序存储方式存储，现两栈共享空间V[1, .. m], top[i]代表第i (i=1, 2)个栈的栈顶，栈1的底 在V[1], 栈2的底在V[m]， 则栈满的条件是( )。**

A. |top[2]-top[1]|=0

B. top[1]+1=top[2]

C. top[1]+top[2]=m

D. top[1]=top[2]

**9.栈在()中应用。**

A.递归调用

B.子程序调用

C.表达式求值

D.A,B,C

**10.表达式a\*(b+c)-d的后缀表达式是( )。**

A. abcd\*+-

B. abc+\*d-

C. abc\*+d-

D. -+\*abcd

**11.设计一个判别表达式中左、右括号是否配对出现的算法，采用( )数据结构最佳。**

A.线性表的顺序存储结构

B.队列

C.线性表的链式存储结构

D.栈

**12.对于链队，在进行出队操作时( ? )。**

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

**13.用不带头结点的单链表存储队列时，其队头指针指向队头结点，其队尾指针指向队尾结点，则**

**在进行出队操作时( )。**

A.仅修改队头指针

B.仅修改队尾指针

C.队头、队尾指针都要修改

D.队头、队尾指针都可能要修改

**14.递归过程或函数调用时，处理参数及返回地址要用一种称为()的数据结构。**

A.队列

B.多维数组.

C.栈

D.线性表

**15.循环队列存储在数组A[0，... m]中，则入队时的操作为(H )。**

A. rear-rear+1

B. rear=(rear+1)%(m- 1)

C. rear=(rear+ 1)%m

D. rear=(rear+ 1)%(m+1)

**16. 若用一个大小为6的数组来实现循环队列，并且当前rear和front的值分别为0和3，当从队列**

**中删除一个元素，再加入两个元素后，rear 和front的值分别为( )。**

A.1和5

B.2和4

C.4和2

D.5和1

**17.最大容量为maxSize的循环队列，队尾指针是rear,队头是front, 则队空的条件是( )。**

A. (rear+1)%maxSize=-front

B. rear=-front

C. rear+ 1==front

D. (rear-1)%maxSize=-front

**18.栈和队列的共同点是( )。**

A.都是先进先出

B.都是先进后出

C.只允许在端点处插入和删除元素

D.没有共同点

**19.栈和队都是( )。**

A.顺序存储的线性结构

B.链式存储的非线性结构

C.限制存取点的线性结构

D.限制存取点的非线性结构

**20.元素A，B, C, D依次进顺序栈后，栈顶元素是( )， 栈底元素是( )。**

A. A

B. B

C.C

D. D

**21.经过以下栈的操作后，x的值为( )。**

**initStack(st); push(st,a); push(st,b); pop(st,x); getTop(st,x);**

A. a

B. b

C.1

D.0

**22.经过以下栈的操作后，isEmpty(st)的返 回值为( )。**

**initStack(st); push(st,a); push(st,b);pop(st,x); pop(st,y);**

A. a

B.b

C.1

D.0

**23.设n个元素进栈序列是1, 2，3， ，n,其输出序列是p，P2，P3， .，Pn。若p=3，则**

**P2的值()。**

A.一定是2

B.一定是1

C.不可能是1

D.以上都不对

**24.最不适合用作链栈的链表是(以下链表没有头结点) ( )。**

A.只有表头指针、没有表尾指针的循环双链表

B.只有表尾指针、没有表头指针的循环双链表

C.只有表尾指针、没有表头指针的循环单链表

D.只有表头指针、没有表尾指针的循环单链表

**25.如果以链表作为栈的存储结构，则元素出栈的时候()。**

A.必须判断链栈是否为满

B.必须判断链栈元素的类型

C.必须判断链栈是否为空

D.无须做任何判断

**26.一个队列的入队列序列为1, 2, 3，4,则可能的出队序列为( )。**

A.4, 3, 2，1

B.1，2，3，4

C.1,4，3，2

D.3，2，4,1

**27.设循环队列的下标范围是0~n-1,其头、尾指针分别为f和r，则其元素个数为( )。**

A. r-f

B. r-f-1

C. (r-f)%n+1

D. (r-f+n)%n

**28.最适合用作链队的链表(链表有头结点，有队首指针则指向头结点，有队尾指针则指向终端结**

**点)是( )。**

A.只带队首指针的循环单链表

B.只带队尾指针的循环单链表

C.只带队首指针的非循环单链表

D.只带队尾指针的非循环单链表

**29.最不适合用作队列的链表是( )。**

A.只带队首指针的非循环双链表

B.只带队首指针的循环双链表

C.只带队尾指针的循环双链表

D.只带队尾指针的循环单链表

**30. (多选)用单链表(含有头结点)表示的队列的队头在链表的( ) 位置。**

A.链头

B.链尾

C.链中

D.以上都可以

**31.用单链表表示的链队的队尾在链表的( )位置。**

A.链头

B.链尾

C.链中

D.以上都可以