
第四章 堆栈和队列

在计算机领域

程序设计

枚举法、回溯法
递归程序的执行过程

编译程序

变量的存储空间的分配
表达式的翻译与求值计算

操作系统

作业调度、进程调度
内存空间的分配

. . .

堆栈

队列

本章内容

4.1 堆栈的基本概念

4.2 堆栈的顺序存储结构

4.3 堆栈的链式存储结构

4. 4 堆栈的应用举例(习题课)

4.5 队列的基本概念

4.6 队列的顺序存储结构

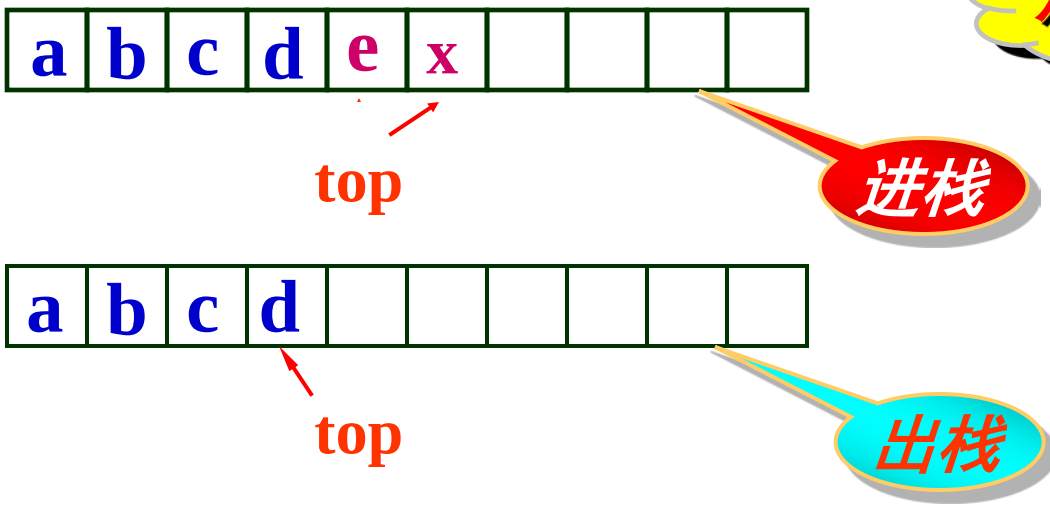
4.7 队列的链式存储结构

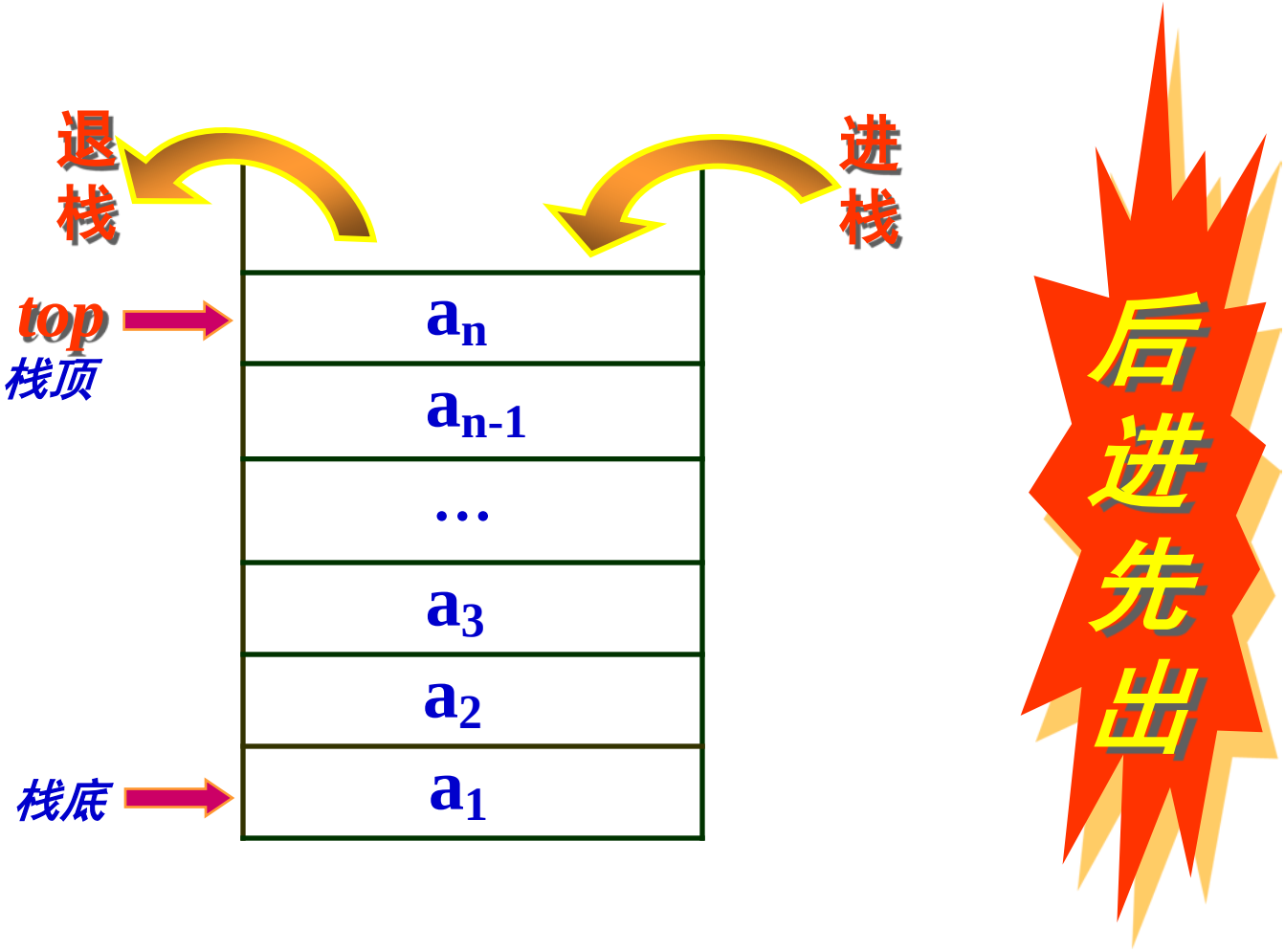
4.1 堆栈的基本概念

一. 堆栈的定义

堆栈是一种只允许在表的一端进行插入操作和删除操作的线性表。允许操作的一端称为栈顶，栈顶元素的位置由一个称为栈顶指针的变量给出。当表中没有元素时，称之为空栈。

后进先出




退栈
进栈
top
栈顶
a_n
a_{n-1}
…
a_3
a_2
a_1
栈底
后进先出

二. 堆栈的基本操作

1. 插入（进栈、入栈） √
2. 删除（出栈、退栈） √
3. 测试堆栈是否为空 √

主要操作

4. 测试堆栈是否已满
5. 检索当前栈顶元素

特殊性

1. 其操作仅仅是一般线性表的操作的一个 子集。
2. 插入和删除操作的位置受到限制。

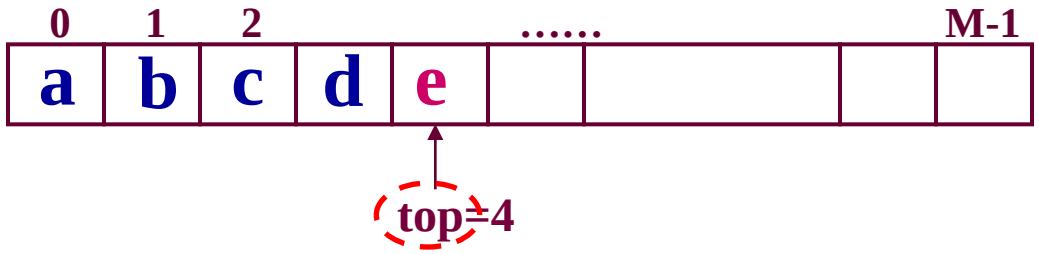
4.2 堆栈的顺序存储结构

一. 构造原理

顺序堆栈

描述堆栈的顺序存储结构最简单的方法是利用一维数组 **STACK[0..M–1]** 来表示, 同时定义一个整型变量（不妨取名为**top**) 给出栈顶元素的位置。

STACK[0.. M-1]



数组： 静态结构
堆栈： 动态结构

溢出

上溢 — 当堆栈已满时做插入操作。 **(top=M–1)**
下溢 — 当堆栈为空时做删除操作。 **(top=–1)**

类型定义

```
#define M     1000
SElemType  STACK[M];
int  top;
```

初始时，top=-1

二. 堆栈的基本算法

1. 初始化堆栈

```
void  INITIALS( int &top )
{
      top= –1;
}
```

2. 测试堆栈是否为空

栈空,返回1,否则,返回0。

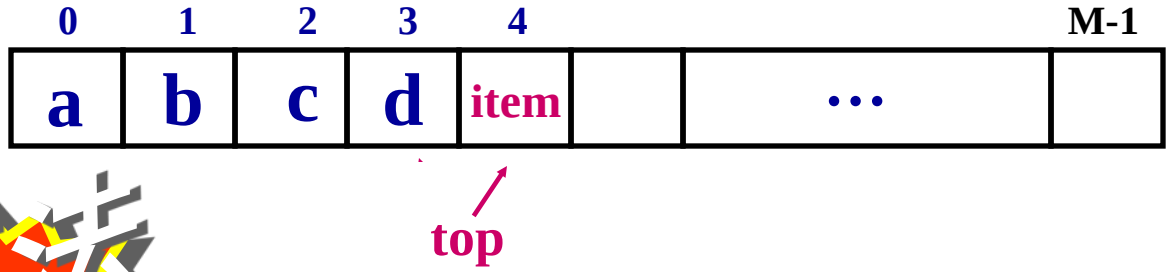
```
int  EMPTYS( int top )
{
      return top== –1;
}
```

3. 测试堆栈是否已满

```
int FULLS( int top )
{
    return top==M–1;
}
```

栈满,返回1,否则,返回0。

4. 插入(进栈)算法

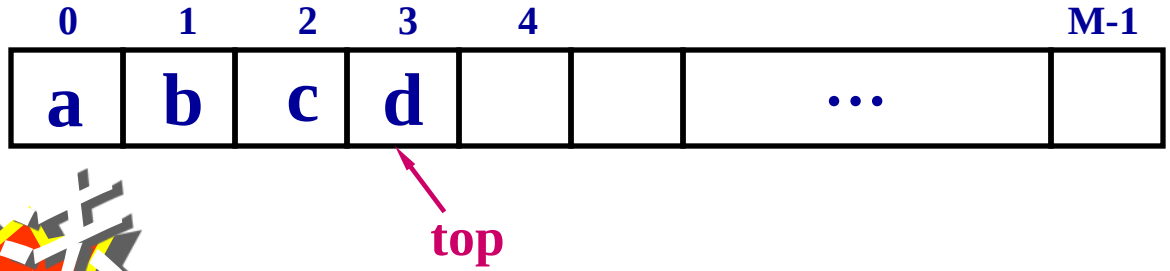


算法

```
int  PUSH( SElemType STACK[ ],  int &top,
                SElemType item )
{
        if( FULLS(top) )
              return 0;
        else {
              STACK[++top]=item;
              return 1;
         }
}
```

插入成功,返回1,
否则,返回0。

5. 删除(退栈)算法




```
int POP( SElemType STACK[ ], int &top,
                SElemType &item )
{
    if( EMPTYS(top) )
        return 0;
    else{
            item=STACK[top--];
            return 1;
    }
}
```

删除成功,返回1,
否则,返回0。

三. 多栈共享连续空间问题

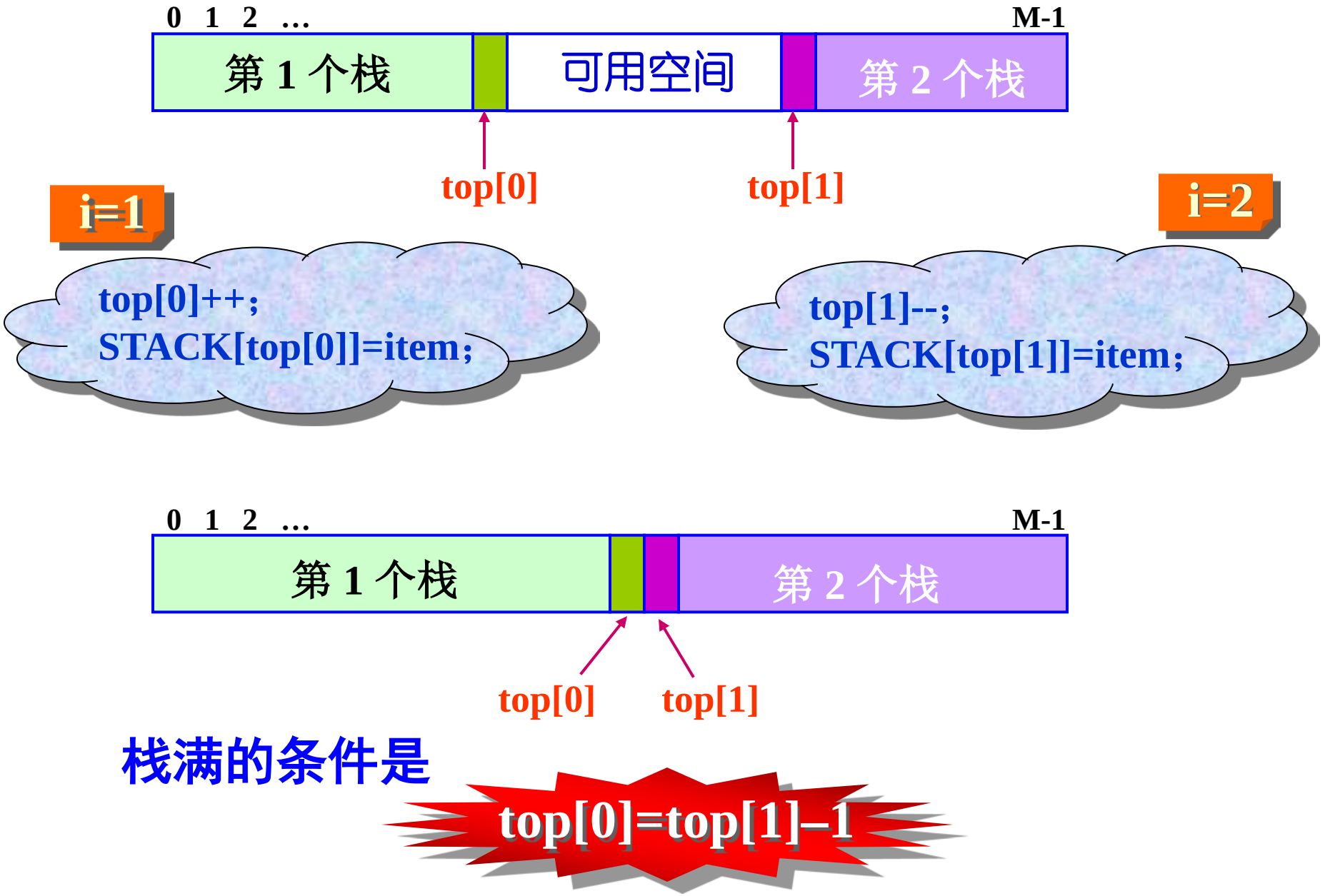
(以两个堆栈共享一个数组为例)

STACK[0: M-1]

top[0]、 top[1] 分别为第1个与第2个栈的栈顶元素指针。

插入：

当i=1时，将item 插入第1个栈，
当i=2时，将item 插入第2个栈。

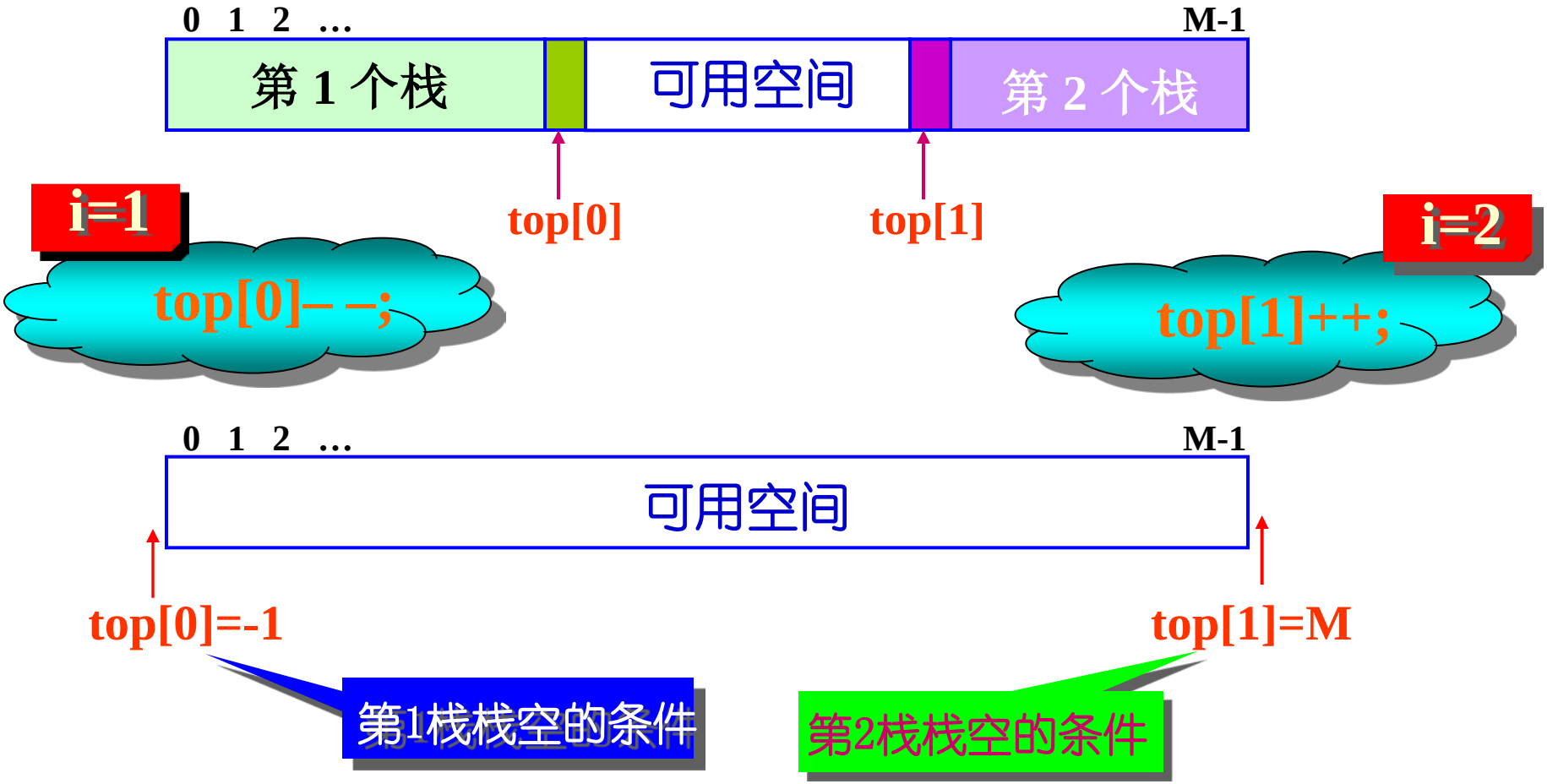

0 1 2 …
M-1
第 1 个栈
可用空间
第 2 个栈
top[0]
top[1]
i=1
top[0]++;
STACK[top[0]]=item;
i=2
top[1]--;
STACK[top[1]]=item;
0 1 2 …
M-1
第 1 个栈
第 2 个栈
top[0]
top[1]
栈满的条件是
top[0]=top[1]–1


```
int  PUSH( SElemType STACK[ ], int top[ ],
                int i, SelemType item )
{
     if (top[0]==top[1]–1)            /* 栈满 */
          return 0;
     else {
           if (i==1)                        /* 插入第1个栈 */
                STACK[++top[0]]=item;
           else                             /* 插入第2个栈 */
                STACK[– –top[1]]=item;
           return 1;
      }
}
```


```
int  PUSH( SElemType STACK[ ], int top[ ],
                int i, SElemType item )
{
      if (top[0]==top[1]–1)
            return 0;
      else {
            if (i==1)
                  top[0]++;
            else
                  top[1]– –;
            STACK[top[i–1]]=item;
            return 1;
      }
}
```

删除：

当i=1时，删除第1个栈的栈顶元素，
当i=2时，删除第2个栈的栈顶元素。




```
int  POP( SElemType STACK[ ], int top[ ],
          int i, SElemType &item )
{
    if (i==1)
        if (top[0]== –1)
            return 0;
        else {
            item=STACK[top[0]– –];
            return 1;
        }
    else
        if   (top[1]==M)
            return 0;
        else {
            item=STACK[top[1]++];
            return 1;
        }
}
```

对第一个栈进行操作

对第二个栈进行操作

4.3 堆栈的链式存储结构

一. 构造原理

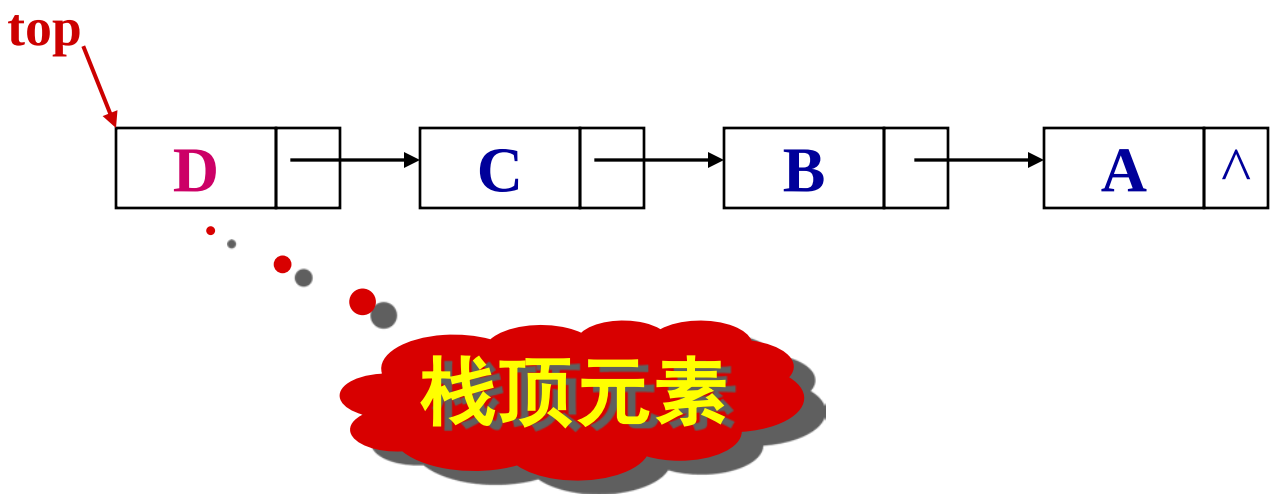
链接堆栈
链栈

链接堆栈就是用一个线性链表来实现一个堆栈结构，同时设置一个指针变量（这里不妨仍用**top**表示）指出当前栈顶元素所在链结点的位置。当栈为空时，有**top=NULL**。

在一个初始为空的链接堆栈中依次插入数据元素

A，B，C，D

以后，堆栈的状态为



类型定义

```
typedef  struct  node {
        SElmeType  data;
        struct  node  *link;
} STNode, *STLink;
```

二. 基本算法

1. 堆栈初始化

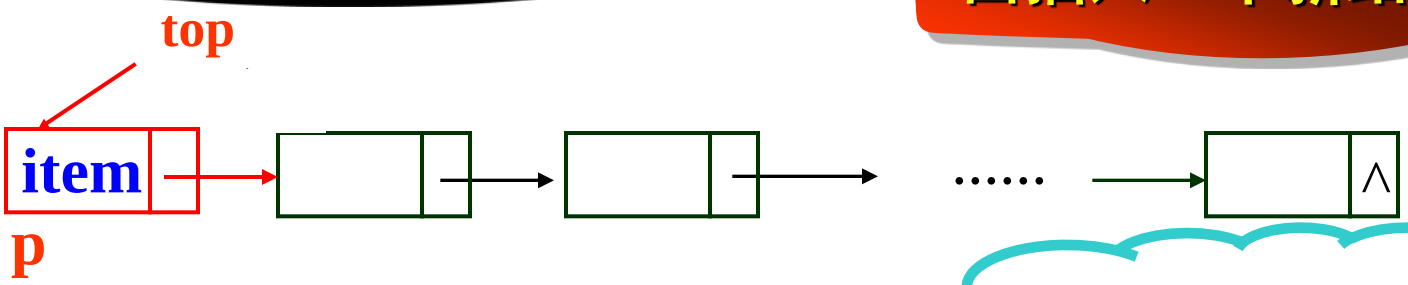
```
void INITIAL( STLink &top )
{
      top=NULL;
}
```

2. 测试堆栈是否为空

```
int  EMPTYS( STLink top )
{
      return  top==NULL;
}
```

3. 插入(进栈)算法

等效于在链表最前面插入一个新结点



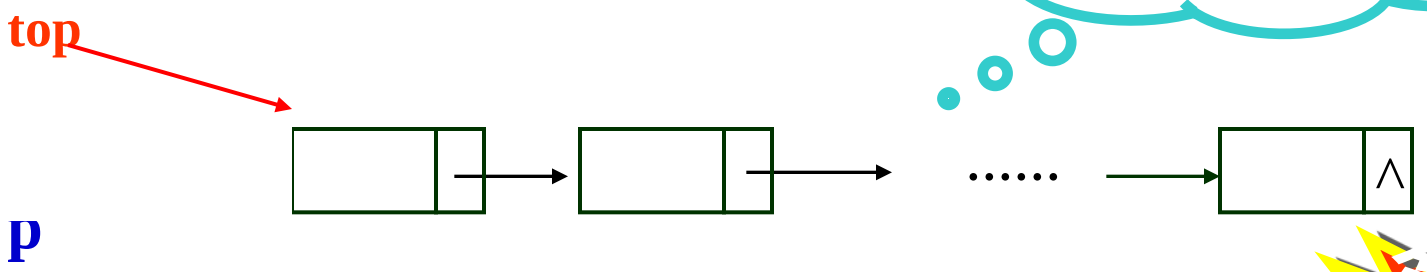
不必判断栈满

算法

```
int  PUSH_LINK( STLink &top, SElemType item )
{    STLink p;
     if( !(p=(STLink)malloc(sizeof(STNode)) )
          return 0;
     else{
          p->data=item;          /*将item送新结点数据域*/
          p->link=top;           /*将新结点插在链表最前面*/
          top=p;                 /*修改栈顶指针的指向*/
          return 1;
     }
}
```

四. 删除(退栈)算法

仍然要判断栈空！



算法

```
int   POP_LINK( STLink &top, SElemType &item )
{    STLink p;
     if ( EMPTYS(top) )
          return 0;                    /* 删除失败*/
     else {
          p=top;                       /* 暂时保存栈顶结点的地址*/
          item=top->data;              /*保存被删栈顶的数据信息*/
          top=top->link;               /* 删除栈顶结点 */
          free(p);                     /* 释放被删除结点*/
          return 1;                    /* 删除成功*/
     }
}
```

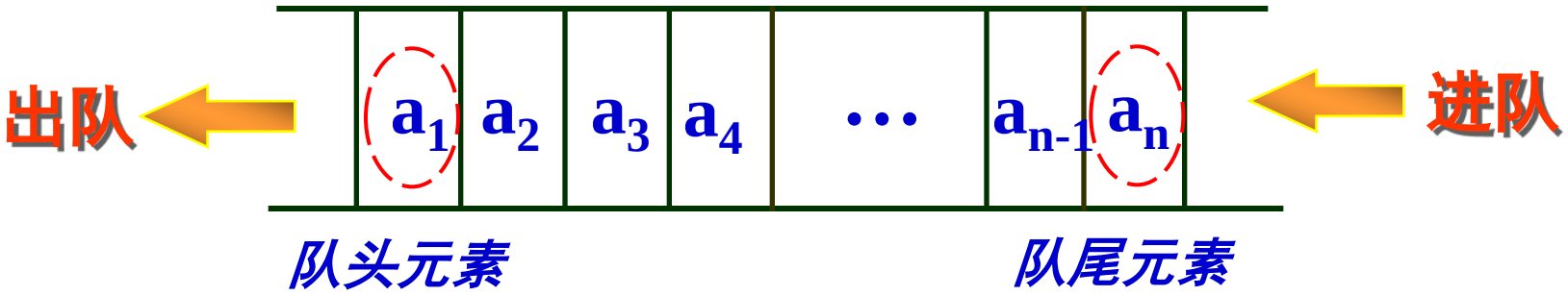
4.4 堆栈的应用举例

4.5 队列的基本概念

一．队列的定义

队列简称**队**。是一种只允许在表的一端进行插入操作，而在表的另一端进行删除操作的线性表。允许插入的一端称为队尾，队尾元素的位置由rear指出； 允许删除的一端称为队头，队头元素的位置由front指出。

先进先出



先进先出

二. 队列的基本操作

1. 队列的插入（进队、入队） √
2. 队列的删除（出队、退队） √
3. 测试队列是否为空(满) √

主要操作

4. 检索当前队头元素
5. 初始化一个队列

特殊性

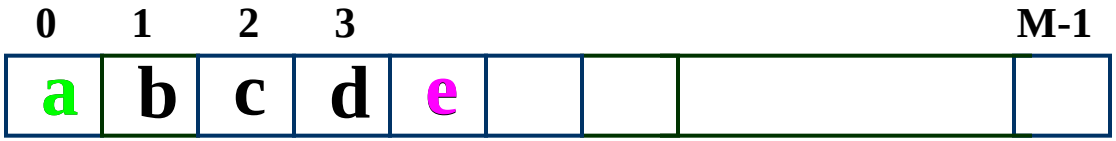
1. 其操作仅是一般线性表的操作的一个子集。
2. 插入和删除操作的位置受到限制。

4.6 队列的顺序存储结构

一. 构造原理

在实际程序设计过程中，通常借助一个一维数组**QUEUE[0..M–1]**来描述队列的顺序存储结构，同时，设置两个变量 **front**与**rear**分别指出当前队头元素与队尾元素的位置。

QUEUE[0.. M-1]



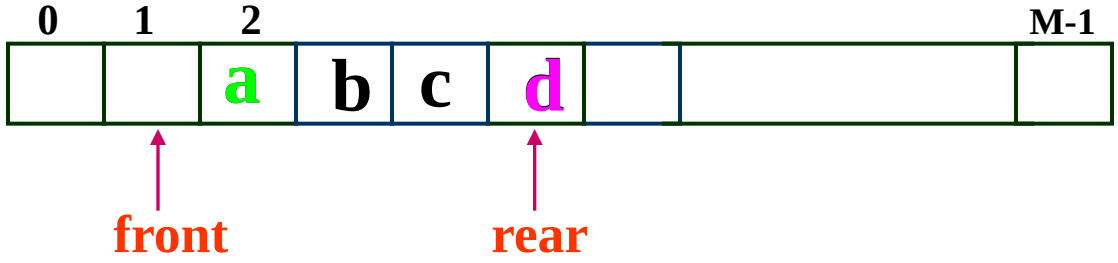
队头元素　　队尾元素

约定

rear 指出实际队尾元素所在的位置，

front 指出实际队头元素所在位置的前一个位置。

QUEUE[0.. M–1]



初始时，队列为空，有

front= –1　　rear= –1

测试队列为空的条件是

front=rear

类型定义

```
#define M     1000
QElemType  QUEUE[M];
int  front, rear;
```

二. 基本算法

一. 初始化队列

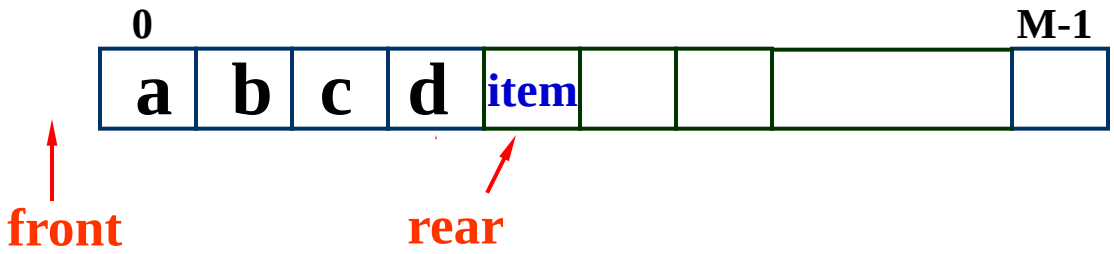
```
void  INITIALQ( int &front, &rear )
{
      front= –1;
      rear= –1;
}
```

二、测试队列是否为空

队空,返回1,否则,返回0。

```
int  EMPTYQ( int front,int rear )
{
      return  front==rear ;
}
```

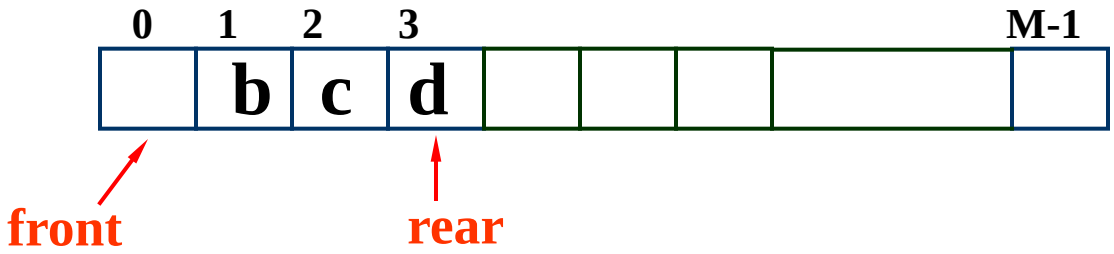
三. 插入(进队)算法



算法

```
int ADDQ( QElemType QUEUE[], int &rear, int item )
{
    if (rear==M-1)                          /* 队列满，插入失败 */
        return  0;
    else {
        QUEUE[++rear]=item;
        return  1;                          /* 队列未满，插入成功 */
    {
}
```

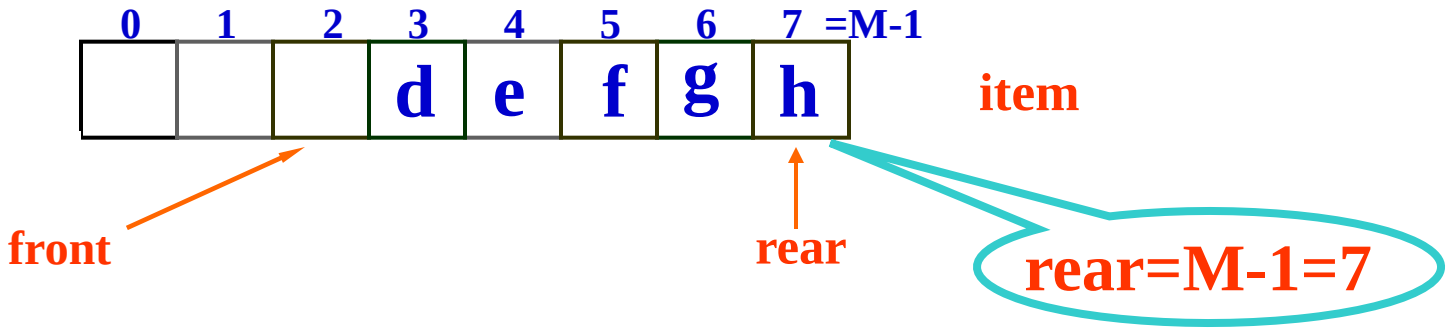
四. 删除(出队)算法



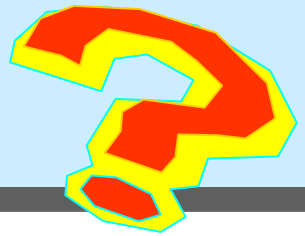
算法

```
int  DELQ( QElemType QUEUE[],
                int &front, int &rear, QElemType &item )
{
      if ( EMPTYQ(front,rear) )
           return 0;                        /* 队列为空，删除失败 */
      else {
           item=QUEUE[++front];
           return 1;                        /* 队列非空，删除成功 */
      }
}
```

假溢出



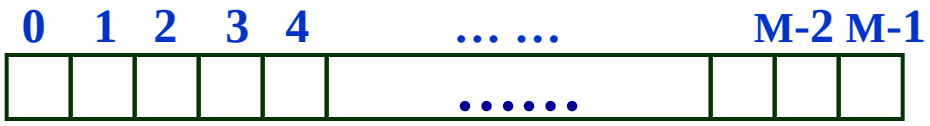

```
int ADDQ( QElemType QUEUE[], int &rear, int & item )
{
    if (rear==M-1)                    /* 队列满，插入失败 */
        return  0;
    else {
        QUEUE[++rear]=item;
        return  1;                    /* 队列未满，插入成功 */
    {
}
```

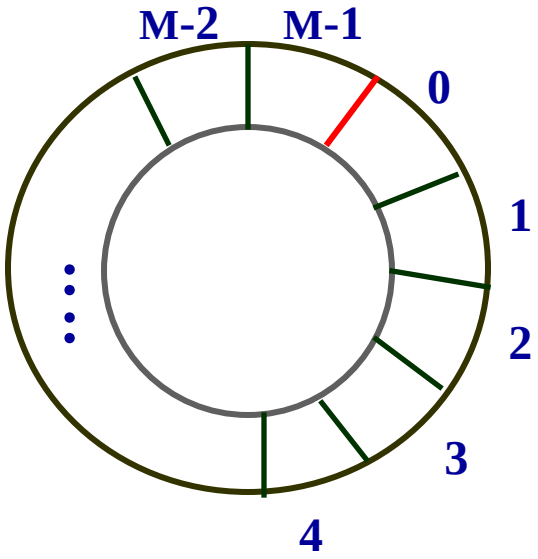


五. 循环队列

把队列(数组)设想成头尾相连的循环表，使得数组前部由于删除操作而导致的无用空间尽可能得到重复利用，这样的队列称为循环队列。

QUEUE[0.. M-1]





4.7 队列的链式存储结构

一. 构造原理

队列的链式存储结构是用一个线性链表表示一个队列，指针**front** 与**rear**分别指向实际队头元素与实际队尾元素所在的链结点。

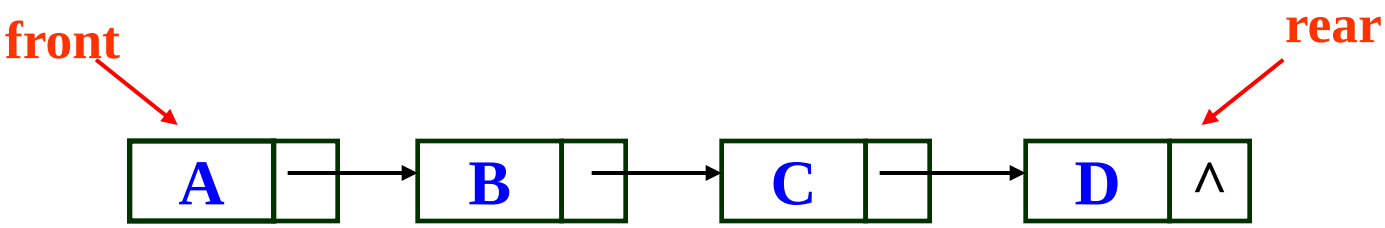
约定

rear 指出实际队尾元素所在的位置，
front 指出实际队头元素所在位置的前一个位置。

在一个初始为空的链接队列中依次插入数据元素

A，B，C，D

以后，队列的状态为



空队对应的链表为空链表，空队的标志是

front = NULL

类型定义

```
typedef  struct  node {
        QElmeType   data;
        struct  node   *link;
} QNode, *QLinkList;
```

二. 基本算法

一. 初始化队列

```
void  INITIALQ( Qlink  & front, Qlink & rear )
{
      front=NULL;
      rear=NULL;
}
```

二. 测试队列是否为空

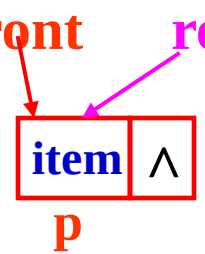
队空,返回1,否则,返回0。

```
int  EMPTYQ( Qlink front )
{
      return   front==NULL  ;
}
```

三. 插入算法

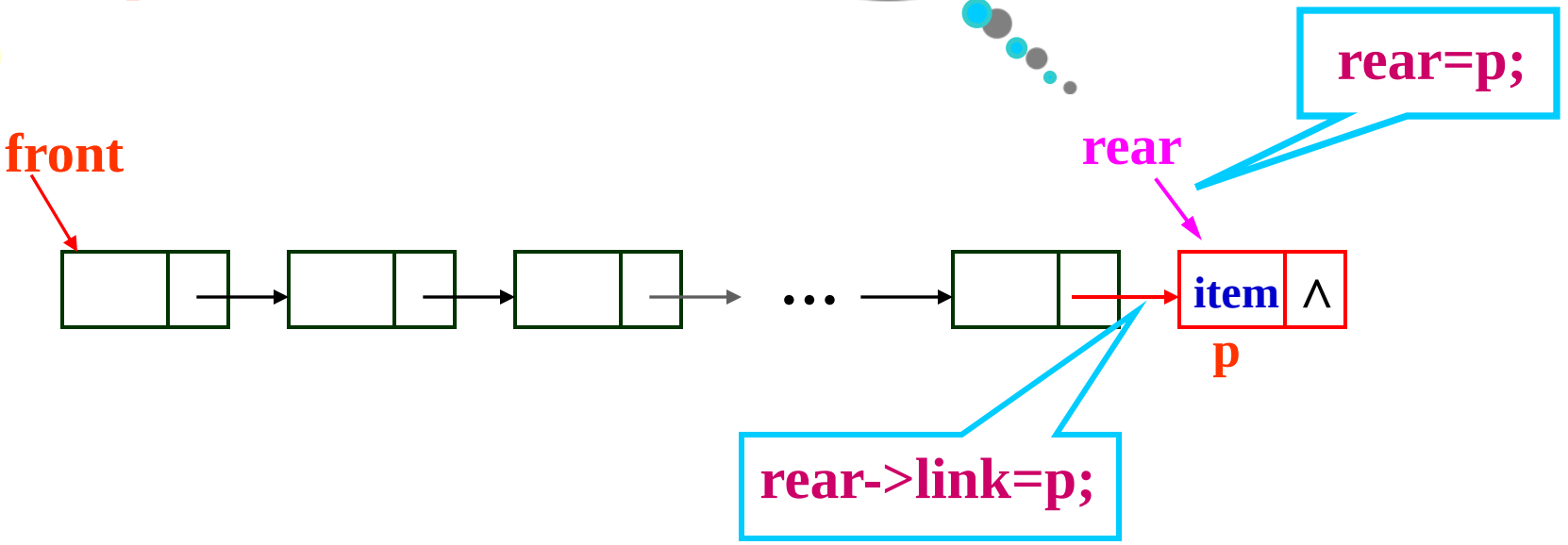
分两种情况

1 front=rear=NULL



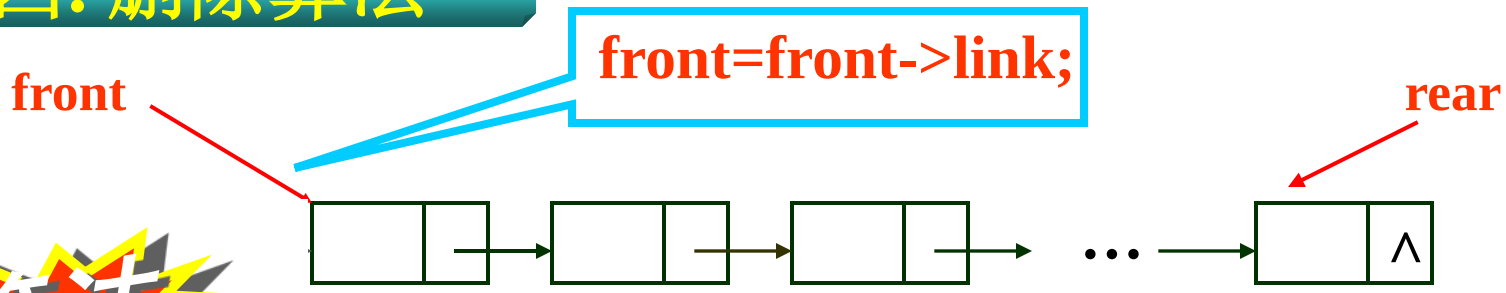
为什么？

2




```
int  ADDLINKQ( QLink  &front, QLink  &rear,
                        QElemType  item )
{       QLink  p;
          if(!(p=(Qlink)malloc(sizeof(QNode)))   /* 申请链结点 */
              return  0;
           p->data=item;
           p->link=NULL;
           if (front ==NULL)
                 front=p;                          /* 插入空队的情况 */
           else
                 rear->link=p;                     /* 插入非空队的情况 */
           rear=p;
           return 1;
}
```

四. 删除算法

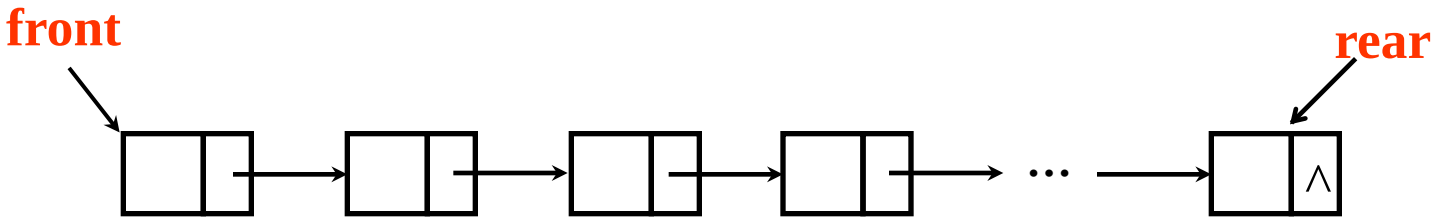


算法

```
int  DEL_LINKQ( Qlink &front,  Qlink &rear,
                QElemType  &item )
{    Qlink  p;
     if ( EMPTYQ(front) )
          return 0;                /* 队列为空，删除失败 */
     else {
          p=front;
          front=front->link;
          item=p->data;
          free(p);
          return  1;               /* 队列非空，删除成功 */
     }
}
```

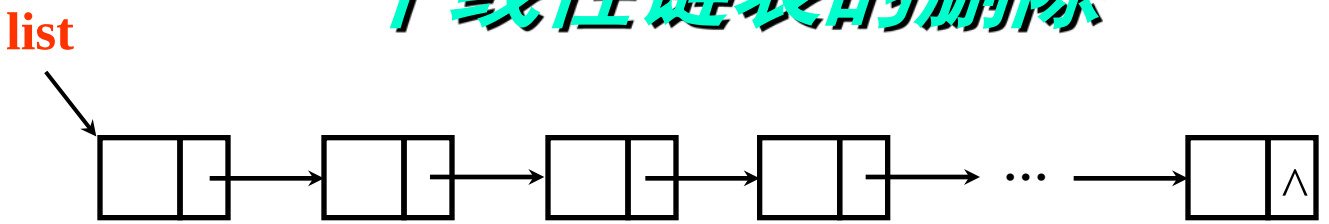
五.销毁一个队列

所谓销毁一个队列是指将队列所对应的链表中所有结点都删除，并且释放其存储空间，使队列成为一个空队(空链表)。



归结为一个线性链表的删除

一个线性链表的删除



list=list->link;

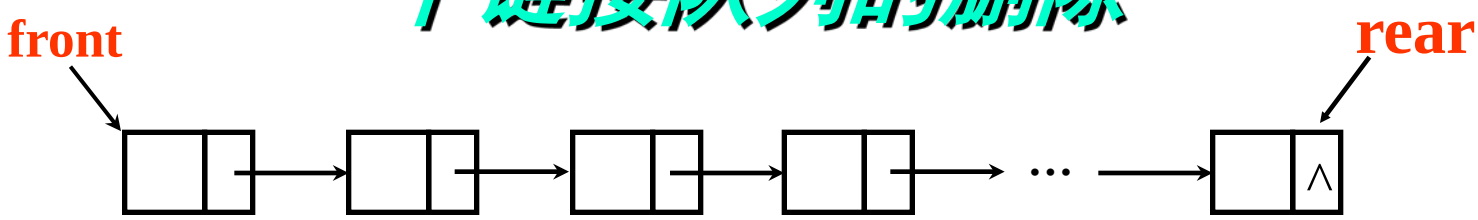
list=NULL

list

p

p=list;

一个链接队列的删除



rear=front->link;

rear=NULL

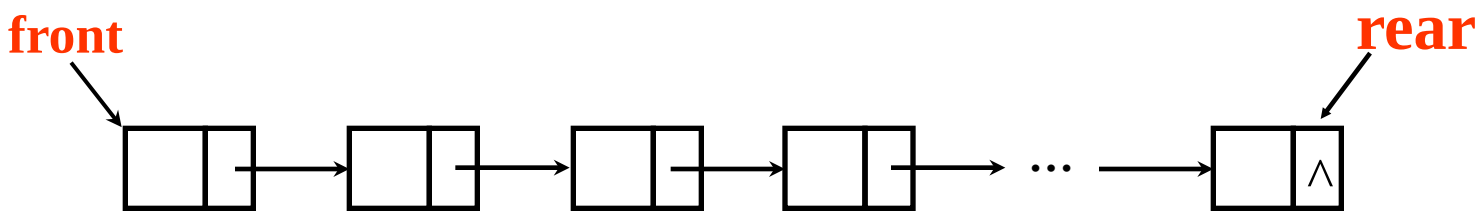
rear

front

front=rear;

算法

```
void DESLINKQ(QLink &front,  QLink &rear)
{
      while(front){                    /* 队列非空时 */
          rear=front->link;
          free(front);                 /* 释放一个结点空间 */
          front=rear;
      }
}
```



本章内容小结

堆栈、队列

堆栈、队列的基本概念

- 堆栈、队列的定义
- 堆栈、队列的基本操作

堆栈、队列是特殊线性表（特殊性）

操作的时间都为O(1)

堆栈、队列的顺序存储方法

- 构造原理、特点
- 对应的插入、删除操作的算法设计（循环队列）

堆栈、队列的链式存储结构

- 构造原理、特点
- 对应的插入、删除操作的算法设计

应用举例（见习题课）