第二章

基本数据结构及其运算

2.1 数据结构的基本概念

数据结构 主要讨论3个方面问题：

- 1）数据集合中各数据元素之间所固有的逻辑关系，即数据的逻辑结构；
- 2）各数据元素在计算机中的存储关系，即数据的存储结构；
- 3）对各数据结构进行的运算；

2.1.1 例子

一.基本概念

- 根据姓名查找

姓名
周清
储旭
朱蔡敏
科尔沁夫
张永坦
陈彪
许拓夫
唐波
左骥
彭文耀
许炎杰
杨杞
谢嘉彬
吉喆
李毅舟
王子汉
顾震曦
李威
彭澎
陆昱州
蒋泽浩
李定一
王侃

2.1.1 例子

一.基本概念

- 查找姓名

1	0704210256	周清	07042102
2	0704220222	储旭	07042202
3	0704480135	朱蔡敏	07044801
4	0804210228	科尔沁夫	08042102
5	0804220156	张永坦	08042201
6	0804220220	陈彪	08042202
7	0804330133	许拓夫	08043301
8	0804620121	唐波	08046201
9	0804620129	左骥	08046201
10	0808190236	彭文耀	08081902
11	0810190141	许炎杰	08101901
12	0710200144	杨杞	08102001
13	0810200237	谢嘉彬	08102002
14	0810200322	吉喆	08102003
15	0810200327	李毅舟	08102003
16	0910190243	王子汉	09101902
17	0910200324	顾震曦	09102003
18	1010190228	李威	10101902
19	1010190237	彭澎	10101902

根据具体问题将数据组织成
不同的形式，可以提高处理效率

2.1.2 什么是数据结构

一.基本概念

- **数据结构（Data Structure）**

是互相有关联的数据元素的集合

例子：向量，矩阵；

{春，夏，秋，冬}：季节，有固定的前后件关系；

具有相同特征的数据元素集合中，各数据元素之间存在某种关系，这种关系反映了该集合中数据元素所固有的一种结构。通常简单的用前后件关系来描述。

更通俗：

带有结构的 数据元素的集合。结构：数据元素之间的前后件关系。

二. 数据的逻辑结构

特点：描述数据间的顺序（逻辑）关系，抽象地反映数据元素的结构，而不管它们在计算机中如何存放。

两要素：数据元素的集合D，数据元素的前后件关系R

描述方法：用二元组来描述：

B=（D，R）

其中：

B：数据结构；

D：是数据元素的有限集合；

R：是数据元素之间关系的集合。

例子1

- B=(D,R)
- D={春，夏，秋，冬}
- R={（春，夏），（夏，秋），（秋，冬）}

例子2

- **B=(D,R)**
- **D={父亲，儿子，女儿}**
- **R={（父亲，儿子），（父亲，女儿}**

例子3（复杂些）

成员：由1名教师、1~3名研究生、1~6名本科生组成；

- 成员关系是：教师指导研究生、研究生指导1~2名本科生。
- 数据结构的形式化描述：定义如下：

$$B=（D，R）$$

其中：　$D=\{T，G_1，\ldots,G_n,S_{11},\ldots S_{nm}\}$　$1 \le n \le 3$，$1 \le m \le 2$　　$R=\{R_1,R_2\}$

$R_1=\{<T,G_i>|1 \le i \le n , 1 \le n \le 3\}$

$R_2=\{<G_i,S_{ij}>|1 \le i \le n ,1 \le j \le m , 1 \le n \le 3 , 1 \le m \le 2 \}$

三、数据的存储结构

~是指数据结构在计算机中的表示(又称映象)，即数据在计算机中的存放形式。

是逻辑结构在存储器中的映射，又称物理结构

常用数据存储结构

1. 顺序存储结构
2. 链式存储结构
3. 索引存储结构

1. 顺序存储结构

概念：把数据元素按某种顺序存放在一块连续的存储单元中的存储形式。

数据结点结构：

特点：

① 连续存放;逻辑上相邻,物理上也相邻。

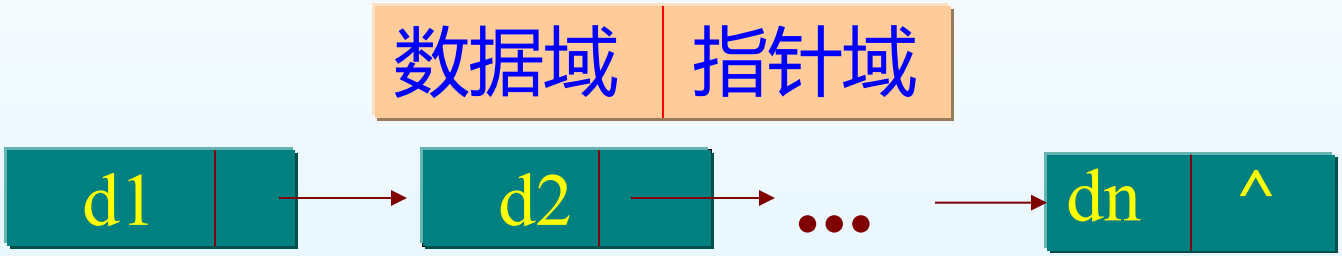
② 结构简单，易实现。

③ 插入、删除操作不便（需大量移动元素）。

2. 链式存储结构

概念： 以链表形式将数据元素存放于任意存储单元中，可连续存放，也可以不连续存放，以指针实现链表间的联系。

数据结点结构：



特点:

① 非连续存放,借助指针来表示元素间的关系;

② 插入、删除操作简单，只要修改指针即可；

③ 结构较复杂，需要额外存储空间。

总结：

（1）逻辑结构和物理结构的关系

① 数据的逻辑结构是从逻辑关系（某种顺序）上观察数据，它是独立于计算机的；可以在理论上、形式上进行研究、推理、运算等各种操作。

② 数据的存储结构是逻辑结构在计算机中的实现，是依赖于计算机的；离开了机器，则无法进行任何操作。

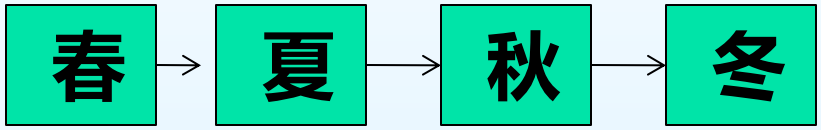
③ 任何一个算法的设计取决于选定的逻辑结构；而算法的最终实现依赖于采用的存储结构。

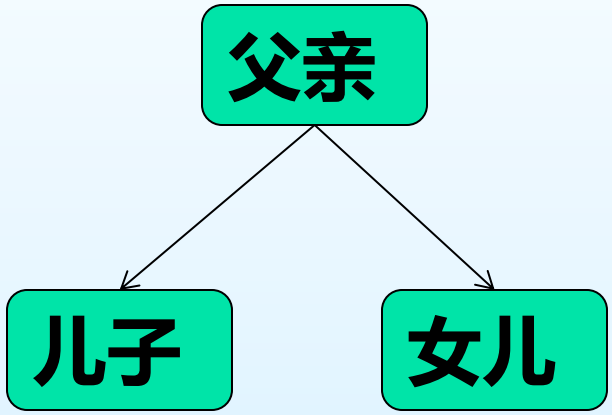
逻辑结构：独立于计算机。存储结构：依赖于计算机

算法设计考虑逻辑结构，算法实现依赖存储结构。

2.1.3 数据结构的图形表示

- 数据结构除了用二元关系表示外，还可以直观的用图形表示：





2.1.3 数据结构的图形表示

- 根结点：没有前件的结点；

  春，父亲

- 终端结点（叶子结点）：没有后件的结点。

  冬，儿子，女儿

2.1.3 数据结构的图形表示

数据结构可能是动态变化的：如

- 结点的插入运算；
- 结点的删除运算；

空数据结构：

一个数据结构中一个数据元素都没有。

2.1.3 线性结构

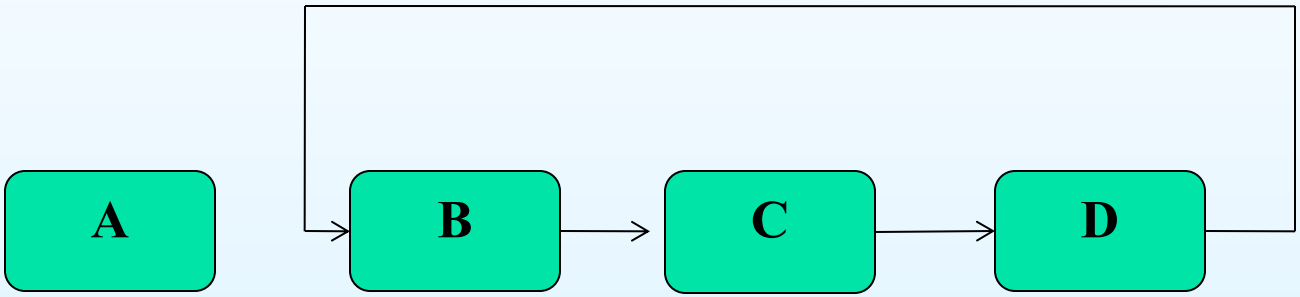
线性结构：

非空的数据结构满足下面两个条件：

- 有且只有一个根节点；
- 每一个结点最多有一个前件，也最多有一个后件。
- 插入或删除任一个结点后，前两个条件仍成立。

Note

一个线性结构中插入和删除任何一个结点后还是线性结构。



满足线性结构两个条件，但是不是线性结构

空数据结构

- 线性结构和非线性结构都可以是空的数据结构。空的数据结构属于线性还是非线性要按照具体情况确定。
- 如果对该数据结构的运算是按线性规则来处理的，则属于线性结构，否则属于非线性结构。

2.2 线性表及其顺序存储结构

2.2.1 线性表及其运算

- 线性表是指数据元素之间的关系为一一对应的线性关系的数据结构。

  例如，一星期七天的英文缩写表示：

  （Sun，Mon，Tue，Wed，Thu，Fri，Sat）

是一个线性表，其中的元素是字符串，表的长度为7。

- 线性表虽然简单，但是应用范围非常广泛。

1. 什么是线性表

定义： 线性表是n（n≥0）个元素$a_1,a_2,\ldots,a_n$ 的有限序列；表中每个数据元素，除第一个外，有且只有一个前件；除最后一个外，有且只有一个后件。即线性表或是一个空表，或可以表示为

$$(a_1,a_2,\ldots,a_i,\ldots a_n)$$

例如：一星期七天的英文缩写表示：

（Sun，Mon，Tue，Wed，Thu，Fri，Sat）

是一个线性表，其中的元素是字符串，表的长度为7。

2. 线性表的顺序存储结构

- 线性表的顺序存储结构

  将表中元素一个接一个的存入一组连续的存储单元中，这种存储结构是顺序结构。

- 顺序表

  采用顺序存储结构的线性表简称为“顺序表”。

元素序号	内存状态	存储地址
1	a_1	$LOC(a_1)$
2	….	$LOC(a_1)+1L$
….	a_2	….
i	….	$LOC(a_1)+(i-1)L$
….	a_i	….

2. 特点

1） 所有元素所占的存储空间是连续的；

2）各数据元素在存储空间中是按照逻辑顺序依次存放的；

- 存储位置计算：
- 只要确定了起始位置，表中任一元素的地址都通过下列公式得到：

$$LOC(a_i)=LOC(a_1)+(i-1)*L \quad 1\le i \le n$$

其中，L是每个元素占用存储单元的长度。

程序代码

- 通常可以用一维数组来表示线性表
- 开辟存储空间时要考虑留出空余的空间，便于后续的插入等运算；

对C语言，顺序表可定义如下：[按需修改]

```
#define MaxLength 50
typedef int ElemType;
 typedef struct
 {
   ElemType list[MaxLength];
  int length;
}SeqList;
```

今后使用此定义时，MaxLength及ElemType要根据实际问题的需要可重新选定。

顺序表的运算

顺序表的插入

顺序表的删除

顺序表的初始化

顺序表的长度

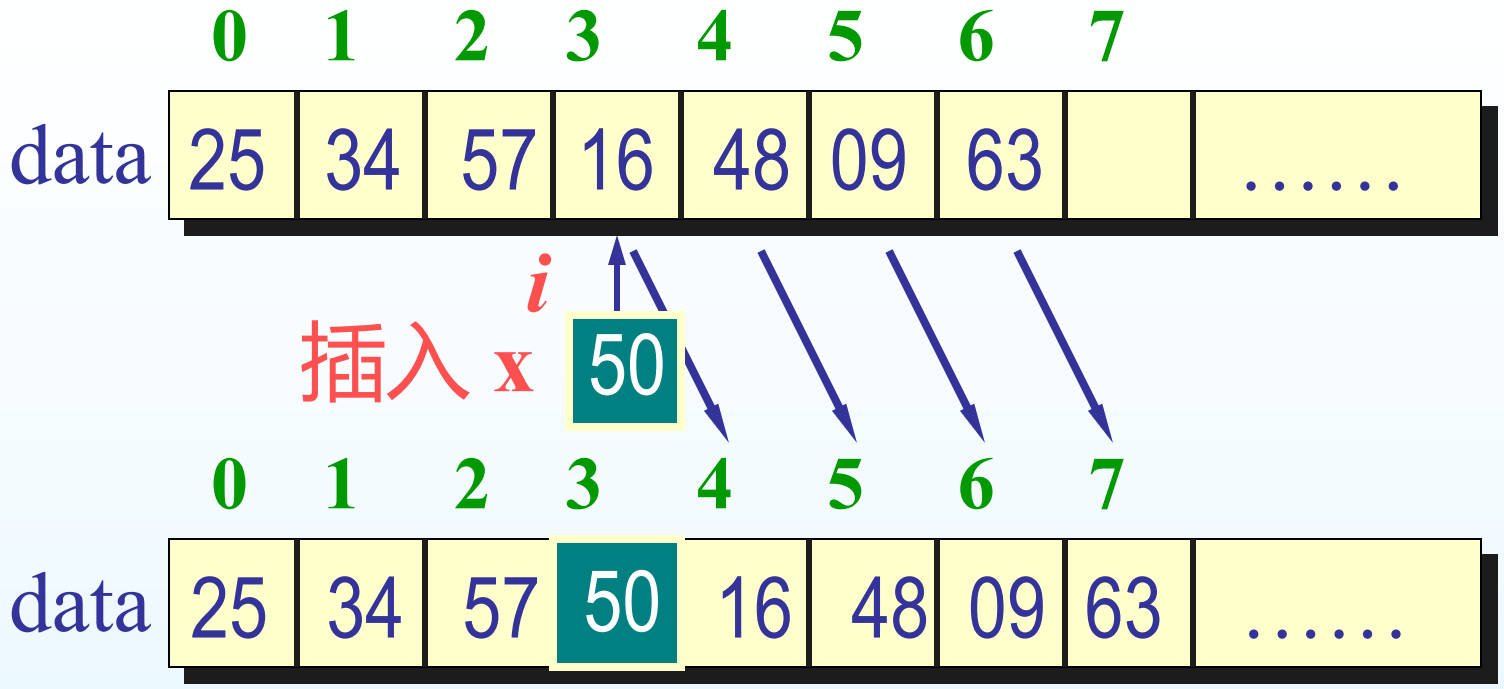
顺序表的取第i个元素

顺序表的查找(定位运算)

1.顺序表的插入

设长度为 n 的顺序表为（$a_1,a_2,\ldots,a_i,\ldots,a_n$），要在顺序表的第i(1≤i≤n)个元素$a_i$之前插入一个新 元素x，插入后得到长度为 n+1的线性表($a_1,a_2,\ldots,a_{i-1},x,\ a_i,\ldots,a_n$)，即($a_1,a_2,\ldots,a_{i-1},\ a'_i,\ a'_{i+1},\ldots,a'_{n+1}$)，其中$a'_i$为新插入的元素x，$a'_{i+1}$ 为原表中的a_i ，其余类推，a'_{n+1}为原表中a_n 。

表项的插入


0 1 2 3 4 5 6 7
data
25 34 57 16 48 09 63 .....
i
插入 x
50
0 1 2 3 4 5 6 7
data
25 34 57 50 16 48 09 63 .....

在平均情况下，插入一个新元素，需要移动表中一半的元素。

注意：若最后一个元素之后没有多余的自由空间（即表的大小**n=MaxLength**）时，那么插入一个元素，将会发生上溢。

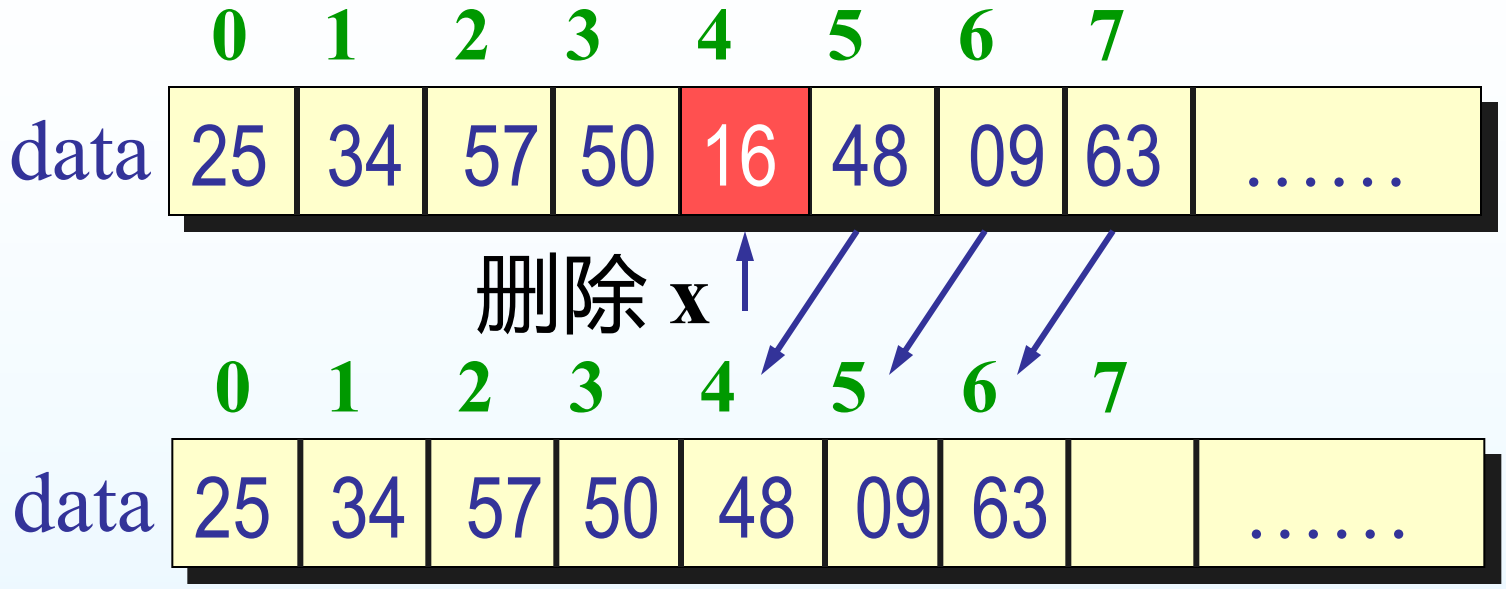
```
void InsertList(SeqList *L,int i,
ElemType x)
{    int j,n=L->length;
     if(i<0||i>n+1)
     { printf("\n i值不合法");
        exit(1);
     }
     if(n>=MaxLength)
     {  printf("\n 表空间上溢");
        exit(1);
     }
```

```
for(j=n-1;j>=i-1;j--)
 L->list[j+1]=L->list[j];/*数据元素依次向后移动一个位置*/
 L->list[i-1]=x; /*插入x */
 L->length++;  /*表长增加1*/
}
```

2.顺序表的删除

通常，在长度为 n 的顺序表中，要删除线性表的第i(1≤i≤n)个元素a_i 。得到长度为n-1的线性表($a_1,a_2,\ldots,a_{i-1},a_{i+1},\ldots,a_n$) 。

表项的删除


0 1 2 3 4 5 6 7
data
25 34 57 50 16 48 09 63 .....
删除 x
0 1 2 3 4 5 6 7
data
25 34 57 50 48 09 63 .....

即 $(a_1, a_2,\ldots,a_{i-1},a'_i,a'_{i+1},\ldots,a'_{n-1})$，其中$a'_i$为原表中的$a_{i+1}$ ，其余类推，a'_{n-1}为原表中a_n 。

在平均情况下，要在顺序表中删除一个元素，需要移动表中一半的元素。

```
void DeleteList(SeqList *L,int i,
ElemType *x)
{
    int j,n=L->length;
    if(i<1||i>n)
    {
       printf(" \n  i值不合法!");
       exit(1);
    }
```

```
*x=L->list[i-1];
        /*将被删元素的值，赋给*x */
    for(j=i;j<=n-1;j++)
        L->list[j-1]=L->list[j];
        /*元素依次向前移动一个位置 */
    L->length--; /*表长减少1 */
}
```

3.顺序表的初始化

构造一个空的顺序表L（即表的初始化）的算法如下：

```
void InitList(SeqList *L)
{       /*构造一个空的顺序表L */
  L->length=0;/*线性表长度赋0值*/
}
```

4.顺序表的长度

确定顺序表L的长度算法如下：

```
int ListLength(SeqList *L)
{       /*求线性表L的长度*/
return(L->length);/*返回L的长度*/
}
```

5. 取顺序表的第i个元素

从顺序表中取第i（1≤i≤n）个数据元素的算法如下：(用于在表中随机的访问任意一个结点)

```
ElemType GetElem(SeqList *L,int i)
{/*取表中第i个数据元素*/
    if(i<1||i>L->length)
    {  printf("\n i值非法!");
       exit(1);
    }
    return(L->list[i-1]);
 }
```

6. 顺序表的定位运算

根据数据项的值x，对顺序表L进行查找，若L中有元素的值与x相同，则返回首次找到的元素在L中的位置；若查找失败，则返回-1的算法如下：

```
int LocateElem( SeqList *L,
 ElemType x)
{/*查找与x相匹配的元素并返回其位置*/
    int i=0,n=L->lenth;
    if(n==0)
```

```
    {
      printf("\n Empty List !");
       exit(1); /*若空表，则返回*/
    }
    while(i<n&&L->list[i-1]!=x)
       i++;
    if(i<n)
      return(i+1);/*找到返回其位置*/
    else
      return(-1);/*查找失败返回-1*/
}
```

插入运算时间复杂度:

假设在长度为n的顺序表的任意位置i（1≤i≤n）插入一个元素的概率为$p_i=1/(n+1)$，所需移动元素的次数为n-i+1，那么每插入一个元素，所需移动元素的次数的平均值为：$A_{is}=n/2$

在表中插入一个元素，平均要移动一半的元素，平均时间复杂度为O(n)。

最好的情况是在表尾插入时，不需要移动元素；最坏的情况是在表头插入时，需要移动表中n个元素。

删除运算时间复杂度:

假设，在长度为n的顺序表的任意位置i（1≤i≤n）删除该位置元素的概率为$q_i=1/n$，所需移动元素的次数为n-i，那么，每删除一个元素，所需移动元素的次数的平均值为：

$A_{de}=(n-1)/2$

在顺序表中删除一个元素，平均约移动表中一半的元素。平均时间复杂度为O(n)。最好的情况是当i=n，即在表尾删除时，不需要移动元素；最坏的情况是当i=1，即在表头删除时，需要移动表中n-1个元素。

顺序表使用例子

```
SeqList myList;//定义
InitList(&myList);//初始化
InsertList(&myList,1,10);
InsertList(&myList,2,20);
InsertList(&myList,3,30);
 InsertList(&myList,1,40);
```

```
int i;
for (i=0;i<4;i++)
{printf("%d\n",GetElem(&myList,i+1));
}
```

课堂练习

(1)建立空顺序表，并初始化；

(2)按顺序插入元素其值分别为1-30；

(3)将其中能被3整除的元素删除；

(4)打印输出所有元素。

总结：
顺序存储结构的优缺点

- 数据连续存放、随机存取
- 逻辑上相邻，物理上也相邻
- 存储结构简单、易实现
- 插入、删除操作不便
- 存储密度大，空间利用率高

结论：

顺序存储结构适合于表中元素变动较少的情况。

2.2.2 栈及其应用

1 什么是栈

实际上也是线性表 只不过是一种特殊的线性表

现实中的例子

- 物料仓库中的储存
- 机器零部件的装配与拆卸

- 堆栈（英文：stack），也可直接称栈。是一种特殊的数据结构，它的特殊之处在于只能允许在链结串行或阵列的一端（称为堆栈顶端指标，英文为top）进行加入资料（push）和输出资料（pop）的运算。
- 由于堆栈数据结构只允许在一端进行操作，因而按照后进先出（LIFO, Last In First Out）的原理运作。

堆栈数据结构使用两种基本操作：

推入（push）　　：将数据放入堆栈的顶端，堆栈顶端top指标加一。

弹出（pop）　　：将顶端数据资料输出，堆栈顶端资料减一。

堆栈(Stack)

- 栈是允许在同一端进行插入和删除操作的特殊线性表。
- 允许进行插入和删除操作的一端称为*栈顶*(top)，另一端为*栈底*(bottom)；栈底固定，而栈顶浮动；
- 栈中元素个数为零时称为*空栈*。
- 栈结构也称为后进先出表（LIFO）。

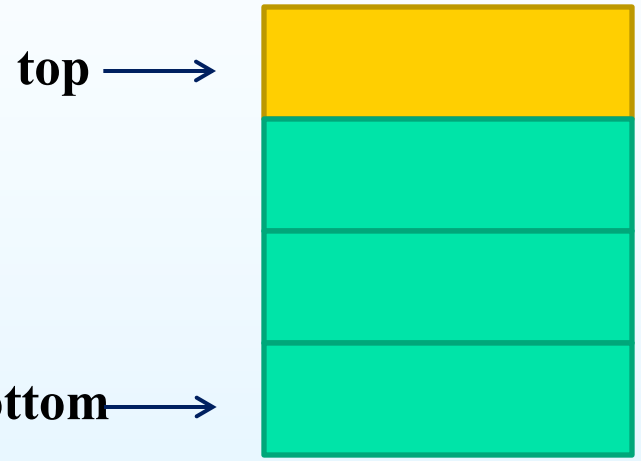
栈、栈顶、栈底、空栈

后进先出表　栈底固定，而栈顶浮动

栈有关概念

栈顶指针

在栈操作过程中，有一个专门的栈指针(习惯上称它为TOP)，指出栈顶元素所在的位置。



栈空的条件：top = 0

栈满的条件：top = MAXSIZE

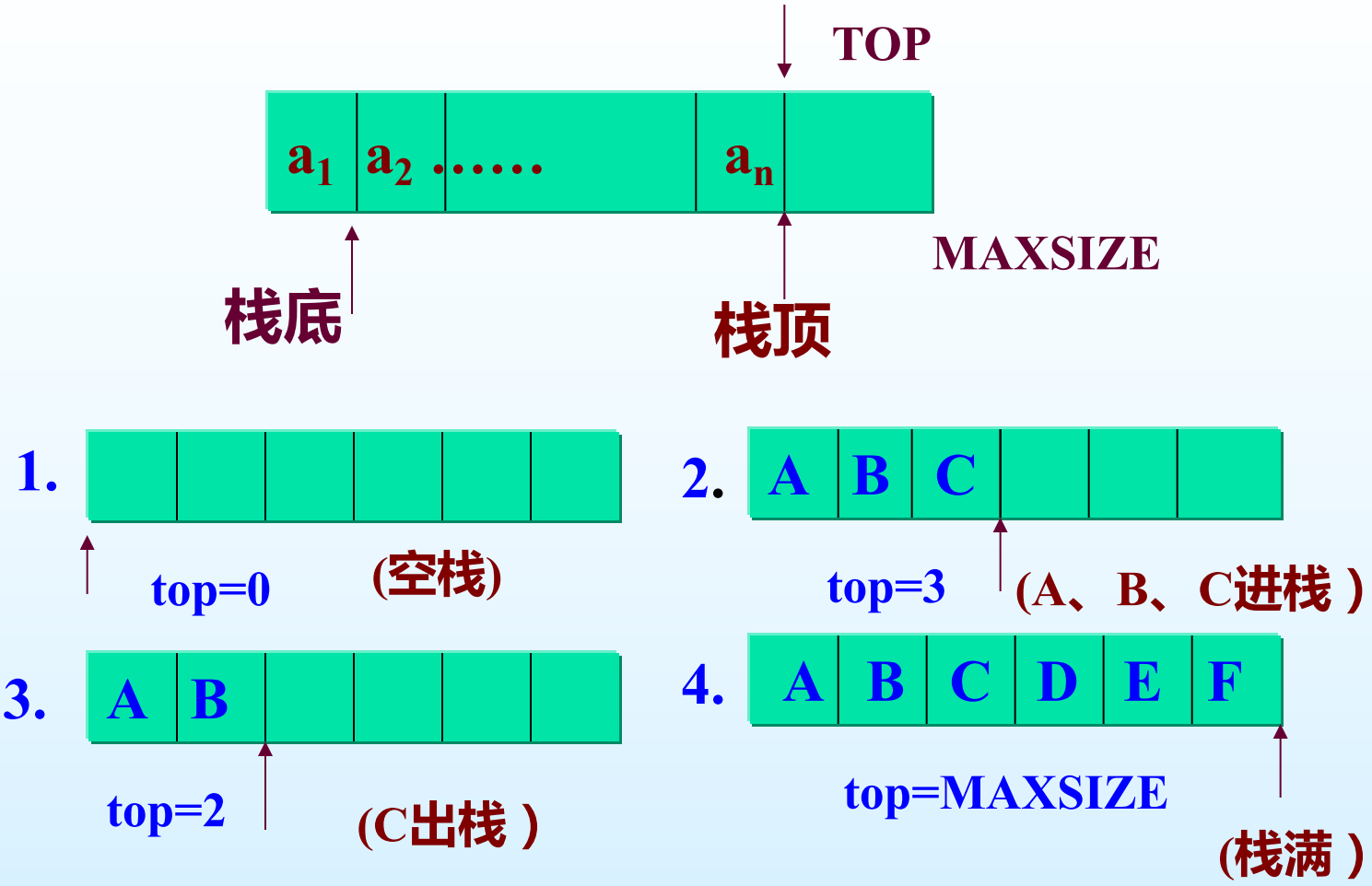
栈上溢

栈空间是有限的，若栈已满，再进行入栈操作时，就要产生上溢

栈下溢

若栈空，再要执行出栈操作，则会发生下溢。

栈操作举例


TOP
a_1 a_2 ..... a_n
MAXSIZE
栈底
栈顶
1.
top=0
(空栈)
2.
A B C
top=3
(A、B、C进栈)
3.
A B
top=2
(C出栈)
4.
A B C D E F
top=MAXSIZE
(栈满)

2 栈的顺序存储及其运算

（1）栈的顺序存储结构：用一维数组作为存储空间。

（2）顺序栈：栈的顺序存储结构称为顺序栈。

栈的操作只能在一端进行；即栈顶位置随进栈和出栈而变化。

```
#define MaxSize 100
 /*设预分配的栈空间最多为100个元素*/
typedef char ElemType;
 /*设栈元素的数据类型为字符型*/
typedef struct
{
 ElemType stack[MaxSize];
 int top;/*top指示栈顶元素的位置*/
}SeqStack;
```

设s是SeqStack类型的指针变量，在顺序栈中，s->top==0表示空栈；

当s->top已经指向了MaxSize-1表示栈满。向一个满栈插入元素和从一个空栈删除元素会产生**上溢**或**下溢**。上溢是一种出错状态应该避免，下溢则常用来作为程序控制转移的条件。

栈的**五种基本运算**是：构造一个空栈、判断栈是否为空、进栈、退栈与读栈顶元素。下面介绍相应的算法。

1）构造空栈

构造空栈是指栈的初始化即给s->top赋值0，算法如下：

```
void InitStack(SeqStack *s)
{
  s->top=0;
}
```

2）判断栈是否为空

判断栈是否为空是指判断s->top是否等于0，若是，则表示栈空返回1；否则返回0，算法如下：

```
int StackEmpty(SeqStack *s)
                    /*int可省略不要*/
{
   return s->top==0;
}
```

3) 进栈

- 算法步骤：
  - **step1**

    判别栈满否，若满，则显示栈溢出信息，停止执行；否则，执行step 2；
  - **Step2**

    栈顶指针top上移(加1)；
  - **Step3**

    在top所指的位置插入元素x。

```
void push(SeqStack *s,ElemType x)
{     /* 插入一个值为x的新元素*/
   if(s->top==MaxSize-1)
   {
      printf("\n 栈已满，上溢！");
      exit(1);
   }
      s->top++; /*栈顶指针加1*/
   s->stack[s->top]=x;   /*值为x的新元素进栈*/
}
```

4) 出栈算法

- 算法步骤:
  - **step1** 判别栈是否为空；若空，则输出栈下溢信息，并停止执行；否则，执行**step2**；
  - **step2** 弹出（删除）栈顶元素；
  - **step3** 栈顶指针**top**下移**(**减**1)**。
  - **step4** 返回出栈元素

```
ElemType pop(SeqStack *s)
{
 if(StackEmpty(s))
   { /*调用判断空栈函数若栈空则下溢！*/
      printf("\n 栈已空，下溢！");
      exit(1);
   }
   return s->stack[s->top--];
      /*返回栈顶元素，栈顶指针减1*/
 }
```

5）读栈顶元素

读栈顶元素是指将栈顶元素赋给一个指定的变量。必须注意，这个运算不删除栈顶元素只是将它的值赋给一个变量，因此在这个运算中栈顶指针不会改变。但当栈为空时，就读不到栈顶元素了，算法如下：

```
ElemType GetTop(SeqStack *s)
{
 if(StackEmpty(s))
   {
     printf("\n 栈已空！");
     exit(1);
   }
      return s->stack[s->top];
 /*若栈非空，则返回栈顶元素，但不删除*/
}
```

顺序栈的5个基本操作

```
void InitStack(SeqStack *s)
int StackEmpty(SeqStack *s)
void push(SeqStack *s,ElemType x)
ElemType pop(SeqStack *s)
ElemType GetTop(SeqStack *s)
```

课堂练习

- 建立空顺序栈并初始化；
- 入栈1,2,3,4,5; 出栈两次并输出出栈的元素；最后输出栈中所有的元素。

表达式计算

计算表达式，首先要正确地定义运算规则：

四则运算
- 先乘除、后加减
- 从左到右
- 先括号内，再括号外

- 为了让计算机能识别表达式，规定：
  - 表达式由操作数（Operand）和操作符（Operator）和结束符 ;组成。

    例如，3+2*7-5；=12

  - 实际处理表达式是用两个栈结构OPTR（运算符）和OPND（操作数）加运算规则组成；

计算表达式算法步骤

- **Step1 初始化，清空OPTR和OPND，将左定界符压OPTR栈；**
- **Step2 循环输入表达式中的每个字符**
  - **若输入操作数，则进OPND栈**
  - **若是操作符，则和OPTR栈顶元素比较，按规则进行相应操作**
  - **操作服从优先关系表（参考P38）**
    - **Q1<Q2　Q2入OPTR栈，再读入下一个元素**
    - **Q1>=Q2　Q1出OPTR栈，从OPND中取两个数运算**
- **Step3 直到出现右定界符为止。**
- **举例，计算： 3+2*7-5**

  输入： 3+2*7-5 ;

步骤	OPTR栈	OPND栈	输入字符	主要操作
1	;		3	PUSH（OPND，3）
2	;	3	+	PUSH（OPTR，‘+’）
3	;+	3	2	PUSH（OPTD，‘2’）
4	;+	3,2	*	PUSH（OPNR，*）
5	;+*	3,2	7	PUSH（OPTD，‘7’）
6	;+*	3,2,7	-	Operate（2，‘*’，7）
7	;+	3, 14	-	Operate（3，‘+’，14）
8	;	17	-	PUSH（OPNR，-）
9	;-	17	5	PUSH（OPTD，‘5’）
10	;-	17,5	;	Operate（17，‘-’，5）
11		12	;	RETURN（GETTOP（OPND））

课堂练习

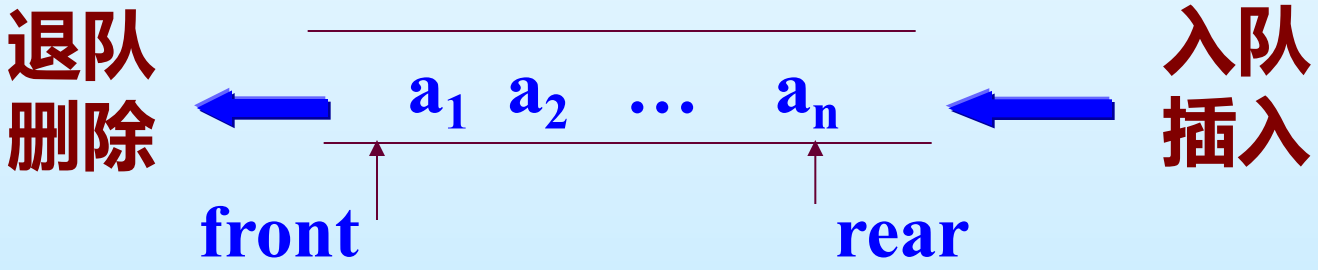
- 参考P40
- 表达式
- **1+9*2/3-13**
- 写出表达式每一个步骤的状态。

- 带括号的复杂些, 参考教材 **P40**

2.2.3 队列及其应用

什么是队列

- 队列是一种特殊的线性表，它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作。
- 进行插入操作的端称为队尾，进行删除操作的端称为队头。
- 队列中没有元素时，称为空队列。
- 队列具有先进先出（FIFO）的特点。



- front为队头指针，指示队头元素的前一个位置。
- rear 为队尾指针，指示队尾元素的位置。

队列的操作

清空队列

判别队列是否为空；空，取T;

非空，取值为F。

插入操作

删除操作

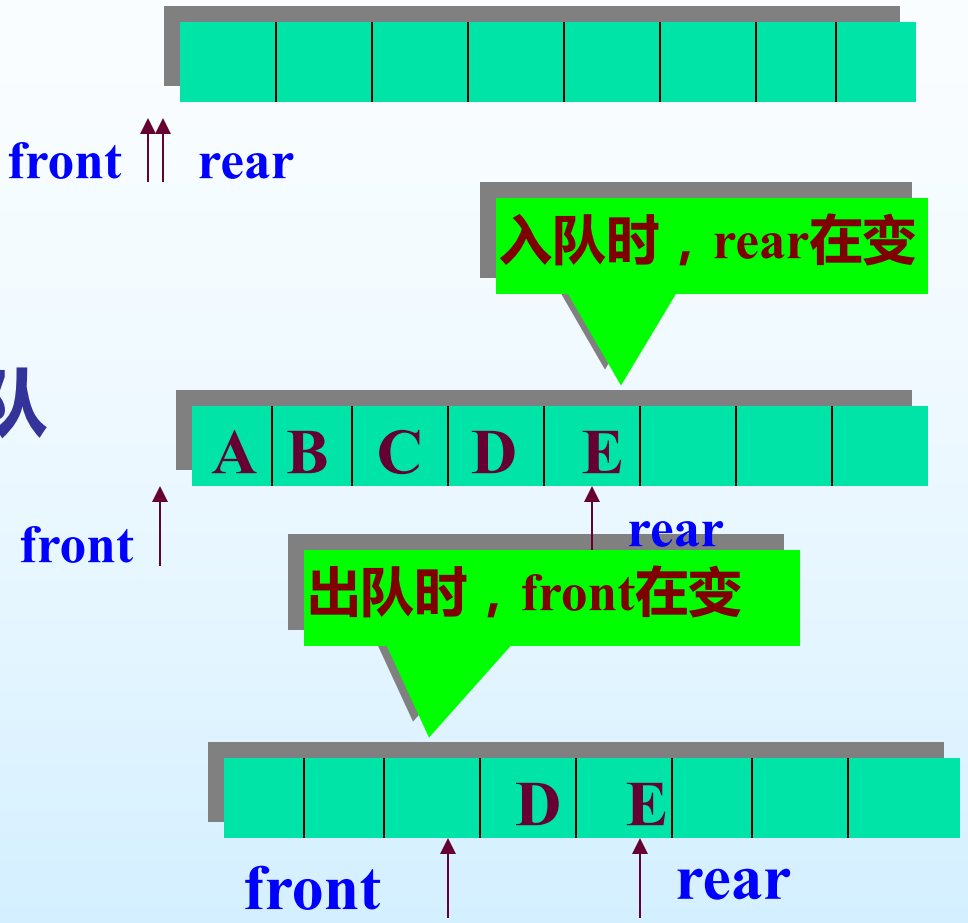
取队头元素

举例：顺序队列的入队、出队操作

（A）空队列

（B）A、B、C、D、E入队

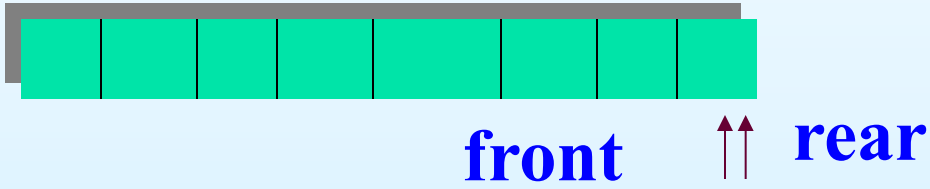
（C）A、B、C出队



举例：顺序队列的入队、出队操作

（D）F、G、H入队

(E）D、E、F、G、H出队，出现假“溢出”



注：一方面队列中是空的，另一方面又出现溢出。

显然，这是逻辑设计上的问题。

循环队列

循环队列的概念

- 如果使当rear = MAXSIZE +1时，即超过队列末端时，令rear = 1；从而使队列的首尾相连接，只有当队列中真正没有空位置时，才产生溢出。
- 设定 queue[0]接在queue[MAXSIZE-1]之后,使得

```
if （rear > MAXSIZE ）
    rear = 1 ;
else
    rear＝rear + 1;
```

这样就构成了循环队列。

循环队列的指针移动

循环队列在指针移动处理时与一般队列不同：

（1） 队头指针

```
front =
   front%MAXSIZE+1;
```

等价于：

```
if（(front) >MAXSIZE）
   front  = 1 ;
else
   front ＝    front + 1;
```

（2）队尾指针

```
rear =
   rear%MAXSIZE+1;
```

等价于：

```
if  （rear >MAXSIZE）
   rear =1;
else
   rear＝    rear + 1;
```

front 和rear 的范围：1--- MAXSIZE

循环队列队空、队满条件

循环队列

front == rear

不能确定是队空还是队满，因此增加一个标志s

- 队空条件

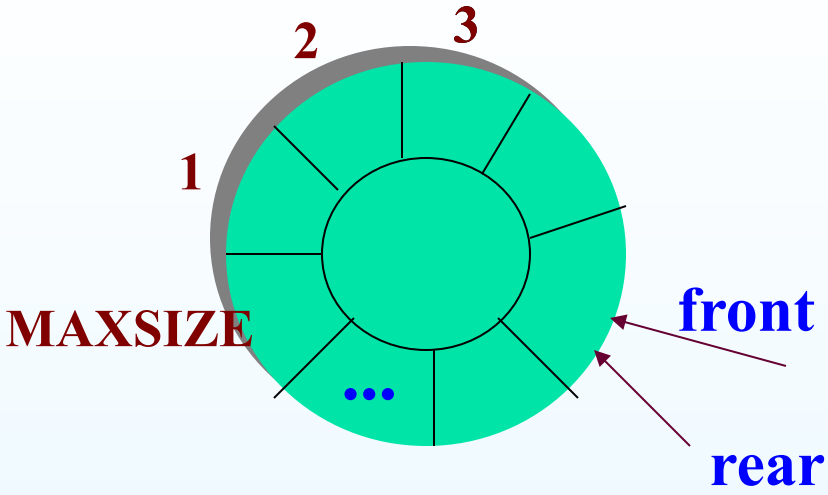
  front == rear ;

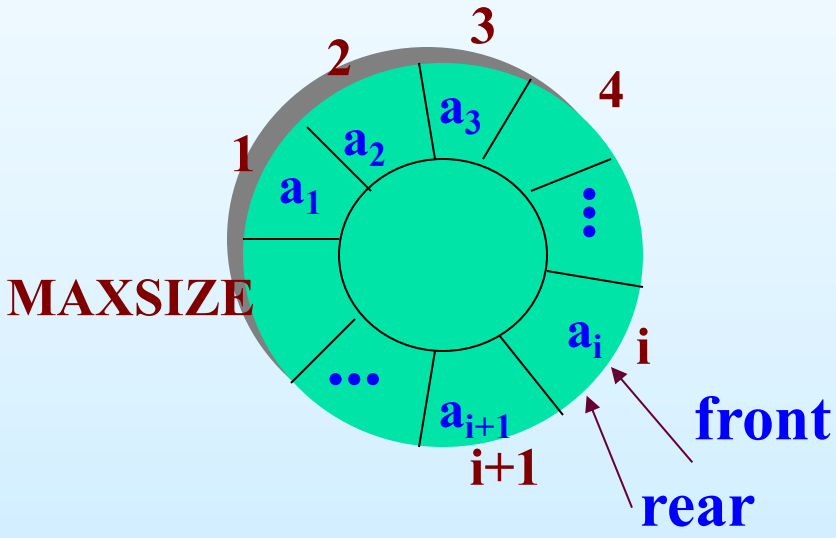
  s=0

- 队满条件

  front == rear

  s=1





```
#define  MAXSIZE  100    /*符号常量MAXSIZE代表队列的最大容量100*/
typedef  char  ElemType;  /*说明新类型ElemType是字符型*/
typedef  struct
{
 ElemType data[MAXSIZE];
 int front;
 int rear;
 int s;
}CircularQueue;
    /*新类型CircularQueue是结构体*/
```

1）构造一个空队列

```
Void InitQueue(CircularQueue *q)
{
   q->front=q->rear= MAXSIZE;
   s=0;
}
```

2）判断队列空

```
int QueueEmpty(CircularQueue *q)
{   /*队列为空返回1，否则返回0*/
   if((q->front==q->rear)&&s==0)
      return(1);
   else
      return(0);
}
```

3）入队

```
void InsertQueue(CircularQueue *q,
     ElemType x)
{
 if((q->front==q->rear)&&s==1)
   {
    printf("\n 队满，上溢！");
    exit(1);
   }
   q->rear=(q->rear+1);
if (q->rear==MAXSIZE+1)   q->rear=1;
q->data[q->rear-1]= x;  //新元素入队
//与机械教材有异，保持与清华教材一致
   s=1;
}
```

4）出队

```
ElemType DeleteQueue(CircularQueue *q)
{
  ElemType x;
  if(QueueEmpty(q))
 { printf("\n 队空，下溢！");
   exit(1);   }
  q->front=q->front+1; /*队头指针加1*/
  if(q->front==MAXSIZE+1) q->front=1;
  x=q->data[q->front-1];/*取出队头元素*/
  if(q->front==q->rear) s=0;
  return x;
}
```

5）读取队头元素

```
ElemType GetHead(CircularQueue *q)
{
  ElemType x;
  if(QueueEmpty(q))
{
  printf("\n 队空，下溢！");
  exit(1);
}
x=q->data[q->front-1]; /*取出队头元素*/
return x;
}
```

课堂练习1

- 循环队列容量为100
- 其序号为：1-100
- front=14 rear=60;
- front=60 rear=14;
- 画出队列示意图 说明有多少个元素

课堂练习2

- 程序练习
- 建立空队列 容量**100**
- 初始化
- 将**9，8，7，6**入队；
- 退队两次 并输出退队元素。

队列的应用

- **OS(Operating System)**中的各种排队器。
- 缓冲区中的循环使用技术。
- 离散事件模拟。
- 给工人分配工作的模拟
- 汽车加油站的工作模拟

作业

假设以数组**sequ[m]**存放循环队列的元素，设变量**rear**和**quelen**分别为指示队尾元素位置和队中元素个数，试写出入队和出队算法。