

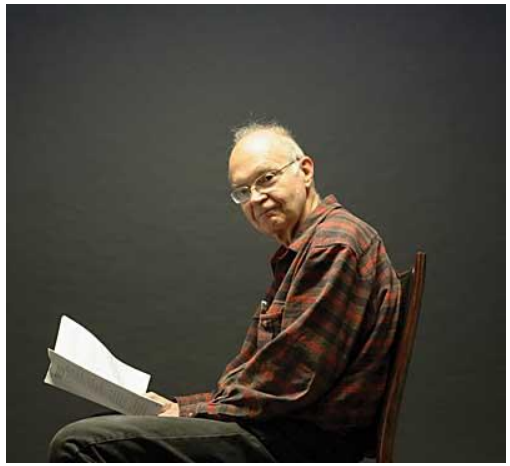
第6讲 数组（Part I）

周水庚

2026年10月15/22日

最伟大的程序员
——高德纳（Donald E. Knuth）

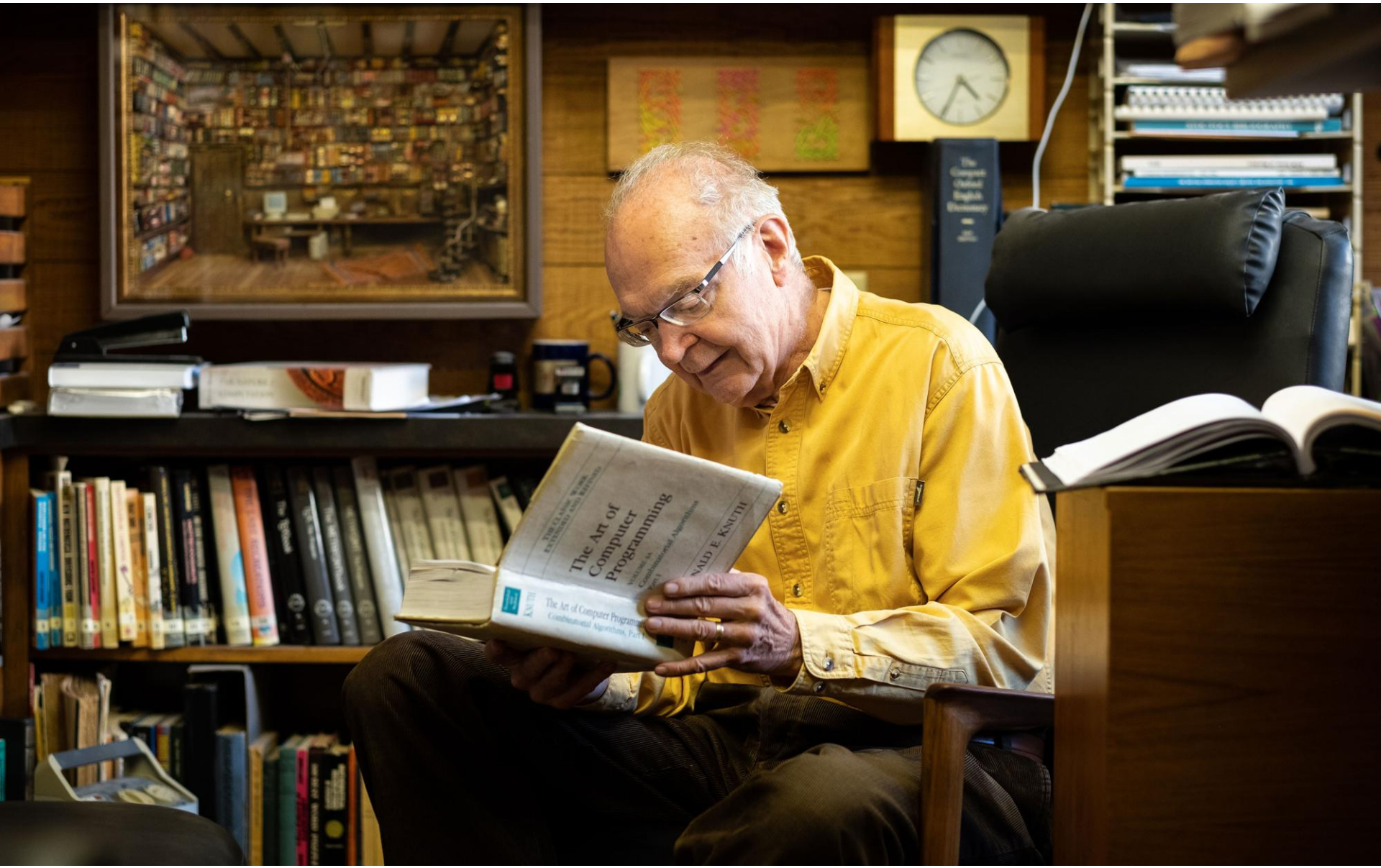
- **高德纳**（Donald Ervin Knuth，1938年1月10日生于美国威斯康辛的密尔沃基（Milwaukee），和司马贺（H. A. Simon）是同乡
- 1963年获加州理工博士学位，留校工作至1968年，然后转入斯坦福大学任教。为了专心完成他的写作计划，他1993年（**55岁**）即从斯坦福大学退休
- 1974年获图灵奖时36岁，迄今为止最年轻的获奖者
  - For his major **contributions to the analysis of algorithms and the design of programming languages**, and in particular for his contributions to the "**art of computer programming**" through his well-known books in a continuous series by this title.
- 世界著名计算机科学家，主要贡献
  - 《计算机程序设计艺术》（The Art of Computer Programming）（4卷）、排版软件TEX和字体设计系统Metafont







《程序设计》-2026年秋季



《程序设计》-2026年秋季

《计算机程序设计的艺术》(The Art of Computer Programming)

- 计算机科学理论与技术的经典巨著，有评论认为其作用与地位可与数学史上欧几里得的《**几何原本**》相比
- 1999年，《美国科学家》杂志列为**20世纪最重要的12本物理科学类专著之一**，与爱因斯坦《相对论》、狄拉克《量子力学》、理查·费曼《量子电动力学》等经典比肩而立
- 目前已经完成并出版卷1（1968）、2（1969）、3（1973）和4A（2011）/B（2023），预计到2030年完成卷4。计划中，还有卷5、6和7

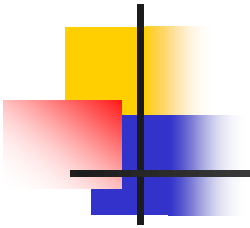




https://www-cs-faculty.stanford.edu/~knuth/taocp.html

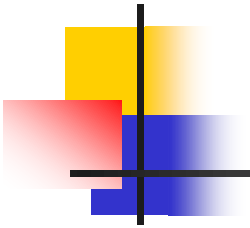
C语言中的复杂数据类型

- **复杂数据类型**是由若干基本数据类型或其它复杂数据类型按一定的方法(规则)构造而成的(也称“**导出类型**”)
- C语言中构造复杂数据类型的机制有**数组**、**结构**和**联合**。在基本数据类型和指针等类型基础上，通过反复应用这几种C语言提供的构造手段，**能描述复杂的数据结构**
- **说明或定义复杂数据**必须**指出它的成分元素的名称与类型及其构造方法**。对于复杂类型的变量来说，重点是访问它的元素的方法



提要

- 数组的基本概念
- 一维数组(One-dimensional Array)
- 多维数组(Multi-dimensional Array)



提要

- **数组的基本概念**
- 一维数组
- 多维数组

数组的基本概念

- 数组是由若干同类元素组成的对象
- 数组的每个元素的数据类型相同，元素个数固定，其元素按顺序存放，每个元素对应一个序号（称为下标），各元素按下标存取（引用）
- 数组元素的存储顺序与其下标相对应，数组元素的下标从0开始顺序编号

数组概念说明

- 数组元素在数组中的**下标是固定不变**的，而数组元素是变量，其**值是可以变化**的。数组元素与相同类型的独立变量一样使用
- 程序借助于数组定义时为元素变量隐含设定的**下标**，**引用数组元素**
- 引用数组元素所需的**下标个数**由**数组的维数**决定
  - 数组有一维数组、二维数组和多维数组之分

数组举例

- 一行文本可以表示成由字符组成的数组

  ```
  char  s[120];   /*字符数组*/
  ```

- 一个整数向量可以表示成由整数组成的数组

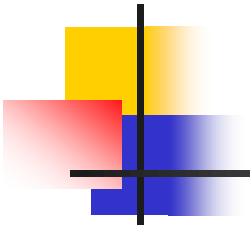
  ```
  int intVector[80];   /*由80个整数组成的数组*/
  ```

- 一个矩阵就可以表示成由向量构成的数组

  ```
  double matrix[40][50];  /*40行X50列实数矩阵*/
  ```

- 学生成绩表可表示成由学生成绩单组成的数组

  ```
  int score[40][7]; /* 40名学生课程成绩，每个学生学7门课程 */
  ```



提要

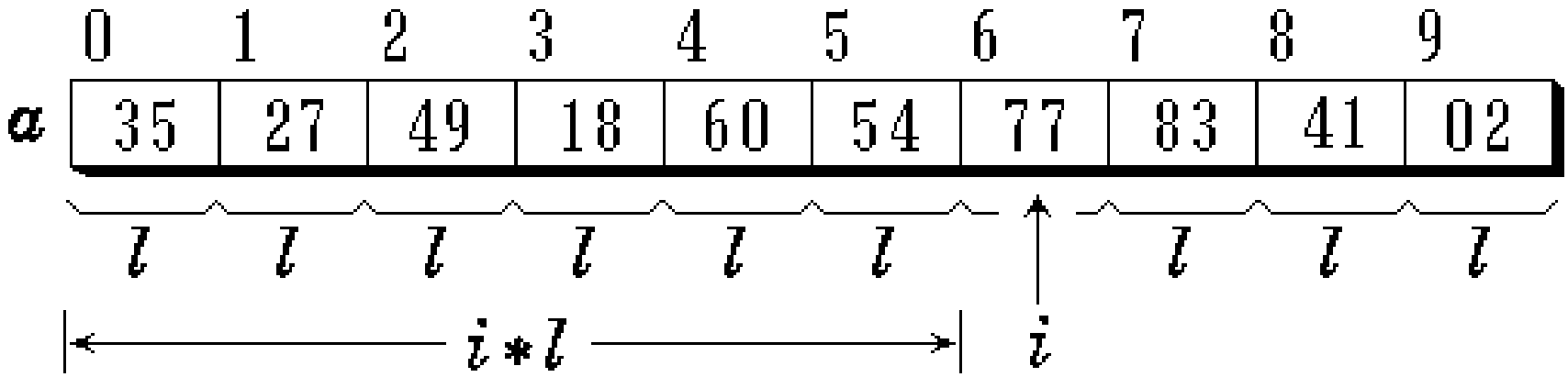
- 数组的基本概念
- **一维数组**
- 多维数组

一维数组定义

- **定义形式：　类型说明符　数组名[常量表达式];**
  - **类型说明符**用来指明数组元素的类型，同一数组元素的类型相同
  - 数组逻辑上是一个“变量”，用标识符命名，**数组名**遵守标识符的命名规则
  - **方括号“[]”**是数组的标志，其中的常量表达式的值表示数组的元素个数，即数组的长度
  - **常量表达式**通常是整型常量、字符常量或 sizeof(类型名)，以及由它们组成的常量表达式
  - C语言约定，**当数组名单独在程序中使用时，数组名可以代表为它分配的内存区域的开始地址，即数组中下标为0的元素的地址**

一维数组示例

int a[10]={35,27,49,18,60,54,77,83,41,2};



一维数组元素的引用

- 引用形式: **数组名[下标]**

- 例如， int a[5];
  - 数组a的5个元素可分别用a[0]，a[1]，a[2]，a[3]，a[4]来引用
  - 如

```
a[4] = a[0] + a[1] + a[2] + a[3];
a[0] = a[i+z];  /* 如果 0 <= i+z < 5 */
```

一维数组元素的引用示例

- 设有定义： int x[20], i;
  - 顺序输入数组x的全部元素

```
printf("Enter data for array x[].\n");
for(i = 0; i < 20; i++)
    scanf("%d", &x[i]);
```

  - 顺序输出x的全部元素

```
for(i = 0; i < 20; i++)
    printf("%d\t", x[i]);
```

一维数组元素的引用示例(续)

- 设有定义： int x[20], i;
  - 分别统计数组x中大于、等于和小于 0 的元素个数

```
great = equal = less = 0;
for(i = 0; i < 20; i++)
    if (x[i] > 0) great++;
    else if (x[i] == 0) equal++;
        else less++;
```

一维数组定义及元素引用程序样例

```
#include <stdio.h>
int main()
{ int i, digits[10];
  for(i = 0; i < 10; ++i)  digits[i] = i;
  for (i = 0; i < 5; ++i)
      printf("%3d", digits[2*i]);
  printf("\n");
  for(i = 0; i < 5; ++i)
      printf("%3d", digits[2*i+1]);
  printf("\n\n\n");
      return 0;
}
```

程序运行结果

```
  0  2  4  6  8
  1  3  5  7  9
```

数组初始化

- 数组定义的初始化或**数组初始化**指的是**数组定义同时给出元素初值**的表述形式
- 数组初始化形式有
  - **数组定义时，顺序列出数组全部元素的初值**
  - **只给数组的前面一部分元素设定初值**
  - **当对数组的全部元素都明确设定初值时，可以不指定数组元素的个数**

数组初始化示例-1

- **数组定义时，顺序列出数组全部元素初值**
  - 例如：　int d[5] = {0, 1, 2, 3, 4};
  - 将数组元素的初值依次写在一对花括弧内，有

    d[0]=0；d[1]=1；d[2]=2；d[3]=3；d[4]=4

数组初始化示例-2

- 只给数组的前面一部分元素设定初值
  - 例如： int e[5] = {0, 1, 2};
  - 定义数组e有5个整型元素，其中前3个元素设定了初值，而后2个元素未明确地设定初值
  - **系统约定，当一个数组的部分元素被设定初值后，那些未设定初值的元素自动被设定0值。**所以，数组e的后2个元素的初值为0；如果实型数组，未被初始化的元素置成0.0f (float)/0.0 (double)/0.0l (long double)
  - 当定义数组时，**如未对它的元素指定过初值，对于局部数组，则它的元素的值是不确定的；对于全局或静态局部数组，没有显示初始化，自动置零。**

数组初始化示例-3

- 当对数组的全部元素都明确设定初值时，可以不指定数组元素的个数
  - 例如： int g[] = {5, 6, 7, 8, 9};
  - 由花括号内的初值个数确定数组的元素个数
  - 若提供的初值个数小于数组希望的元素个数时，则方括号中的数组元素个数不能省略
    - 例如: int b[10] = {1,2,3,4,5}
    - 数组b有10个元素,前5个如设定所示，后5个都为0
  - 注意：初值个数不允许超过数组元素个数
    - 例如：int c[5] = {0, 1, 2, 3, 4, 5}；是错误的代码

一维数组程序实例-1

- **用数组计算Fibonacci(斐波那契)数列1、1、2、3、5、8、……的前30项值**
- 程序实例分析
  - Fibonacci数列有规律
    - f[0] = f[1] = 1;
    - f[k] = f[k-2] + f[k-1];(k = 2,3,4,……)
  - 利用数组存储Fibonacci数列的各项值，其中，数列的前2项可用初值为它设定

一维数组程序实例-1（续）

```
#include <stdio.h>
#define  MAXN   30
#define  Aline  10
int main()
{  long f[MAXN] = {1L, 1L};  int  k;
   for(k = 2; k < MAXN; k++)
     f[k] = f[k-2] + f[k-1];
   for(k = 0; k < MAXN; k++) {
     if(k%Aline == 0) /* 每行输出Aline个数 */
         printf("\n");
     printf("%8ld", f[k]);
   }
   printf("\n");
   return 0;
}
```

一维数组程序实例-2

- 在数组中，找一个值等于变量key值元素的下标
- 程序实例分析
  - 在数组的n个元素中找值为key的元素，有多种解法
    - 直观解法
    - 设置哨兵法
    - 若已顺序存放，则用二分法

一维数组程序实例-2（续）

- **在数组中找一个值等于变量key值元素的下标-直观解法**
  - 从数组的第一个元素开始，顺序查找至数组的末尾，若存在值为key的元素，程序就终止查找过程；若不存在值为key的元素， 程序将找遍整个数组。程序代码如下

```
for(i = 0; i < n; i++)
    if(key == a[i]) break; /*找到终止循环,若找到则i<n;否则i=n*/
```

  - 查找过程也可用以下代码来描述

```
for(i=0;i<n&&key!=a[i];i++); /*找到结束循环,若找到i<n;否则i=n*/
```

  - 假定每个元素的被查找的机会均等，则采用上述查找方法，访问元素的平均个数v为： v = (1+2+3+...+n)/n = (n+1)/2
  - 若在数组中没有指定值的元素，则需查找n次

一维数组程序实例-2（续）

- 在数组中找一个值等于变量key值元素下标-设置哨兵
  - 先在数组第n个元素的后面（如数组定义时已预留了空闲的元素），第n+1元素位置放入要寻找的值（设置了一个哨兵），然后再从第一个元素开始顺序寻找，这能简化寻找循环的控制条件

```
a[n] = key;
for(i = 0; key != a[i]; i++); /*若找到i<n；否则i=n*/
```

  - 这种写法，数组需多用一个元素，但程序比前一种更简单

一维数组程序实例-2（续）

- **在数组中找一个值等于变量key值元素下标-二分法**
  - 假定数组a的元素已**按它们的值从小到大的顺序存放**，则二分法是更好的查找方法
  - **算法基本思想**是对任意a[i]到a[j](i<=j)的连续一段元素，根据它们值的有序性，试探位置m=(i+j)/2上的元素，a[m]与key比较有三种可能结果，分别采取三种对策
    - key = a[m]；找到，结束查找
    - key > a[m]；下一轮的查找区间为 [m+1，j]
    - key < a[m]；下一轮的查找区间为 [i，m-1]
  - 当 j<i 时，区间[i,j]变为一个空区间，即表示在数组a中没有值为key的元素。由于每轮查找后，使查找区间减半，因此称为二分法查找。初始查找区间为i=0，j=n-1

一维数组程序实例-2（续）

- **在数组中找值等于变量key值元素下标-二分法(续)**
  - 二分法查找可用C代码描述如下

```
i = 0 ;   j = n-1 ; /* 设定初始查找区间 */
while(i <= j) {
      m = (i+j)/2 ;
      if (key == a[m])  break;
      else if (key > a[m])
                i = m+1;
           else
                j = m-1;
    }
/* 找到时，i <= j，a[m] == key；否则 i > j */
```

- **问题：找到数组中所有等于key的元素**

二分搜索算法的历史

- **思想起源**：1946年约翰·莫奇利（John Mauchly）在摩尔学院讲座中首次提及了二分搜索
  - 莫奇利，美国物理学家/工程师，与普雷斯珀·埃克特合作，设计了世界上第一台通用电子数字计算机 ENIAC（1946年），以及后续的 EDVAC、BINAC 和第一台商用计算机 UNIVAC I
- **正确实现**：1960年德里克·亨利·莱默（Derrick Henry Lehmer）才发表了首个正确的二分搜索算法
  - 莱默，美国数学家，在1945-1946年间在 ENIAC 上工作，并兴趣在于伪随机数的生成
- **后续优化**：1962年赫尔曼·博滕布鲁赫（ Hermann Bottenbruch）在 ALGOL 60语言中实现了另一种优化版本
  - 博滕布鲁赫，德国数学家和计算机科学家， ALGOL 58 国际工作组的成员，参与了这一重要算法语言的制定

一维数组程序实例-3

- **从文件输入一批数据到数组，对数组的元素用冒泡法排序，然后输出**
- 程序实例分析
  - 设文件中的元素不超出过1000个，程序可先把它们从文件输入到数组，然后对数组的元素进行排序
  - 把数组元素按值的大小进行整理，数组元素值按从小到大(或从大到小)的顺序重新存放的过程称为**数组排序**
  - **冒泡法排序的思想**是对数组作多次比较调整遍历，每次遍历是对遍历范围内的相邻两个数作比较和调整，将小的调到前面，大的调到后面(设从小到大排序)

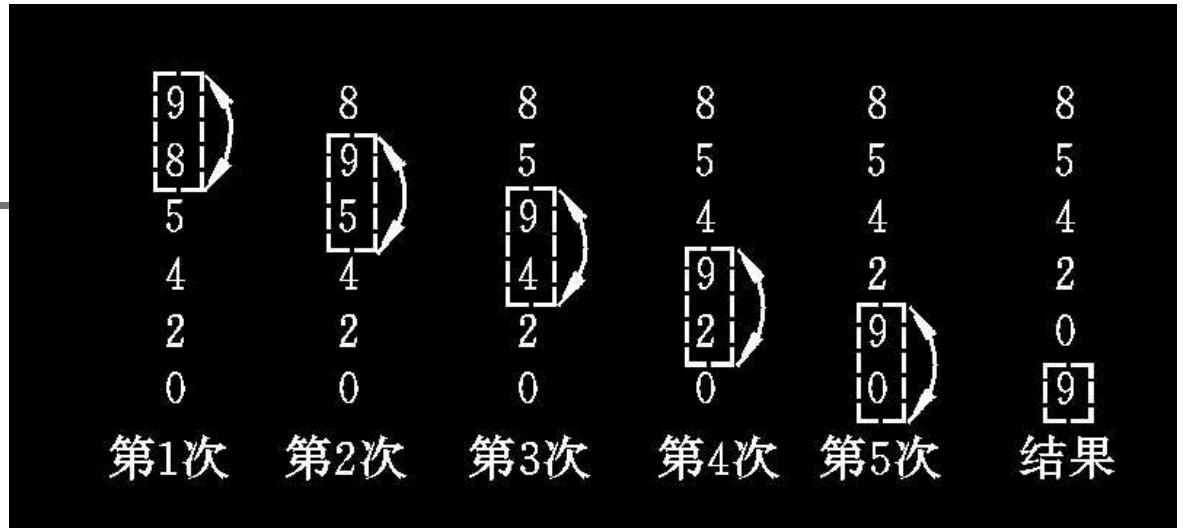
一维数组程序实例-3(续)

- 冒泡排序
  - 设要排序数组a有n个元素，冒泡法排序共要作n-1次遍历
    - **第一次遍历：**从a[0]开始，依次对相邻两个元素进行比较调整。即a[0]与a[1]比较调整；a[1]与a[2]比较调整；……；直至a[n-2]与a[n-1]比较调整，每次比较调整使小的调到前面，大的调到后面。这次遍历共做了n-1次比较调整，使值最大的元置于a[n-1]
    - **第二次遍历：**再次从a[0]开始，依次对两个相邻元素作比较调整，直至a[n-3]与a[n-2]的比较调整。这次遍历共做了n-2次比较调整，使次最大元置于a[n-2]
    - **依次类推，直至第n-1次遍历，**仅对a[0]和a[1]作比较调整。至此，完成对数组从小到大排序的要求

一维数组程序实例-3(续)

- 冒泡排序 - 示例
  - 如有定义： int a[] = {7,8,4,5,1};
  - 对a的5个数的冒泡排序（从小到大）过程为
    - **第一次遍历：**首先7与8比较，与要求顺序一致，不必对调；第二次8与4比较，与要求顺序不一致，需对调；第三次是8与5的比较和对调；直至第四次8与1的比较和对调。第一次遍历后，使a[]={7,4,5,1,8}，a的最大数置于a[4]
    - **第二次遍历：**首先7与4比较，对调；第二次7与5比较，对调；直至第三次7与1的比较和对调。第二次遍历后，a[]={4,5,1,7,8}，a的次最大数置于a[3]
    - **第三次遍历：**首先4与5比较，不必对调；第二次5与1比较，对调。第三次遍历后，使a[]={4,1,5,7,8}，a原来的元素5被置于a[2]
    - **第四次遍历是最后一次，**只需作一次比较调整，4与1比较，对调。最后使a[]排好序，为a[]={1,4,5,7,8}

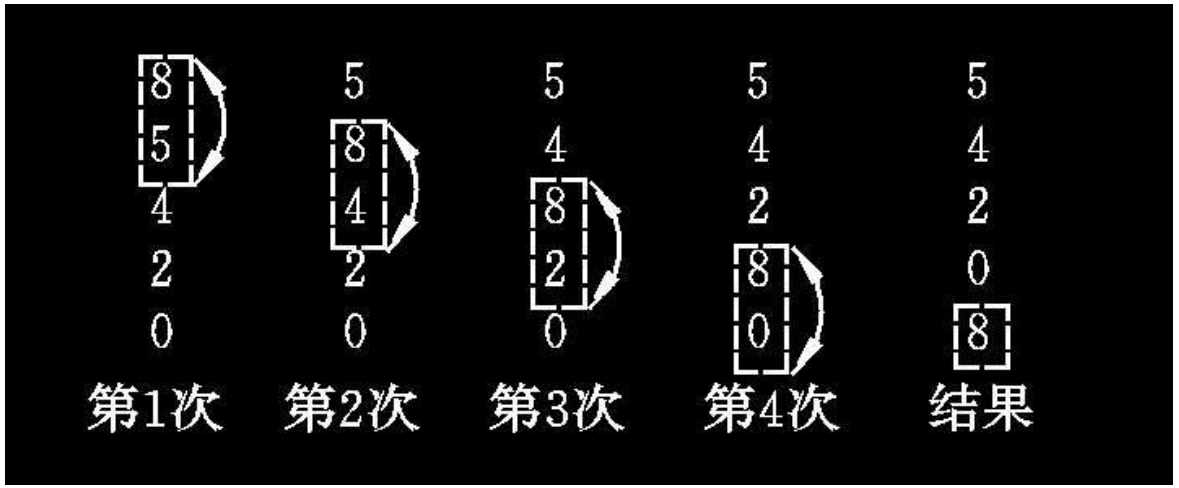
第一轮



例子：

9 8 5 4 2 0

第二轮



一维数组程序实例-3(续)

- **冒泡排序 - 示例**
  - 用C语言描述上述排序过程如下

```
for(i = 0; i < n-1; i++)   /* 控制n-1次比较调整遍历 */
  for(j=0;j<n-1-i; j++)    /* 第i次遍历需比较 n-1-i 次 */
    if(a[j]>a[j+1]){       /* a[j]与a[j+1] 交换 */
        temp   = a[j];
        a[j]   = a[j+1];
        a[j+1] = temp;
    }
```

- 总共要进行n(n-1)/2次比较
- 最多要进行n(n-1)/2次调整

一维数组程序实例-3(续)

- **完整程序：**从文件输入一批数据到数组，对数组的元素用冒泡法排序，然后输出

```
#include <stdio.h>
#define MAXN 1000
FILE *fp;  char fname[40];
int main()
{  int i, n, j, temp, k; int x, a[MAXN];
   printf("输入文件名。\n");
   scanf("%s",fname);
   if ((fp = fopen(fname, "r")) == NULL) {
      printf("不能打开文件%s。\n", fname);
      return 0;
    }
   n = 0;
```

一维数组程序实例-3(续)

- **完整程序(续):** 从文件输入一批数据到数组，对数组的元素用冒泡法排序，然后输出

```
while (n < MAXN && fscanf(fp, "%d", &x) == 1)
        /* 当还能从文件读入数时，循环 */
   a[n++] = x;
fclose(fp);
/* 对数组a的前n个元素采用冒泡排序算法排序 */
for(i = 0; i < n-1; i++)
  for(j = 0; j < n-1-i; j++)
    if(a[j] > a[j+1]) {
      temp   = a[j];
      a[j]   = a[j+1];
      a[j+1] = temp;
     }
```

一维数组程序实例-3(续)

- **完整程序(续):**
  从文件输入一批数据到数组，对数组的元素用冒泡法排序，然后输出

```
for(k = 0, i = 0; i < n; i++) {
    if (k++ % 5 == 0) printf("\n");
    printf("%d\t", a[i]);
  }
  printf("\n\n");
  return 1;
}
```

排序例子

- 对９８０２４５排序
  - 第1轮后：８０２４５９（交换5次）
  - 第2轮后：０２４５８９（交换4次）
  - 第3轮后：０２４５８９（交换0次）
  - 实际上，第2轮就已经排好。但是，第2轮过后，你并不知道已经排好了，要到第3轮你才知道。所以，总共需要遍历3次！

不是所有的排序都要遍历(n-1)次！

冒泡排序法的改进1

- 在冒泡排序过程中，如果某次遍历未发生交换调整情况，这时数组实际上已排好序。程序若发现这种情况后，应提早结束排序过程
- 为此，程序引入一个标志变量
  - 每次遍历前，预置该变量的值为0
  - 当发生交换时，置该变量的值为1
  - 一次遍历结束时，就检查该变量的值，若其值为1，说明发生过交换，继续下一次遍历；
  - 如该变量的值为0，说明未发生过交换，则立即结束排序循环

冒泡排序法的改进1(续)

- 引入标志变量后相应的排序代码

```
for(i=0;i < n-1;i++){    /* 控制 n-1 次比较调整遍历 */
  for(flg=j=0;j<n-1-i;j++)    /* 比较 n-1-i 次 */
    if(a[j] > a[j+1]) {
          temp   = a[j];  a[j]   = a[j+1];
          a[j+1] = temp;  flg    = 1;
    }

   if (flg == 0) break;
}
```

排序例子

- 对 9 8 3 4 2 5排序
  - 第1轮交换5次后：8 3 4 2 5 9
  - 第2轮交换4次后：3 4 2 5 8 9
  - 第3轮交换1次（不是3）后：3 2 4 5 8 9
  - 第4轮交换1次（不是2）后：2 3 4 5 8 9
  - 第5轮交换0次
  - 结束后，总共需要遍历5次，交换11次！

并非第*i*次遍历，一定要进行 (*n*-1-*i*)次比较/交换！

冒泡排序法的改进2(续)

- **更好的改进**：在冒泡排序过程中，若某次遍历在某个位置j发生最后一次交换，j以后的位置都未发生交换，则实际上j以后位置的全部元素都是已排好序的
  - 利用这个性质，后一次遍历的范围可立即缩短至上一次遍历的最后交换处
  - 程序为利用这个性质，引入每次遍历的上界变量end，另引入变量k记录每次遍历的最后交换位置
  - 为了考虑到可能某次遍历时一次也不发生交换的情况，在每次遍历前时，预置变量k为0。一次遍历结束后，将k赋给end，作为下一次遍历的上界。
  - 冒泡排序过程至end为0结束，end的初值为n-1，表示第一次遍历至最后一个元素

冒泡排序法的改进2(续)

- **更好的改进：按上述想法把冒泡排序代码改进如下**

```
end = n-1;              /* end 用于记录本次遍历范围的上界 */
while (end > 0) {       /* 扫视范围非空时循环 */
  for(k = j = 0; j < end; j++)    /* 比较至end */
    if(a[j] > a[j+1]) {
       temp   = a[j];    a[j]   = a[j+1];
       a[j+1] = temp;
       k = j;           /* 记录发生交换的位置 */
    }
    end = k;
}
```

排序例子

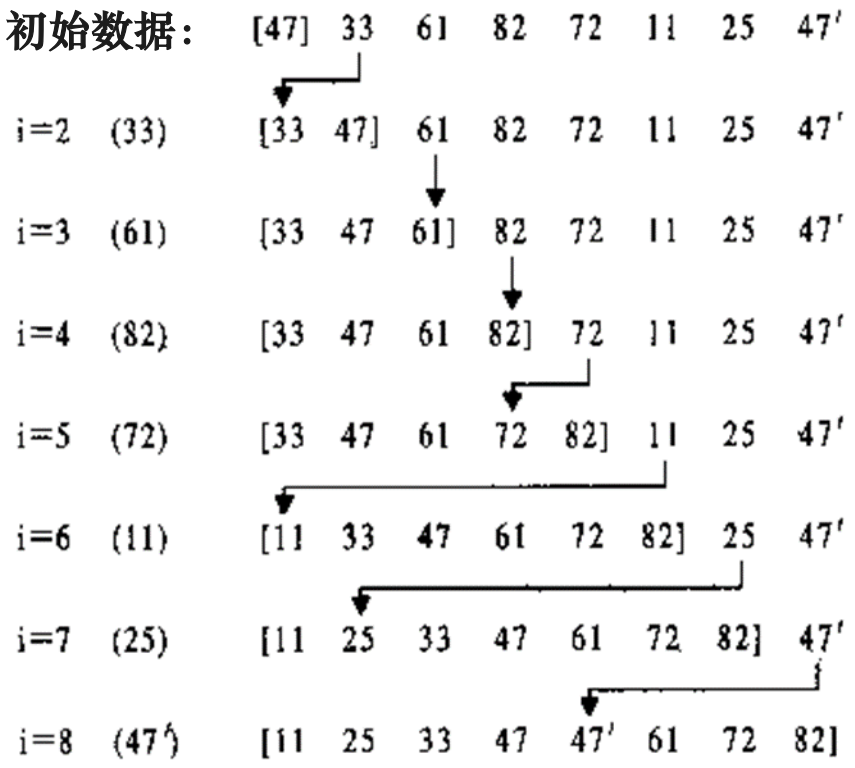
- 对９８０４２５排序
  - 第1轮后：８０４２５９（比较5次，交换5次）
  - 第2轮后：０４２５８９（比较4次，交换4次）
  - 第3轮后：０２４５８９（比较3次，交换1次）
  - 第4轮后：０２４５８９（比较1次，交换0次）
  - 结束，共遍历4次，比较13次，交换10次！
  - 按第1个算法，比较15次，总共遍历5次
  - 按第2个算法，比较14次，总共遍历4次

冒泡排序算法的历史

- **算法的早期记载**：爱德华·H·弗兰德（Edward H. Friend） 在1956年发表的文章《Sorting on Electronic Computer Systems》。不过，当时它并不叫“冒泡排序”，而是被归类为“交换排序”（Exchange sort）
- **名称的首次出现**：肯尼斯·艾弗森（Kenneth Iverson）在1962年的著作《A Programming Language》中，首次在印刷物中使了“bubble sort”这一名称。因此，他通常被认为是这个名称的创造者，而不一定是算法本身的发明人
  - 艾弗森，加拿大裔美国计算机科学家，计算机编程语言先驱，因创立APL语言而闻名，1979年图灵奖得主。曾在哈佛大学任教，后长期任职于IBM
- 常见的误传：网络上有说法称冒泡排序由冯·诺伊曼在1945年提出，但这缺乏可靠依据，冯·诺伊曼实际贡献的是归并排序。此外，快速排序的发明者托尼·霍尔也曾提到，他在发明快排之前先想到了冒泡排序，但这属于个人回忆，并非算法起源的定论

直接插入排序-1

- **算法思想**
  - 将序列中的数据一个个插入数组中
  - 当插入第i（$i\geq1$）个数据时，前面的$V[0]$，$V[1]$，…，$V[i-1]$个数据已经排好序，这时，用$V[i]$与$V[i-1]$，$V[i-2]$，…顺序进行比较，找到插入位置即将$V[i]$插入，原来位置上的对象向后顺移。



直接插入排序示例

直接插入排序-2

```
void InsertionSort(int a[], int size)
{
    for(int i = 1; i < size; i++)
    {
        int temp = a[i];
        int j = i - 1;
        while(j >= 0 && temp < a[j])
        {
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = temp;
    }
}
```

```
void main()
{
    int array[5]={8, 9, 28, 11, 23, 8};

    InsertionSort(array, 5);

    for(int i = 0; i < 5; i++)
    {
        printf("%d ", array[i]);
     }
      printf("\n");

}
```

一维数组程序实例-4

- 输入n个整数，找出其中出现次数最多的整数，并输出其在数组中最早出现的位置




```
#include <stdio.h>
#define MAXN 1000
int main()
{  int i, n, tc, c, j, pos;  int a[MAXN];
   printf("Enter n   "); scanf("%d", &n); /* 输入参数n*/
   for(i = 0; i < n; i++) {  /* 输入数组数据*/
     printf("Enter  a[%d]  ", i);
     scanf("%d", &a[i]);
    }
```

一维数组程序实例-4(续)

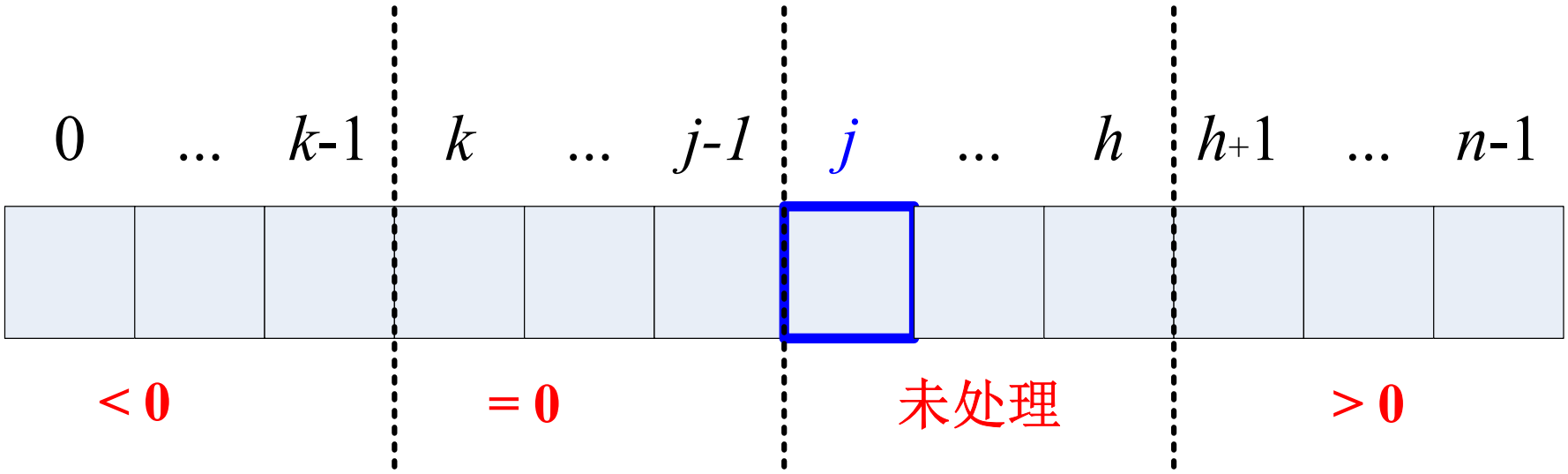



```
for(c = i = 0; i < n-1; i++) {
   for(tc = 1, j = i+1; j < n; j++)    /* 统计a[i]的出现次数 */
        if (a[j] == a[i]) tc++;
   if (tc > c) {      /* 找到出现次数更多的数 */
        c = tc;       pos = i; /* 记录当前找到的出现次数最多的数 */
    }
 }
 printf("COUNT = %d  VALUE = %d\n\n\n", c, a[pos]);
 return 0;
}
```

思考：1）如何改进这个程序？
2）如果数据已经排好序，该如何写程序？

一维数组程序实例-5

- **输入n个整数存于数组，对数组作整理，使其中小于0的元素移到前面，等于0的元素留在中间，大于0的元素移到后面**
- 程序实例分析
  - 引入变量k、h，设在整理过程中，a[0]~a[k-1]都小于0，a[h+1]~a[n-1]都大于0。
  - 从a[0]开始顺序考察。设当前正要考察a[j]，则有a[k]至a[j-1]=0, a[j]至a[h]还未考察过。对元素a[j]有下列情况
    - a[j] < 0 : a[j]与a[k]交换，并且j++和k++;
    - a[j] = 0 : j++;
    - a[j] > 0 : a[j]与a[h]交换，并且h--;
  - 初始时，j = 0, k = 0, h = n-1；整理结束后满足: j > h



初始状态： $k=j=0$; $h=n-1$

一维数组程序实例-5(续)

- （续）数组整理程序

```
#include  <stdio.h>
#define MAXN 1000
int main()
{ int j, n, k, temp, h, m;  int a[MAXN];
  printf("Enter n\n");   scanf("%d", &n); /* 输入数组中实际数据*/
  printf("Enter a[0] -- a[%d]\n", n-1);
  for(j = 0; j < n; j++)  scanf("%d", &a[j]);
```

一维数组程序实例-5(续)

（续）数组整理程序

```
    j = k = 0; h = n-1;
    while (j <= h)
      if (a[j] < 0) { temp=a[j];  a[j]=a[k];  a[k]=temp;
            j++; k++;
        }
      else if (a[j] == 0)  j++;
      else { temp=a[j]; a[j]=a[h]; a[h]=temp;
        h--;
        }
    for(j = 0; j < n; j++) printf("%4d", a[j]);/* 输出结果*/
    printf("\n\n\n");
    return 0;
}
```

一维数组程序实例-6

- 编一个程序，输入n个互不相等的整数存于数组，并输出。程序如发现输入的数据已输入过，则要求重新输入

```
#include  <stdio.h>
#define MAXN 1000
int main()
{ int i, n, k;   int a[MAXN];
  printf("Enter n   "); scanf("%d", &n);
```

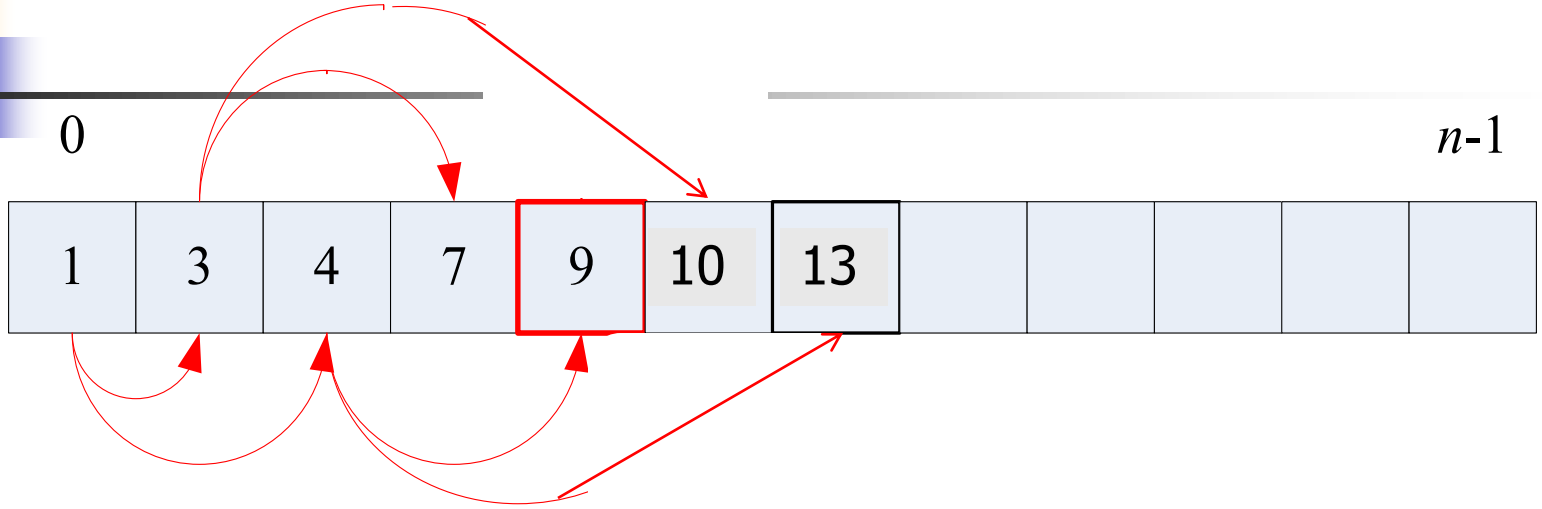
一维数组程序实例-6(续)

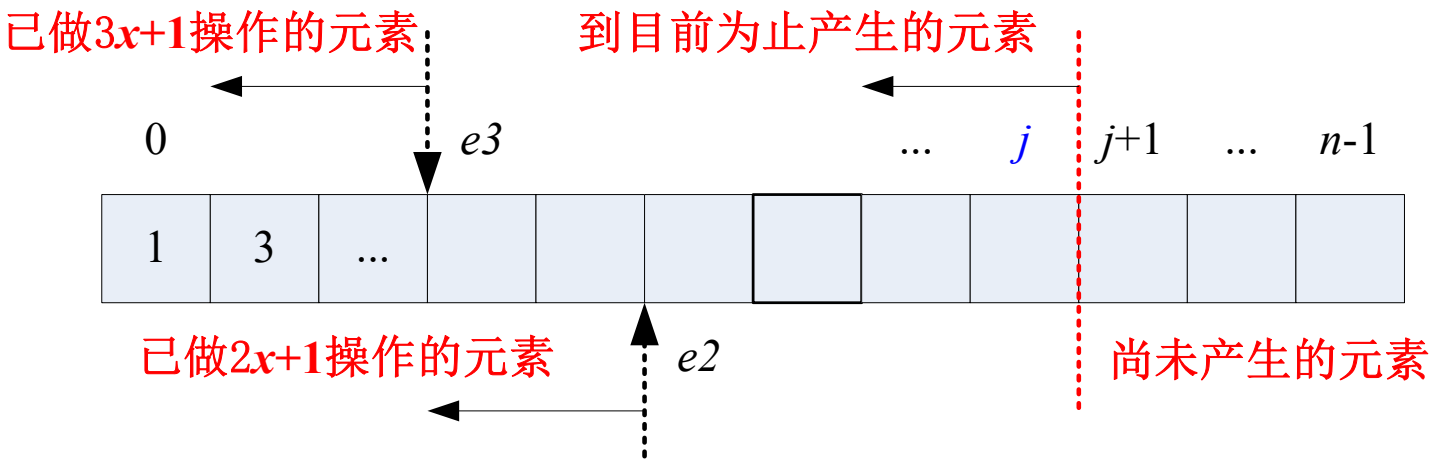
（续）输入n个互不相等的整数存于数组，并输出

```
    i = 0;
    while (i < n) {
        printf("Enter  a[%d]  ", i);
        scanf("%d", &a[i]);
        for(k=0;a[k]!=a[i];k++);     /* 查是否有重复 */
        if ( k < i )     /* 如果重复 */
            printf("Error, input again!\n");
        else i++ ;      /* 对于不重复情况 */
     }
    for(i = 0; i < n; i++)  printf("%4d", a[i]);
     printf("\n\n\n");
     retun 0;
}
```

一维数组程序实例-7

- 按递增顺序生成集合M的前n个元素。
- 集合M定义如下
  - 整数1属于M
  - 如果整数X属于M，则整数2x+1和3x+1也属于M
  - 再没有别的整数属于M
- 程序实例分析
  - 根据集合M的定义，前几个元素为 M={1，3，4，7，9，…}
  - 为存储M的元素设计一个足够大的整数组m，用j表示集合M中已生成的元素个数。按题意，首先将1放入集合M中，然后按递增顺序用M中的现有元素枚举出M的新元素


0
n-1
1
3
4
7
9
10
13


已做3x+1操作的元素
到目前为止产生的元素
0
e3
...
j
j+1
...
n-1
1
3
...
已做2x+1操作的元素
e2
尚未产生的元素

一维数组程序实例-7(续)

- 程序实例分析（续）
  - 每个元素可以施行两种枚举方法。在按递增顺序枚举出M的新元素过程中，把M的元素分成以下四部分
    - 已执行过2x+1枚举操作的元素
    - 已执行过3x+1枚举操作的元素
    - 已被枚举生成，但还未执行过任何枚举操作
    - 因还未被枚举出来，其值还未定义
  - 引进辅助变量e2 和e3，分别用于指出m中下一个要执行2x+1枚举的元素的下标和要执行3x+1枚举的元素的下标。为了按递增顺序生成M的元素，下一个可能执行枚举操作的元素是m[e2]和m[e3]，选择哪一个由它们枚举出来的值更小来确定
  - 另设j为m中已有元素个数

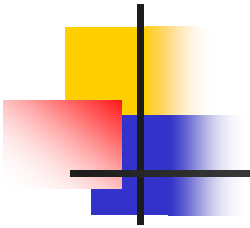
一维数组程序实例-7(续)

- 以下代码能描述m在第一次枚举之前的状态

```
m[0] = j = 1;   e2  = e3 = 0;
```

- 而一系列的枚举生成m的n个元素的C代码描述如下

```
while (j < n) {
   if (m[e3]*3+1 >= m[e2]*2+1) {   /* 对m[e2]执行2x+1枚举 */
      m[j] = m[e2++]*2+1;
      if (m[e3]*3+1 == m[j])
         e3++;     /* 当两个元素枚举值相同时，同时枚举 */
    }
   else  m[j] = m[e3++]*3+1 ;
   j++;
}
```

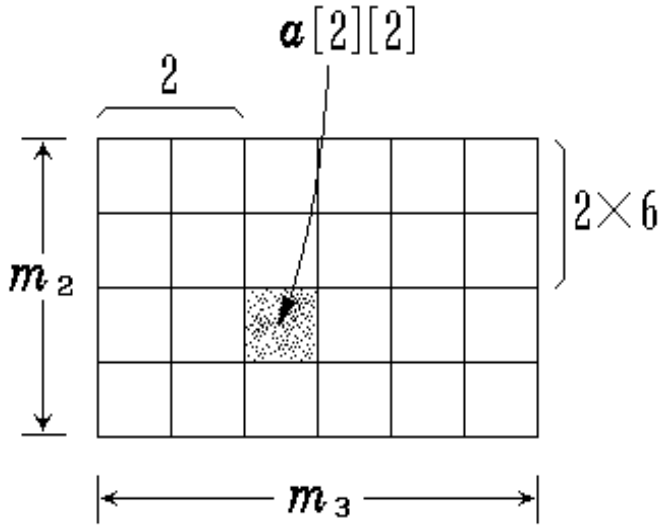


提要

- 数组的基本概念
- 一维数组
- **多维数组**

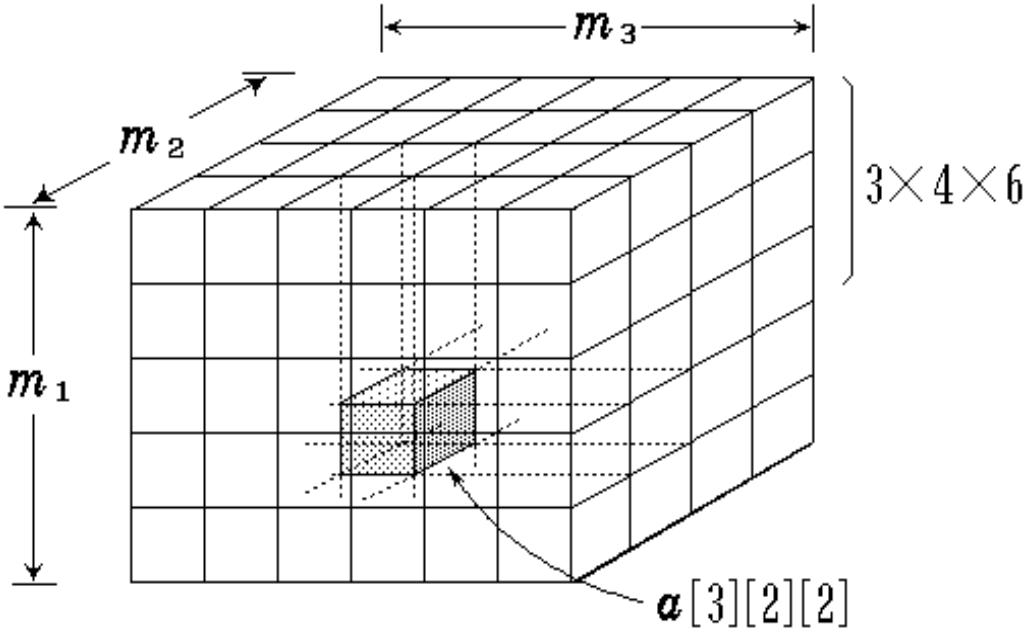
多维数组 – 示例

二维数组a[4][6]



行下标i； 列下标j

三维数组a[5][4][6]



页下标i; 行下标j; 列下标k

多维数组定义

- **二维数组的定义**形式为

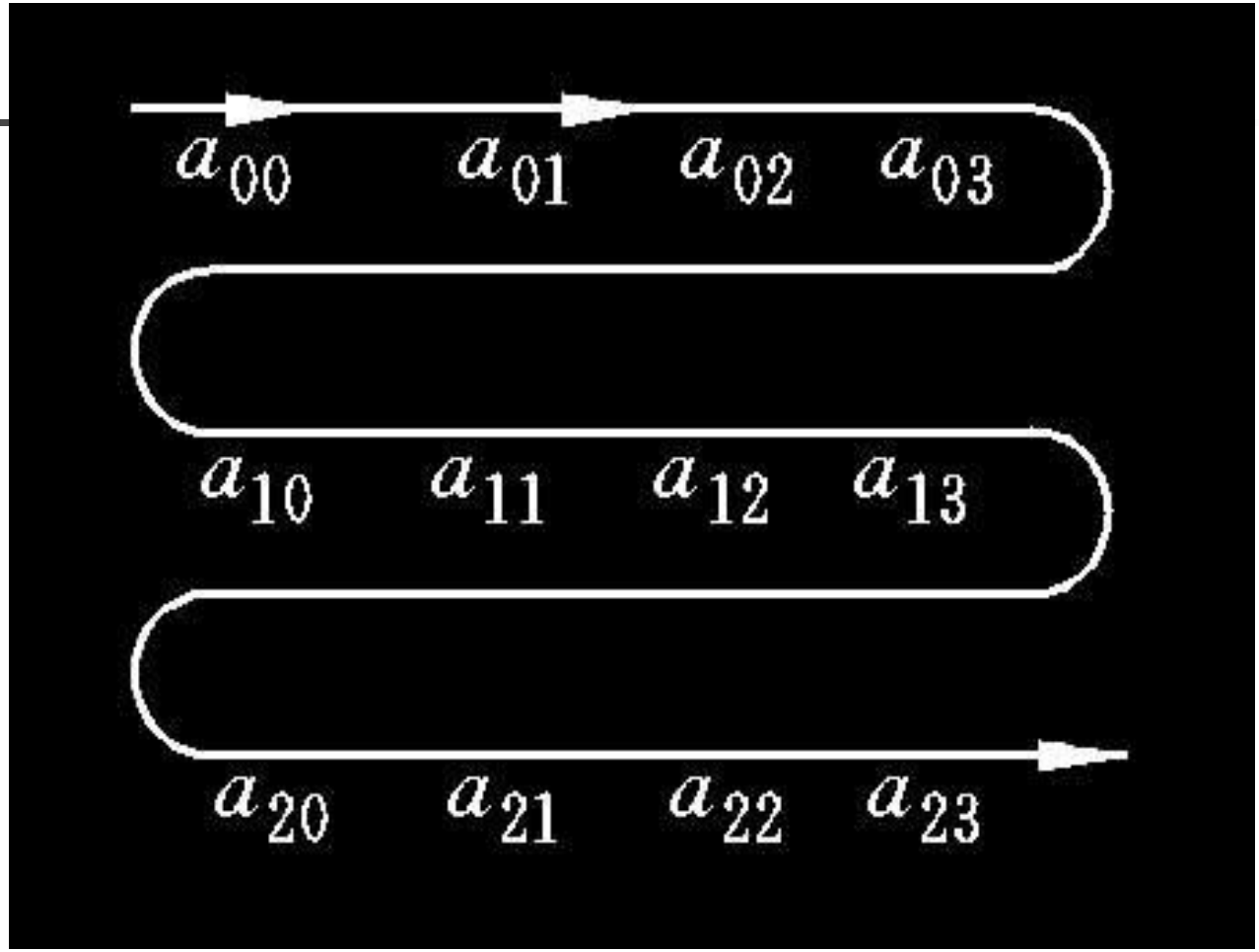
  类型说明符　数组名 [常量表达式][常量表达式];

  - 多维数组示例

    ```
    float a[2][3],b[3][4]; /* 两个二维数组 */
    float c[2][2][3];      /* 一个三维数组 */
    ```

  - 二维数组的元素的存放顺序是**按行存放**，即从数组首地址开始，先顺序存放第一行元素，再存放第二行元素
    - 例如，对于上面数组a[2][3]，其元素在内存中的存放顺序为：

      a[0][0]、a[0][1]、a[0][2]、a[1][0]、a[1][1]、a[1][2]


a_{00} a_{01} a_{02} a_{03}
a_{10} a_{11} a_{12} a_{13}
a_{20} a_{21} a_{22} a_{23}

多维数组存放特点

- 一个多维数组的元素在内存中的存放顺序有这样的特点：**第一维的下标变化最慢，最右边的下标变化最快**
  - 例如，对于前面例子中的三维数组c[2][2][3]，其元素的存放顺序为

    c[0][0][0]、c[0][0][1]、c[0][0][2]、c[0][1][0]、
    c[0][1][1]、c[0][1][2]、c[1][0][0]、c[1][0][1]、
    c[1][0][2]、c[1][1][0]、c[1][1][1]、c[1][1][2]。

多维数组元素引用

- 引用二维数组元素的表示形式为

  数组名[下标][下标]

- 在用下标引用数组的元素时，应该注意下标值的有效性，应在已定义的对应维大小的范围内，即大于等于0，和小于对应维的元素个数

多维数组初始化

- 按行给二维数组的全部元素赋初值

```
int a1[2][3]={ {1, 2, 3},{4, 5, 6}};
```

- 按元素存储顺序给数组元素赋初值

```
int a2[2][3] = {1, 2, 3, 4, 5, 6};
```

- 按行给数组的部分元素赋初值

```
int a3[2][3]={ {1, 2}, {0, 5}}; /*其余均为 0*/
```

- 按元素存储顺序给前面部分元素赋初值

```
int a4[2][3] = {1, 2, 3, 4}; /*其余均为 0*/
```

- 按元素存储顺序，给部分或全部元素赋初值，并且不指定第一维的元素个数

```
int a5[][3] = {1, 2, 3, 4, 5};/* 两行 */
```

- 用按行赋初值方法，对各行的部分或全部元素赋初值，并省略第一维的元素个数

```
int a6[][3] = {{0, 2}, {}};/*  两行 */
```

多维数组程序示例-0

- 二维数组元素按行输入和按行输出示意程序

```
#include <stdio.h>
int main()
{  int a[2][3], i, j;
   for(i=0; i < 2; i++)       /* 二维数组元素按行输入 */
      for(j = 0; j < 3; j++) {
         printf("Enter a[%d][%d] ", i, j);  scanf("%d", &a[i][j]);
      }
   for(i=0; i < 2; i++){      /* 二维数组元素按行输出 */
      for(j = 0; j < 3; j++)
         printf("%d\t", a[i][j]);
      printf("\n");  /* 按行换行 */
      }
}
```

多维数组程序示例-1

- 有一个3×4的矩阵，求出其中值最大的那个元素及其所在的行号和列号

```
#include<stdio.h>
int main()
{int i, j, row=0, colum=0, max;
 int a [3] [4] ={{1, 2, 3, 4}, {9, 8, 7, 6}, {-10, 10, -5, 2}};
     max=a [0] [0] ;
     for (i=0;i<=2;i++)
       for (j=0;j<=3;j++)
          if (a [i] [j] >max)
           {max=a [i] [j] ;
             row=i; colum=j;
           }

   printf("max=%d, row=%d, colum=%d\n", max, row, colum);
   return 0;
 }
```

输出结果为：　　max=10，row=2，colum=1

多维数组程序示例-2

- **把某月的第几天转换成年的第几天**
- 程序实例分析
  - 为确定一年中的第几天，需要一张每月的天数表，该表给出每个月份的天数。
  - 由于二月份天数因闰年（leap year）和非闰年有所不同，为程序处理方便，把月份天数表组织成一个二维数组
  - 闰年的精确计算
    - 普通年能被4整除且不能被100整除的为闰年。如：2004年就是闰年,1901年不是闰年
    - 世纪年能被400整除的是闰年，如：2000年是闰年，1900年不是闰年

多维数组程序示例-2(续)

- （续）把某月的第几天转换成年的第几天

```
#include <stdio.h>
int day_table[][12] =
{{31,28,31,30,31,30,31,31,30,31,30,31},
 {31,29,31,30,31,30,31,31,30,31,30,31}};
int main()
{  int year, month, day, leap, i;
   printf("Input year, month, day.\n");
   scanf("%d%d%d", &year, &month, &day);
   leap=year%4==0 && year%100 || year%400==0;
   for(i = 0; i < month-1; i++)
        day += day_table[leap][i];
   printf("\nThe days in year is %d.\n", day);
 }
```

多维数组程序示例-3

- 已知年份与年中的第几天，计算该天是这年的哪一月和哪一日

```
#include <stdio.h>
int day_table[][12] =
   {{31,28,31,30,31,30,31,31,30,31,30,31},
    {31,29,31,30,31,30,31,31,30,31,30,31}};
char monthname[][10] = {"January", "February", "March", "April",
"May",  "June",  "July",  "August",   "September",  "Octorber",
"November", "December"};
```

多维数组程序示例-3(续)

- （续）已知年份与年中的第几天，计算该天是这年的哪一月和哪一日

```
int main()
{  int year, day, leap, i;
   printf("Input year, day of year.\n");
   scanf("%d%d", &year, &day);
   leap=year%4==0 && year%100 || year%400==0;
   for(i = 0; day > day_table[leap][i]; i++)
        day -= day_table[leap][i];
   printf("The month is %s,the Day is %d.\n",monthname[i], day);
   return 0;
 }
```

多维数组程序示例-4

- 输入整数 n ，输出以下形式的二维数组

```
1    2    3    ...    n
n    1    2    ...    n-1
n-1  n    1    ...    n-2
          ...
2    3    4    ...    1
```

- 注：上面问题有多种解法

多维数组程序示例-4(续)

- **解法1：先给出第0行，然后利用i行与i-1行的关系，求出i行**

```
for(j = 0; j < n ; j++)    /* 先形成第0行 */
    a[0][j] = j+1;
for(i = 1; i < n; i++) {
    a[i][0]=a[i-1][n-1];  /* 生成i行的第0列 */
    for(j = 1; j < n; j++) /* 生成i行的其余列 */
        a[i][j] = a[i-1][j-1];
 }
```

多维数组程序示例-4(续)

- **解法2：** 将i行的内容分成两部分
- i行内容： n-i+1 ... n 1 ... n-i

```
for(i = 0; i < n; i++) {
    k = 1;
    for(j = i; j < n; j++)  /*右上部分*/
        a[i][j] = k++;
    for(j = 0; j < i; j++)  /*左下部分*/
        a[i][j] = k++;
}
```

多维数组程序示例-4(续)

- **解法3：** 将方法2中的变量k去掉，直接代入算式

```
for(i = 0; i < n; i++) {
    for(j = i; j < n; j++)  a[i][j] = j-i+1;
    for(j = 0; j < i; j++) a[i][j] = j-i+1+n;
 }
```

- **解法4：** 将方法3中的两个j循环合并

```
for(i = 0; i < n; i++)
   for(j = 0; j < n; j++)
      a[i][j] = j < i ? j-i+1+n : j-i+1 ;
```

多维数组程序示例-4(续)

- **解法5：** 将4中的条件表达式简化

```
for(i = 0; i < n; i++)
  for(j = 0; j < n; j++)
    a[i][j] = (j - i + n) % n + 1;
```

- **解法6：** 先给出第0行，再由i行与0行的关系求出i行

```
for(j = 0; j < n; j++)   a[0][j] = j + 1;
for(i = 1; i < n; i++)
  for(j = 0; j < n; j++)
    a[i][j] = a[0][(j - i + n) % n];
```

多维数组程序示例-4(续)

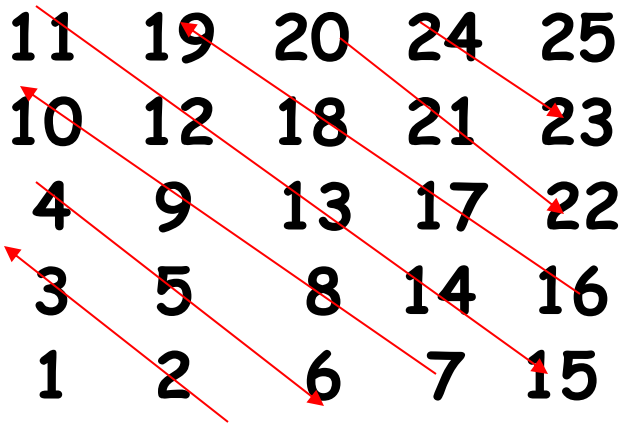
- **解法7：**对i行，从i列开始，按顺序将整数填入到数组中

```
for(i = 0; i < n; i++)
  for(j = i; j < i+n;j++)     /* i行从j列开始填n次 */
    a[i][j%n] = j-i+1;
```

- 还有许多填值的方法，请大家自已再想出方法，并写出C实现代码

多维数组程序示例-5

- 形成以下形式的二维数组(以n = 5为例)，并输出



- 程序实例分析
  - n*n的二维数组，共有2n-1条斜线，将它们顺次编号为1至2*n-1
  - 第奇数条斜线从左上往右下;第偶数条斜线从右下往左上

多维数组程序示例-5(续)

- 程序实例分析(续)
  - 程序按从左下角至右上角的顺序逐条斜线填数。斜线的填数方向按斜线的奇偶性交替变化
  - 每条斜线的行列起始位置的变化规律与斜线位于下三角或上三角有不同
    - **对于下三角，**第奇数条斜线从左上往右下，起始行号为n-d（d为斜线编号，下同），起始列号为0；第偶数条斜线从右下往左上，起始行号为n-1,起始列号为d-1
    - **对于上三角**，第奇数条斜线从上往下，起始行号为0，起始列号为d-n；第偶数条斜线从下往上，起始行号为2*n-1-d,起始列号为n-1

多维数组程序示例-5(续)

- 按以上情况分类，写出程序如下

```
#include  <stdio.h>
#define MAXN 10
int a[MAXN][MAXN];
int main()
{ int i, j, k, d, n;
  while (1) {  /* n 不超过10 */
     printf("Enter n   "); scanf("%d", &n);
     if (n >= 1 && n <= 10) break;
     printf("Error! n must be in [1, 10]\n");
   }
```

多维数组程序示例-5(续)

- 程序（续）

```
for(k = d = 1; d <= 2*n-1; d++) {
    if (d <= n-1)    /* 左下三角 */
      if (d % 2)     /* 奇数号斜线，从左上往右下 */
        for(i = n-d, j = 0; i < n; i++, j++)
          a[i][j] = k++;
      else    /* 偶数号斜线，从右下往左上 */
        for(i=n-1,j = d-1; i >= n-d; i--, j--)
          a[i][j] = k++;
```

多维数组程序示例-5(续)

- 程序（续）

```
    else   /* d >= n，右上三角 */
      if (d % 2)
        for(i=0,j=d-n; i<=2*n-1-d; i++, j++)
           a[i][j] = k++;
      else
         for(i=2*n-1-d,j=n-1; i>=0; i--, j--)
            a[i][j] = k++;
    }
  for(i = 0; i< n; i++) {
     for(j=0; j<n;j++)   printf("%4d", a[i][j]);
      printf("\n");
    }
}
```

本讲小结

- 数组的基本概念
- 一维数组
  - 定义、元素引用、初始化
- 多维数组
  - 二维、三维及多维
  - 定义、元素引用、初始化