计算机程序设计基础(1)
10 数组（下）

清华大学电子工程系
杨昉
E-mail: fangyang@tsinghua.edu.cn

上节内容

- 数组的基本概念
- 数组的定义与引用
  - 一维数组　　int a[10], b[ ]={1,2,3};
  - 二维数组　　int a[10][10], b[ ][3]={{1,2,3},{4},{5,6}};
  - 数组初始化：**赋值语句**、**输入函数**、**初始化赋值**
- 字符数组与字符串
  - 字符数组的定义与初始化　　char a[10];
  - 字符串、字符数组与字符串的输入与输出　　%c, %s
  - 字符串处理函数：puts, gets, strcat, strcpy, strncpy, strcmp, strlen, strlwr, strupr, sprintf, sscanf

课程回顾: 数组

类型	定义	说明
一维数组	一维数组:类型说明符 数组名[常量表达式] int z[] = { 0,0,0,0,0 }; int a[10] ={1,2,3,4,5}; static int y[5];	下标范围是0到N-1,只能逐个引用元素; 常量表达式必须为整型,不能是变量; 部分元素赋值后，后面自动赋初值0; 外部数组和静态数组赋初值0或给定值
二维数组	二维数组:类型说明符 数组名[常量表达式1][常量表达式2]; int c[][4]={0,1,2,3,4,5,6,7,8}; int d[][4]={{1,2},{5},{9,10,11}};	二维数组存储顺序是以行为主的; 分行赋初值，部分赋值后续自动为0; 全部或分行赋值可省略第一维长度说明
字符数组	char 数组名[常量表达式]; char 数组名[常量表达式1][常量表达式2];	字符数组中的一个元素只能存放一个字符，初始化与数组类似
字符串	char a[] = "how do you do?"; 错误：char b[15]; b= "China"; 错误：char b[15]; b[15]= "China";	一对双撇号括起来,隐含包括一个结束符'\0',字符串可以对字符数组进行初始化

本节目录

10.1 数组作为函数参数

- 形参数组与实参数组的结合
- 二维数组作为函数参数

10.2 算法举例

- 查找问题
- 排序问题
- 数值与下标的映射
- 有序表问题
- 数组的应用

数组作为函数参数

10.1 数组作为函数参数

- 形参数组与实参数组的结合
- 二维数组作为函数参数

数组作为函数参数


- C语言规定，**数组名**可以作为函数的形参
  - 基本形式：　　函数类型 函数名(数组类型 形参数组名)
  - 例：　int f(int a[ ]) {…}
- 例10-1：老师为十个小孩分糖果

小孩	1	2	3	4	5	6	7	8	9	10
糖果数	10	2	8	22	16	4	10	6	14	20

  - 调整糖果数：所有的小孩检查手中糖块数，若为**奇数**，则向老师再要一块，再同时将自己手中的糖分**一半**给下一个小孩
  - 多少次调整后，每个小孩糖块数都**相等**？每人各有多少块糖？

数组作为函数参数

- 例10-1：老师为十个小孩分糖果(辅助函数)
  - 函数flag：检查小孩的糖果数**是否相等**，相等返回0
  - 函数pr：输出**调整次数**及当前每个小孩的**糖果数**

```c
int flag(int a[], int n){
  int k;
  for (k = 1; k<n; k++)
    if (a[0] != a[k]) return 1;
  return 0;
}
```

```c
void pr(int k, int b[], int n){
  int j;
  printf(" %2d ", k);
  for (j = 0; j<n; j++)
    printf("%4d", b[j]);
  printf("\n");
}
```

数组作为函数参数


- 例10-1：老师为十个小孩分糖果(解题思路)
  - 主函数：**初始化**并输出每个小孩的糖果数。
  - 调用函数flag( )来检查每个小孩手中的糖果数是否**相等**，若不等则不断进行**调整**，直到每个小孩的糖果数相等：

① 每个小孩将自己的糖果分出一半给下一个小孩（若是**偶数**，直接分出一半；若是**奇数**，则向老师要一块后分出一半）

② 调整次数加1

③ 调用函数pr( )输出调整次数及当前每个小孩手中的糖果数

数组作为函数参数


```
#include <stdio.h>
void main() {
  int s[10] = { 10,2,8,22,16,4,10,6,14,20 }, k, t[10], n = 0, flag(int[], int);
  void pr(int, int[], int);
  printf("child    1   2   3   4   5   6   7   8   9  10\nround|\n");
  printf("-----+----------------------------------------\n");
  pr(n, s, 10);                                  /*开始时每个小孩手中的糖果数*/
  while (flag(s, 10)) {                          /*检查每个小孩糖果数是否相等,若不等则继续*/
    for (k = 0; k < 10; k++) {                   /*每个小孩将自己的糖果分出一半*/
      if (s[k] % 2 == 0) t[k] = s[k] / 2;/*若是偶数块,则直接分出一半*/
      else t[k] = (s[k] + 1) / 2;                /*不是偶数,向老师要一块后再分出一半*/
    }
    for (k = 0; k < 9; k++)/*每个小孩将分出的一半给下一个小孩*/
      s[k + 1] = t[k + 1] + t[k];
    s[0] = t[0] + t[9];      /*最后一个小孩将分出的一半给第一个小孩*/
    n = n + 1;               /*调整次数加1 */
    pr(n, s, 10);            /*调用函数pr( )输出调整次数以及当前每个小孩手中的糖果数*/
  }  }
```

数组作为函数参数

- 例10-1 老师为十个小孩分糖果(运行结果)

round \ child	1	2	3	4	5	6	7	8	9	10
0	10	2	8	22	16	4	10	6	14	20
1	15	6	5	15	19	10	7	8	10	17
2	17	11	6	11	18	15	9	8	9	14
3	16	15	9	9	15	17	13	9	9	12
4	14	16	13	10	13	17	16	12	10	11
5	13	15	15	12	12	16	17	14	11	11
6	13	15	16	14	12	14	17	16	13	12
7	13	15	16	15	13	13	16	17	15	13
8	14	15	16	16	15	14	15	17	17	15
9	15	15	16	16	16	15	15	17	18	17
10	17	16	16	16	16	16	16	17	18	18
11	18	17	16	16	16	16	16	17	18	18
12	18	18	17	16	16	16	16	17	18	18
13	18	18	18	17	16	16	16	17	18	18
14	18	18	18	18	17	16	16	17	18	18
15	18	18	18	18	18	17	16	17	18	18
16	18	18	18	18	18	18	17	17	18	18
17	18	18	18	18	18	18	18	18	18	18

请按任意键继续. . .

数组作为函数参数：形参与实参数组

- 形参数组与实参数组的结合
  - 调用函数与被调用函数中分别定义**实参和形参数组**，其数组名称可以不同，但**类型**必须一致
  - 形参**变量**与实参**变量**间采用**数值结合**，进行数据的**单向传递**
    - 如果在被调用函数中改变了形参的值是**不会**改变实参值的
  - 形参**数组**与实参**数组**间采用**地址结合**，实现数据的**双向传递**
    - 系统并不为形参数组**分配存储空间**，形参数组名中存放的是**实参数组首地址**
    - 被调用函数中改变了形参数组元素的值，实际就改变实参数组元素值

数组作为函数参数

●例10-2：函数中改变**数组值**与**变量值**

```
1  #include <stdio.h>
2  void exchange1(int a[], int b[]) {
3    for (int k = 0; k < 10; k++)  a[k] = b[k];
4  }
5  void exchange2(int a, int b) {
6    a = b;
7  }
8  void main() {
9    int a[10] = {0,0,0}; int a1 = 0, b1 = 233;
10   int b[10] = {1,2,3,4,5,6,7,8,9,10 };
11   exchange1(a, b);
12   exchange2(a1, b1);
13   for (int k = 0; k < 10; k++)
14     printf("%d ", a[k]);
15   printf("\na1 = %d , b1 = %d\n", a1, b1);
16 }
```

```
1 2 3 4 5 6 7 8 9 10
a1 = 0 , b1 = 233
请按任意键继续. . .
```

函数中进行改变后，数组a的值改变，但变量a1的值不变

数组作为函数参数

- 形参数组与实参数组的结合
  - 实参数组与形参数组的大小可以**一致**也可以**不一致**
  - C编译系统对形参数组的大小不作检查，调用时只将实参数组的**首地址**传给形参数组
  - 为了通用性，函数中的一维形参数组通常**不指定大小**，一般要在函数中另设一个传送形参**数组元素个数**的形参变量
  - 向前引用说明可以直接采用函数原型，也可省略形参变量，但**不能省略表示数组的[]**，例如 `int flag(int [], int)`

二维数组作为函数参数

- 被调用函数中对形参数组说明时
  - 可以**指定每一维**的大小

    例：int a[2][4], b[4][3], c[2][3];
  - 可以**省略第一维**的大小说明

    例：int a[ ][4], b[ ][3], c[ ][3];
- 定义形参数组时**不能省略第二维**的大小
  - 错误示例：int a[2][ ], b[4][ ], c[2][ ];
  - 错误示例：int a[ ][ ], b[ ][ ], c[ ][ ];

二维数组数组作为函数参数

- 例10-3：编写一个函数求两个矩阵的乘积矩阵
  - $C = AB$，　$C_{ij} = \sum_{t=1}^{n} A_{it}B_{tj}$
  - 外面两层循环遍历C_{ij}
  - 最内层循环计算C_{ij}

```
1  void matmul(int a[2][4],int b[4][3],int c[2][3],int m,int n,int k){
2    int i, j, t;
3    for (i = 0; i<m; i++)
4      for (j = 0; j<k; j++){
5        c[i][j] = 0;
6        for (t = 0; t<n; t++)
7          c[i][j] += a[i][t] * b[t][j];
8      }
9    return;
10 }
```

二维数组数组作为函数参数

●例10-3：编写一个函数求两个矩阵的乘积矩阵（代码）

```
1  #include <stdio.h>
2  void main(){
3    int i, j, c[2][3];
4    int a[2][4] = {1,2,3,4,5,6,7,8 };
5    int b[4][3] = {1,2,3,4,5,6,7,8,9,10,11,12 };
6    void matmul(int[][4],int[][3],int[][3],int,int,int);
7    matmul(a, b, c, 2, 4, 3);
8    for (i = 0; i<2; i++){
9      for (j = 0; j<3; j++)
10       printf("%5d", c[i][j]);
11     printf("\n");
12   }
13 }
```

```
   70   80   90
  158  184  210
请按任意键继续. . .
```

二维数组作为函数参数

- 被调用函数中，**不能用形参变量**定义二维数组各维大小
  - 错误示例：int a[m][n], b[n][k], c[m][k];
- 二维形参数组在**维度变化**情况下通用性不强。可实参是二维数组，而形参是一维数组，它们的结合还是**地址结合**
  - 可以将二维实参数组看成是一个**元素个数相同**的一维数组
- 二维数组作为形参时，可以转化为**一维数组**来处理
  - 如果二维数组（矩阵）的列数为n，则该二维数组中行标为i、列标为j的元素所对应的一维数组元素的下标为**i*n+j**

二维数组作为函数参数

- 例10-4：实参为二维数组，形参为一维数组
  - 二维数组中的元素**以行为主**存储
  - 实际引用时，将二维数组元素中的**行标与列标**转换成一维数组元素的**下标**，从而实现一维数组元素与二维数组元素对应

```
1  void matmul(int a[],int b[],int c[],int m,int n,int k){
2    int i, j, t;
3    for (i = 0; i < m; i++)
4      for (j = 0; j < k; j++){
5        c[i * k + j] = 0;
6        for (t = 0; t < n; t++)
7          c[i * k + j] += a[i * n + t] * b[t * k + j];
8      }
9    return;
10 }
```

二维数组作为函数参数


- 例10-4：实参为二维数组，形参为一维数组
  - 函数调用时需要**强制类型转换**(int *)，把二维数组强制转换为一维数组，否则编译时，编译系统会给出类型不一致的**警告**

```
1  #include <stdio.h>
2  void main(){
3    int i, j, c[2][3]; int a[2][4]={1,2,3,4,5,6,7,8};
4    int b[4][3] = { 1,2,3,4,5,6,7,8,9,10,11,12 };
5    void matmul(int a[],int b[],int c[],int,int,int);
6    matmul((int*)a, (int*)b, (int*)c, 2, 4, 3);
7    for (i = 0; i < 2; i++) {
8      for (j = 0; j < 3; j++)
9        printf("%5d", c[i][j]);
10     printf("\n");
11   }
12 }
```

```
   70   80   90
  158  184  210
请按任意键继续. . .
```

如果去掉int *

```
: warning C4047: “函数”: “int *”与“int [2][4]”的间接级别不同
 warning C4024: “matmul”: 形参和实参 1 的类型不同
: warning C4047: “函数”: “int *”与“int [4][3]”的间接级别不同
: warning C4024: “matmul”: 形参和实参 2 的类型不同
: warning C4047: “函数”: “int *”与“int [2][3]”的间接级别不同
: warning C4024: “matmul”: 形参和实参 3 的类型不同
```

二维数组作为函数参数


- 例10-5：二维数组(行、列数相同)每行元素的平均值
  - 定义函数avg：计算形参数组s中每一行元素的平均值，顺序存放在形参数组t中

```
1  void avg(int s[], int n, double t[]) {
2  /*计算形参数组s中每一行元素的平均值,顺序存放在形参数组t中*/
3    int i, j;
4    for (i = 0; i < n; i++){
5      t[i] = 0.0;
6      for (j = 0; j < n; j++)
7        t[i] += s[i * n + j];
8      t[i] /= n;
9    }
10 }
```

二维数组作为函数参数

●例10-5：二维数组每行元素的平均值(程序代码)

```
1  #include <stdio.h>
2  void main(){
3    int a[6][6], i, j;
4    void avg(int s[], int, double t[]);
5    double b[6];
6    printf("input MAT a : "); /* 输入前的提示*/
7    for (i = 0; i < 6; i++)
8      for (j = 0; j < 6; j++)
9        scanf("%d", &a[i][j]); /* 逐行输入二维数组a的各元素*/
10   avg((int*)a, 6, b); /* 计算二维数组a每一行元素的平均值,顺序存放在一维数组b中*/
11   for (i = 0; i < 6; i++){
12     for (j = 0; j < 6; j++) /* 输出数组a中的一行元素*/
13       printf("%7d", a[i][j]);
14     printf("%15e\n", b[i]); /* 输出数组a中一行元素的平均值*/
15   }
16 }
```

二维数组作为函数参数

- 例10-5：数组每行元素的平均值(运行结果)

```
input MAT a :
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 18 17 18
19 20 21 22 23 24
25 26 27 28 29 30
31 32 33 34 35 36
       1       2       3       4       5       6   3.500000e+00
       7       8       9      10      11      12   9.500000e+00
      13      14      15      18      17      18   1.583333e+01
      19      20      21      22      23      24   2.150000e+01
      25      26      27      28      29      30   2.750000e+01
      31      32      33      34      35      36   3.350000e+01
请按任意键继续. . .
```

本节目录

10.2 算法举例

- 查找问题
- 排序问题
- 数值与下标的映射
- 有序表问题
- 数组的应用

算法举例：查找问题

- 有序表：线性表中的元素按值**非递减排列**
- 二分查找
  - 适用于**顺序存储的有序表**
  - 设有序线性表的长度为n，被查元素为x，查找方法如下
  - 将x与线性表的**中间项**进行比较

1. 若**x=中间项的值**，则说明查到，查找结束
2. 若**x<中间项的值**，则在线性表**前半部分**以相同的方法查找
3. 若**x>中间项的值**，则在线性表**后半部分**以相同的方法查找

重复这个过程一直进行到**查找成功**或**子表长度为0**为止

算法举例：查找问题


- 例10-6：一般表的顺序查找
  - 无序的一般表，只能**顺序查找**
  - 这个过程一直进行到查找成功 或 全表都找过但没有找到
  - 找到的**平均查找次数**： (1+2+3+……+n)/n = (n+1)/2
  - 最坏情况下，找不到的查找次数：**n**

```
1  int search(ET v[], int n, ET x){
2    int i;
3    for (i = 0; i < n; i++)
4      if (v[i] == x) return i;
5    return(-1);
6  }
```

算法举例：查找问题


●例10-7：有序表的二分查找

- 函数返回被查找元素x在线性表中的序号，未查找到则返回-1
- 对长度为n的有序线性表，在最坏情况下，二分查找只需要比较$log_2 n$次，而顺序查找需要比较n次

```
1  int bisearch(ET v[], int n, ET x){
2    int i, j, k; i = 0; j = n - 1;
3    while (i <= j){
4      k = (i + j) / 2;
5      if (x == v[k]) return(k); //查找结束
6      if (x < v[k]) j = k - 1;
7      else i = k + 1;
8    }
9    return(-1);
10 }
```

- ET表示可以是任何数值类型标识符（char, short, int, long, float, double等），根据线性表中实际的元素类型来确定
- 如果线性表为整型，则应为int；如果线性表为实型，则应为float或double

课堂练习

练习10-1：有序数组v长度n=12，其元素的值如下表；在其中查找x=64，请写出调用函数bisearch的输出结果

下标	0	1	2	3	4	5	6	7	8	9	10	11
v	5	13	19	21	37	56	60	64	75	80	88	92

```
1  int bisearch(int v[], int n, int x){
2    int i, j, k; i = 0; j = n - 1;
3    while (i <= j){
4      k = (i + j) / 2;
5      printf("i = %d, j = %d, k = %d\n", i, j, k);
6      if (x == v[k]) return(k); //查找结束
7      if (x < v[k]) j = k - 1;
8      else i = k + 1;
9    }
10   return(-1);
11 }
```

```
i = 0, j = 11, k = 5
i = 6, j = 11, k = 8
i = 6, j = 7, k = 6
i = 7, j = 7, k = 7
请按任意键继续. . .
```

算法举例：排序问题

●双向冒泡排序的基本过程

1．从**表头开始往后**扫描线性表，在扫描过程中，**逐次比较相邻两个元素**的大小。若相邻两个元素中，前面的元素**大于**后面的元素，则将它们互换

□称之为消去一个**逆序**

□在扫描过程中，不断将两相邻元素中的大者往后移动，最后就将线性表中的**最大者换到了表的最后**

算法举例：排序问题

●双向冒泡排序的基本过程

2．从**后到前**扫描剩下的线性表，在扫描过程中**逐次比较相邻两个元素**的大小。若相邻两个元素中，后面的元素**小于**前面的元素，则将它们互换

□消去一个**逆序**

□在扫描过程中，不断将两相邻元素中的小者往前移动，最后就将剩下线性表中**最小者换到了表的最前**

算法举例：排序问题

●双向冒泡排序的基本过程

3．对剩下的线性表**重复上述过程**，直到剩下的线性表**变空**为止，此时的线性表已经变为有序

□冒泡排序举例

原序列	5 1 7 3 1 6 9 4 2 8 6
第1遍(**从前往后**)	5↔1 7↔3↔1↔6 9↔4↔2↔8↔6
结果	1 5 3 1 6 7 4 2 8 6 9
(**从后往前**)	1 5↔3↔1 6↔7↔4↔2 8↔6 9
结果	1 1 5 3 2 6 7 4 6 8 9

算法举例：排序问题

- 双向冒泡排序的基本过程如下
  - 冒泡排序举例

第2遍(从前往后)	1 1 5↔3↔2 6 7↔4↔6 8 9
结果	1 1 3 2 5 6 4 6 7 8 9
(从后往前)	1 1 3↔2 5↔6↔4 6 7 8 9
结果	1 1 2 3 4 5 6 6 7 8 9
第3遍(从前往后)	1 1 2 3 4 5 6 6 7 8 9
最后结果	1 1 2 3 4 5 6 6 7 8 9

算法举例：排序问题

●例10-8：双向冒泡排序

- 假设线性表的长度为n
- 在最坏情况下，冒泡排序需要经过n/2遍的从前往后扫描，n/2遍从后往前的扫描
- 每次扫描长度递减，需要的比较次数为n(n-1)/2

```
1  void bubsort(ET p[], int n){
2    int m, k, j, i;
3    ET d;
4    k = 0; m = n - 1;
5    while (k < m) {                  /*子表未空*/
6      j = m - 1; m = 0;              /*m记录最后交换的位置*/
7      for (i = k; i <= j; i++)/*从前往后扫描子表*/
8        if (p[i] > p[i + 1]) {/*发现逆序进行交换*/
9          d=p[i]; p[i]=p[i+1]; p[i+1]=d; m=i;
10       }
11     j = k + 1; k = 0;              /*k记录最前交换的位置*/
12     for (i = m; i >= j; i--)/*从后往前扫描子表*/
13       if (p[i - 1] > p[i]) {/*发现逆序进行交换*/
14         d=p[i]; p[i]=p[i-1]; p[i-1]=d; k=i;
15       }
16   }
17   return;
18 }
```

算法举例：排序问题

- 选择排序
  - 扫描整个线性表，从中选出最小元素，将它交换到表的最前面
  - 然后对剩下的子表采用同样的方法，直到子表空为止
  - 对于长度为n的序列，选择排序需要扫描n-1遍
  - 选择排序举例：

原序列	89 21 56 48 85 16 19 47	第4遍选择	16 19 21 **47** 85 89 56 **48**
第1遍选择	**16** 21 56 48 85 **89** 19 47	第5遍选择	16 19 21 47 **48** 89 56 **85**
第2遍选择	16 **19** 56 48 85 89 **21** 47	第6遍选择	16 19 21 47 48 **56** **89** 85
第3遍选择	16 19 **21** 48 85 89 **56** 47	第7遍选择	16 19 21 47 48 56 **85** **89**

算法举例：排序问题

●例10-9：选择排序程序举例

```
1  void selesort(ET p[], int n){
2    int i, j, k;
3    ET d;
4    for (i = 0; i <= n - 2; i++){
5      k = i;
6      for (j = i + 1; j <= n - 1; j++)
7        if (p[j] < p[k]) k = j;
8        if (k != i){ //交换元素位置
9          d = p[i]; p[i] = p[k]; p[k] = d;
10       }
11   }
12   return;
13 }
```

- 假设线性表的长度为n
- 需要的比较总次数为(n-1)+(n-2)+……+1 = n(n-1)/2次

算法举例：排序问题

- 插入排序
  - 将第j个元素 (j=1,2,3,…) 赋值给变量T
  - 从有序子表的最后一个元素（即线性表中第j-1个元素）开始，往前逐个与T进行比较，将大于T的元素均依次向后移动一个位置
  - 直到发现一个元素不大于T为止，此时就将T插入到刚移出的空位置上，有序子表的长度就变为j了
  - 逐渐增加j直到有序表全部排序完成

算法举例：插入排序

- 插入排序例

5 1 7 3 1 6 9 4 2 8 6
↑ j=2

1 5 7 3 1 6 9 4 2 8 6
↑ j=3

1 5 7 3 1 6 9 4 2 8 6
↑ j=4

1 3 5 7 1 6 9 4 2 8 6
↑ j=5

1 1 3 5 7 6 9 4 2 8 6
↑ j=6

1 1 3 5 6 7 9 4 2 8 6
↑ j=7

1 1 3 5 6 7 9 4 2 8 6
↑ j=8

1 1 3 4 5 6 7 9 2 8 6
↑ j=9

1 1 2 3 4 5 6 7 9 8 6
↑ j=10

1 1 2 3 4 5 6 7 8 9 6
↑ j=11

1 1 2 3 4 5 6 6 7 8 9

算法举例：排序问题

●例10-10：插入排序

```
1  void insort(ET p[], int n){
2    int j, k;
3    ET t;
4    for (j = 1; j < n; j++){
5      t = p[j]; k = j - 1;
6      //向后移动有序子表中大于t的元素
7      while ((k>=0) && (p[k]>t)){
8        p[k+1] = p[k]; k=k-1;
9      }
10     p[k + 1] = t;
11   }
12   return;
13 }
```

- 假设线性表的长度为n
- 在**最好**情况下，需比较n-1次
- 在**最坏**情况下，需比较
  1+2+3+……+n-1 = n(n-1)/2次

算法举例：数值与数组下标的映射

- 例10-11：统计各年龄段人数
  - 人口普查时,需要统计各个年龄段的人数，共分为11个年龄段：0~9岁，10~19岁，20~29岁，30~39岁，40~49岁，50~59岁，60~69岁，70~79岁，80~89岁，90~99岁，100岁以上
  - 现有n个人的年龄在a数组中，请编程**统计各年龄段的人数**，统计结果存入数组c[11]

算法举例：数值与数组下标的映射


- 例10-11：统计各年龄段人数
  - 写法1：多个if语句

```
1  #include <stdio.h>
2  void count(int a[], int n, int c[]){
3    int i = 0;
4    for (i = 0; i < 11; i++) c[i] = 0;
5    for (i = 0; i < n; i++){
6      if (a[i] >= 0 && a[i] <= 9) c[0]++;
7      if (a[i] >= 10 && a[i] <= 19) c[1]++;
8      if (a[i] >= 20 && a[i] <= 29) c[2]++;
9      if (a[i] >= 30 && a[i] <= 39) c[3]++;
10     if (a[i] >= 40 && a[i] <= 49) c[4]++;
11     if (a[i] >= 50 && a[i] <= 59) c[5]++;
12     if (a[i] >= 60 && a[i] <= 69) c[6]++;
13     if (a[i] >= 70 && a[i] <= 79) c[7]++;
14     if (a[i] >= 80 && a[i] <= 89) c[8]++;
15     if (a[i] >= 90 && a[i] <= 99) c[9]++;
16     if (a[i] >= 100) c[10]++;
17   }
18 }
```

算法举例：数值与数组下标的映射

- 例10-11：统计各年龄段人数
  - 写法2：if…else if…
  - 还可考虑switch结构

```
1  #include <stdio.h>
2  void count2(int a[], int n, int c[]){
3    int i = 0;
4    for (i = 0; i < 11; i++) c[i] = 0;
5    for (i = 0; i < n; i++){
6      if (a[i] <= 9) c[0]++;
7      else if (a[i] <= 19) c[1]++;
8      else if (a[i] <= 29) c[2]++;
9      else if (a[i] <= 39) c[3]++;
10     else if (a[i] <= 49) c[4]++;
11     else if (a[i] <= 59) c[5]++;
12     else if (a[i] <= 69) c[6]++;
13     else if (a[i] <= 79) c[7]++;
14     else if (a[i] <= 89) c[8]++;
15     else if (a[i] <= 99) c[9]++;
16     else if (a[i] >= 100) c[10]++;
17   }
18 }
```

算法举例：数值与数组下标的映射


- 例10-11：统计各年龄段人数
  - 写法3：改变if顺序

```
1  void count3(int a[], int n, int c[]){
2    int i = 0;  for (i = 0; i < 11; i++) c[i] = 0;
3    for (i = 0; i < n; i++){
4      if (a[i] <= 29)       {
5        if (a[i] <= 9) c[0]++;
6        else if (a[i] <= 19) c[1]++;
7        else c[2]++;          }
8      else if (a[i] <= 59) {
9        if (a[i] <= 39) c[3]++;
10       else if (a[i] <= 49) c[4]++;
11       else c[5]++;          }
12     else if (a[i] <= 89) {
13       if (a[i] <= 69) c[6]++;
14       else if (a[i] <= 79) c[7]++;
15       else c[8]++;          }
16     else if (a[i] <= 99) c[9]++;
17     else c[10]++;
18 } }
```

算法举例：数值与数组下标的映射


●例10-11：统计各年龄段人数

□写法4：寻找数值与数组元素下标的**映射关系**

```
1  void count4(int a[], int n, int c[]) {
2    int i = 0, p;
3    for (i = 0; i < 11; i++) c[i] = 0;
4    for (i = 0; i < n; i++){
5      p = a[i] / 10; /*数值与数组元素下标的映射*/
6      if (p > 10)
7        p = 10;
8      c[p]++; /* 相应年龄段的计数器加1 */
9    }
10 }
```

算法举例：数值与数组下标的映射


●例10-12：统计其中字符串中每个字母出现的次数

□统计时，字母不区分大小写

```
1  void count2(char a[], int c[]) {
2    int i = 0, p, n = strlen(a);
3    for (i = 0; i < 26; i++) c[i] = 0;
4    for (i = 0; i < n; i++){
5      if (a[i] >= 'a' && a[i] <= 'z' )
6        p = a[i] - 'a';        /*数值与数组元素下标的映射*/
7      else if (a[i] >= 'A' && a[i] <= 'Z' )
8        p = a[i] - 'A';        /*数值与数组元素下标的映射*/
9      else continue;           /*不是字母，不统计*/
10     c[p]++;                  /*相应字母的计数器加1 */
11   }
12 }
```

课堂练习

●练习10-2：数组v长度n=12，请写出桶排序函数

□ “**桶排序**”的基本思想：将无序的数组元素分到**有限数量的桶**里，每个桶内部再分别排序，最后依次把各个桶中的元素列出来，得到**有序数组**

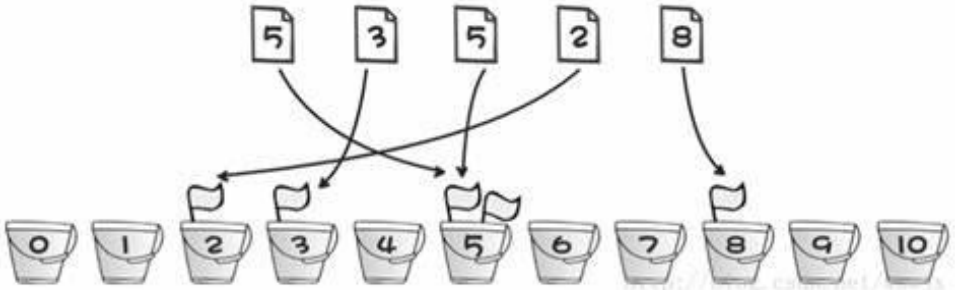
下标	0	1	2	3	4	5
v	21	13	56	60	92	80
下标	6	7	8	9	10	11
v	37	19	60	75	5	88

```
 5 13 19 21 37 56 60 60 75 80 88 92
请按任意键继续. . .
```

如何实现?

课堂练习

练习10-2：“**数值与数组下标的映射**”方法不仅可以用于统计数组中的元素出现次数，而且可以用来排序！




```c
void bucket_sort(int v[], int n){
  static int bucket[100];
  int i, j, k;
  for (i = 0; i < n; i++){
    if (v[i] >= 0 && v[i] < 100)
      bucket[v[i]]++;
  }
  for (j = 0; j < 100; j++){
    for (k = 0; k < bucket[j]; k++)
      printf("%3d", j);
  }
  printf("\n");
}
```

算法举例：有序表的插入

●例10-13：有序表中数组元素**插入**

□方法1：先找到合适位置，向后移动元素，然后插入元素

```
1  void insert1(ET v[], int n, ET x){
2    int i, j;
3    for (i = 0; i < n; i++)
4      if (v[i] < x) break;
5    if (i < n)
6      for (j = n - 1; j >= i; j--)
7        v[j + 1] = v[j]; //从最后一个元素开始,向后移动元素,腾出位置
8    v[i] = x;
9  }
```

算法举例：有序表的插入

- 例10-14：有序表中数组元素**插入**
  - 方法2：边找合适的位置**边向后**移动元素，然后插入元素

```
1  void insert2(ET v[], int n, ET x){
2    int j;
3    for (j = n - 1; j >= 0 && x < v[j]; j--)
4      v[j + 1] = v[j]; /* 从最后一个元素开始,向后移动元素,腾出位置*/
5    v[j + 1] = x;
6  }
```

算法举例：有序表的删除

- 例10-15：有序表中数组元素**删除**
  - 先找到合适的位置，向前移动元素

```
1  void delete(ET v[], int n, ET x){
2    int i, j;
3    for (i = 0; i < n; i++)
4      if (v[i] == x) break;
5    if (i < n - 1)
6      for (j = i; j < n - 1; j++)
7        v[j] = v[j + 1]; /* 从后一个元素开始,向前移动元素,补充空出的位置*/
8  }
```

算法举例：数组应用


- 例10-16：用**递归方式**在数组中找**最大值**
  - 递归思想：如果能在前n-1个元素中找到最大值t，则只需在t和a[n-1]中求最大值返回即可

```
1  ET FindMax(ET a[], int n){
2    ET t;
3    if (n > 1){
4      t = FindMax(a, n - 1);
5      return t > a[n - 1] ? t : a[n - 1];
6    }
7    else return a[0];
8  }
```

算法举例：数组应用

●例10-17：用**递归方式**进行数组元素的**排序**

□递归思想：如果前n-1个元素能先排好序，则只需在把最后一个元素a[n-1]插入到**长度为n-1的a数组**中即可

```
1  void RecurSort(ET a[], int n){
2    if (n > 1){
3      RecurSort(a, n - 1);
4      insert2(a, n - 1, a[n - 1]);
5    }
6  }
```

算法举例：数组应用


●例10-17：用**递归方式**进行数组元素的**排序**(程序代码)

```
1  #include <stdio.h>
2  void insert2(int v[], int n, int x){
3    int j;
4    for (j=n-1; j>=0 && x<v[j]; j--)
5      v[j + 1] = v[j];
6    v[j + 1] = x;
7  }
8  void RecurSort(int a[], int n){
9    if (n > 1){
10     RecurSort(a, n - 1);
11     insert2(a, n - 1, a[n - 1]);
12   }
13 }
```

```
14 void main(){
15   int a[10]={2,1,10,9,6,7,3,4,8,5},i;
16   for (i = 0; i < 10; i++)
17     printf("%d ", a[i]);
18   printf("\n");
19   RecurSort(a, 10);
20   for (i = 0; i < 10; i++)
21     printf("%d ", a[i]);
22   printf("\n");
23 }
```

```
2 1 10 9 6 7 3 4 8 5
1 2 3 4 5 6 7 8 9 10
请按任意键继续. . .
```

算法举例：数组应用(超长整数计算)

- 例10-18：计算m!，要求结果精确到个位
  - 用a[0],a[1]…… 分别表示超长整数的个位、十位……
  - 定义： a[N-1],a[N-2],…… a[n],a[n-1], ……a[2],a[1],a[0]
  - 初值： a[0]=1;
  - 外循环： a[0]=a[0]*k; k= 2,3,……m
  - 内循环： a[n] = a[n]*k + a[n-1]/10; a[n-1] %= 10;

    n=1,2,……N

算法举例：数组应用(超长整数计算)


- 例10-18：计算m!，要求结果精确到个位(程序代码)

```
#include <stdio.h>
#define N 10000
void main(){
   int a[N] = { 1 }; /* a[0]=1 */
   int n, k, m;
   scanf("%d", &m);
   for (k = 2; k <= m; k++){
      a[0] = a[0] * k;
      for (n = 1; n < N; n++){
         a[n] = a[n] * k + a[n - 1]/10;
         a[n - 1] %= 10;
      }
   }
   n = N - 1;
   while (a[n] == 0)
      n--; /*从最高位找第一个非零数*/
   printf("%d! = ", m);
   for (; n >= 0; n--)
      printf("%d", a[n]);
   printf("\n");
}
```

算法举例：数组应用(超长整数计算)

- 例10-18：**计算m!**，要求结果精确到个位(计算结果)

```
50
50! = 30414093201713378043612608166064768844377641568960512000000000000
请按任意键继续. . .
```

```
100
100! =
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000
请按任意键继续. . .
```

算法举例：数组应用(黑色星期五)

- 例10-19：黑色星期五问题
  - 如果某个月的**13号**正好是**星期五**，则被称为“黑色星期五”
  - 源于西方的宗教信仰：耶稣基督死在星期五，而13是不吉利的数字，两者的结合令人相信当天会发生不幸的事情
  - 问题1：统计出在某个年份中，出现了**多少次**“黑色星期五”，并给出这一年出现“黑色星期五”的**月份**
  - 问题2：探究自公元1年以来，一年中**最多**会出现几次黑色星期五、**最少**会出现几次黑色星期五，是否可能某一年**没有**黑色星期五?

算法举例：数组应用(黑色星期五)

- 例10-19：黑色星期五问题一(问题分析)
  - 问题1：统计出在某个年份中，出现了**多少次**“黑色星期五”，并给出这一年出现“黑色星期五”的**月份**
  - 分析：若年份为n，首先计算**前n-1年**有多少天days：

    闰年数：leap = (n - 1) / 4 - (n - 1) / 100 + (n - 1) / 400;

    天数：　days = (n - 1) * 365 + leap;
  - 用数组存储**每月前一个月**的天数：

```
int month[12]={0,31,28,31,30,31,30,31,31,30,31,30};
```

算法举例：数组应用(黑色星期五)

- 例10-19：黑色星期五问题一(问题分析)
  - 若当年为**闰年**，则二月份多一天：

```
if (n%4==0 && n%100 !=0 || n%400==0)
    month[2] +=1;
```

  - 设一个**计数器**mm统计黑色星期五个数
  - 循环计算一年12个月每月的13号是不是黑色星期五，并用数组c存月份

算法举例：数组应用(黑色星期五)

●例10-19：黑色星期五问题一(程序代码)

```
#include <stdio.h>
int fun(int n, int c[]){
  int i, mm = 0, leap, days;
  int month[12]={0,31,28,31,30,31,30,31,31,30,31,30};
  leap = (n-1)/4-(n-1)/100+(n-1)/400;
  days = 365 * (n - 1) + leap;
  if (n%4 == 0 && n%100 != 0 || n%400 == 0)
    month[2] += 1;
  for (i = 1; i <= 12; i++){
    days += month[i - 1];
    if ((days + 13) % 7 == 5)
      c[mm++] = i;
  }
  return mm;
}
```

```
void main(){
  int n, c[12], m = 0, i;
  printf("Please input a year:");
  scanf("%d", &n);
  m = fun(n, c);
  printf("%d年黑五个数:%d\n",n,m);
  printf("黑五的月份是: \n");
  for (i = 0; i < m; i++)
    printf("%d\n", c[i]);
}
```

算法举例：数组应用(黑色星期五)

- 例10-19：黑色星期五问题一(运行结果)

```
Please input a year:2019
2019年黑五的个数是:2
黑五的月份是:
9
12
请按任意键继续. . .
```

```
Please input a year:2020
2020年黑五的个数是:2
黑五的月份是:
3
11
请按任意键继续. . .
```

```
Please input a year:2021
2021年黑五的个数是:1
黑五的月份是:
8
请按任意键继续. . .
```

```
Please input a year:2022
2022年黑五的个数是:1
黑五的月份是:
5
请按任意键继续. . .
```

算法举例：数组应用(黑色星期五)

●例10-19：黑色星期五问题二(问题分析)

2. 探究自公元1年以来，一年中**最多**会出现几次黑色星期五、**最少**会出现几次黑色星期五，是否可能某一年**没有**黑色星期五?

□设数组int dd[13]={0};

□利用前面的fun函数，循环计算自公元1年到2021年**每年出现黑色星期五个数**m，并做统计：

```
dd[m]++;
```

□最后dd[i]就是出现i个黑色星期五的个数

算法举例：数组应用(黑色星期五)

●例10-19：黑色星期五问题二(程序代码)

```c
#include <stdio.h>
int fun(int n, int c[]){
  int i, mm = 0, leap, days;
  int month[12]={0,31,28,31,30,31,30,31,31,30,31,30};
  leap = (n-1)/4-(n-1)/100+(n-1)/400;
  days = 365 * (n - 1) + leap;
  if (n%4==0 && n%100!=0 || n%400==0)
    month[2] += 1;
  for (i = 1; i <= 12; i++){
    days += month[i - 1];
    if ((days + 13) % 7 == 5){
      c[i - 1]++; mm++;
    }
  }
  return mm;
}
```

```c
void main(){
  int c[12]={0},dd[13]={0},m,i;
  for (i = 1; i <= 2021; i++){
    m = fun(i, c);
    dd[m]++;
  }
  for (i = 0; i <= 12; i++)
    printf("%d,", dd[i]);
  printf("\n");
  for (i = 0; i < 12; i++)
    printf("%d,", c[i]);
}
```

算法举例：数组应用(黑色星期五)

- 例10-19：黑色星期五问题二(运行结果)
  - 输出第一行第m个数值表示出现黑色星期五m-1次的年份总数
  - 输出第二行第k个数值表示第k月出现黑色星期五的年份总数

```
0,863,860,298,0,0,0,0,0,0,0,0,0,
293,288,293,294,293,288,294,283,288,282,293,288,
请按任意键继续. . .
```

结论：每年最多3个，最少1个黑色星期五

算法举例：数组应用(报数问题)

- 例10-20：n人报数问题
  - 有n个人围成一圈，**按顺序编号**
  - 从第一个人开始报数（1~k报数），凡报到k的人**退出**圈子
  - 问最后留下的人原来排在第几号?
  - 例：10人围成一圈，1~2报数

1	2	3	4	5	6	7	8	9	10
1		3		5		7		9	
1				5				9	
				5					

算法举例：数组应用(报数问题)


●例10-20：

n人报数问题

```
总人数=10
报数总数=2
最后剩下的人编号为：5
请按任意键继续...
```

```
总人数=10
报数总数=3
最后剩下的人编号为：4
请按任意键继续...
```

```
1  #include <stdio.h>
2  void main(){
3    int a[100], n, num, m = 0, i = 0, k = 0, j = 0;
4    //n表示一共多少人,num表示报数上限
5    //m表示被淘汰的人数，k用来表示当前报数
6    printf("总人数=");      scanf("%d", &n);
7    printf("报数总数=");    scanf("%d", &num);
8    for (j = 0; j < n; j++)  a[j] = j+1;
9    while (m < n - 1){
10     if (a[i] != 0) k++; //未淘汰人进行报数
11     if (k == num)          //淘汰报数为num的人
12       {a[i] = 0; k = 0; m++;}
13     i = (i + 1) % n;      //对下一个人进行判断
14   }
15   for (j = 0; j < n; j++)
16     if (a[j]!=0) printf("最后剩下的人编号为：%d",j+1);
17 }
```

算法举例：数组应用(回文数问题)


- 例10-21：判断一个整数是否为回文数
  - 回文数：数字左右对称的整数
  - 例：121，12345654321是回文数
  - 分析：采用**数组**存储整数各位上的数字，判断左右对称位置上的数字是否相等
  - 如何取整数n每一位上的数字a?

    循环多次取余数和商　　a=n%10;　　n=n/10;

算法举例：数组应用(回文数问题)


●例10-21：
回文数问题

```
请输入数字:
49843164
非回文数
请按任意键继续...
```

```
请输入数字:
1234321
是回文数
请按任意键继续...
```

```
1  #include<stdio.h>
2  void main(){
3    int n = 0, flag = 1, num = 0;
4    printf("请输入数字: \n");
5    scanf("%d", &n);
6    int a[100], i = 0;
7    while (n){
8      a[num] = n % 10;
9      num++;  n /= 10;
10   }
11   for (int b = 0; b < num / 2; b++)
12     if (a[0 + b] != a[num - b - 1]) {
13       printf("非回文数");
14       flag = 0; break;
15     }
16   if (flag == 1) printf("是回文数");
17 }
```

本节总结

- 数组作为函数参数
  - 形参数组与实参数组的结合：**地址结合**
  - 形参数组与实参数组**类型一致**
  - 二维数组作为函数参数，形参可**省略第一维大小**
  - 二维数组作为形参时，可以转化为一维数组来处理

1	2	3
4	5	6
7	8	9

1	2	3	4	5	6	7	8	9

数组总结

- 算法举例
  - **查找**问题
    - 有序表**二分**查找、一般表**顺序**查找
  - **排序**问题
    - **冒泡**排序、**选择**排序、**插入**排序
  - 数值与下标的映射
    - 寻找数值与数组下标的**映射关系**
  - **有序表**的插入、删除
  - 数组的操作与应用

本节作业

- 作业10
  - 课本第九章习题5,11
  - 课本第九章习题14（上机完成，要求写实验报告）
  - 完成后将**word文档或拍照**提交到网络学堂

附加作业

- 第K大元素
  - 请编写C函数，对于输入的长度为n的一维数组a[]，求出其元素中第K大的值，作为函数的返回值
  - 注意：请比较不同实现方式的计算复杂度，尝试分析何种方法的查找效率最高
- 数组元素替换
  - 请编写C函数，对于输入的长度为n的一维数组a[]，请你将数组中的每个元素替换为排序后的序号
  - 例如：输入为arr = [40,10,20,30]时，输出值为[4,1,2,3]，表示原数组排序后的序号
- 最长升高子序列
  - 请编写C函数，对于输入的长度为n的一维数组a[]，请你求出所有子字符串中，最长的升高序列的长度。例如，[32，10，5，67，89，92，7]数组中，最长升高序列为[5，67，89，92]，其长度为4
  - 注意：升高指序列元素保持不变或者变大


THANKS