清華大學
Tsinghua University

程序设计基础

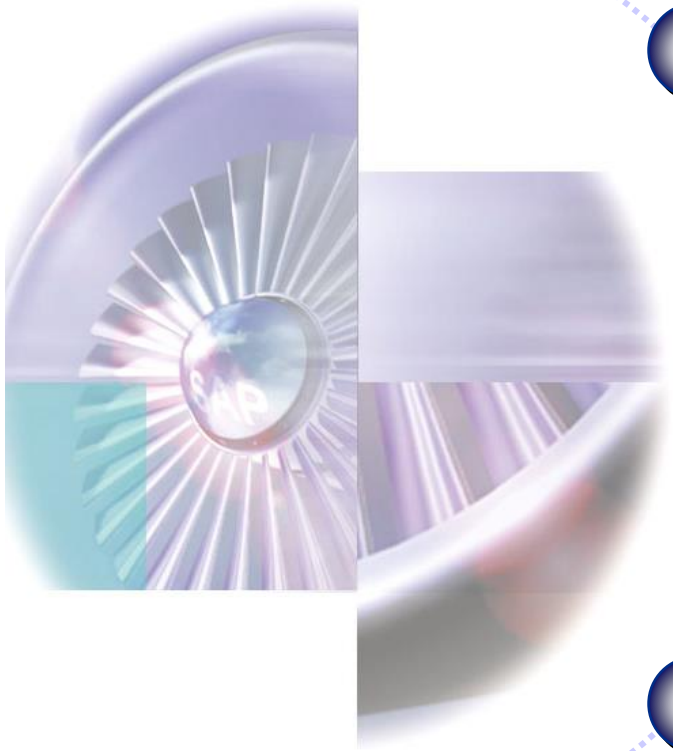
Fundamental of Programming

清华大学软件学院

刘玉身

liuyushen@tsinghua.edu.cn

Lecture 5: 数组



1. 一维数组的定义和引用
2. 二维数组的定义和引用
3. 字符数组
4. 程序举例

示例1: 字符串拼接

问题描述：

接受键盘输入的两个字符串，并将其首尾相接后输出。每个字符串内部不含空格，两个字符串之间以空白符分隔。

问题分析：

（1）数据结构

字符串的存储需要用字符数组

（2）算法要点

字符串拼接方法：先找到第一个字符串的末尾，然后将第二个串的字符逐个添加到末尾。

注意，要去掉第一个串的结束符‘\0’。

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char str1[50], str2[20];
    int i, j;
    printf("输入字符串1: ");
    gets(str1);
    printf("输入字符串2: ");
    gets(str2);
    i = 0;
    while(str1[i] != '\0') i++;
    str1[i] = ' ';
    i++;
    for(j = 0; str2[j] != '\0'; i++,j++)
        str1[i] = str2[j];
    str1[i] = '\0';
    printf("拼接结果: %s\n",str1);
    return 0;
}
```

输入字符串1：hello

输入字符串2：world!

拼接结果：hello world!

示例2: 回文问题

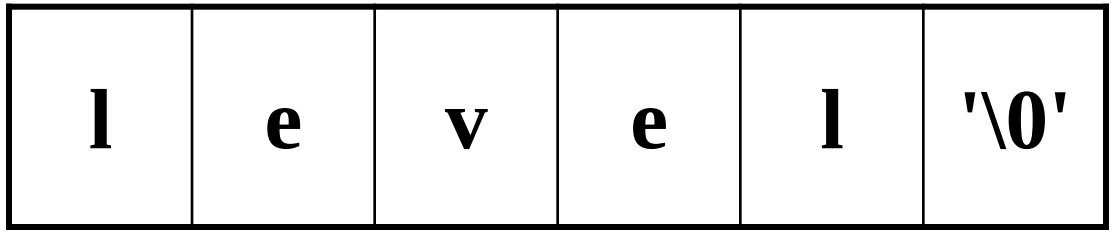
问题描述：

回文是指具有如下特性的一个短语：该短语顺过来读和反过来读所得到的字母序列是完全相同的。例如：“**level**”就是一个回文。编写一个程序，输入一个短语，然后判断它是否是回文。

算法思路

回文判断用首尾比较算法，将字符串的第一个字符与最后一个字符进行比较，第二个字符和倒数第二个字符进行比较，….。若出现不相等的情形，则立即退出。

"level"



str 0 1 2 3 4 5

1. str[0] ↔ str[4]：'l' ↔ 'l'；
2. str[1] ↔ str[3]：'e' ↔ 'e'；
3. str[2] ↔ str[2]
4. 完成！

若字符串长度为N，则最多需要 N / 2 次比较即可。

```
#include <string.h>
#include <stdio.h>
int main( )
{
    char str[80], flag;
    int  i, j;
    printf("输入一个字符串: ");
    scanf("%s", str);
    i = 0;  j = strlen(str)-1; // 初始化
    flag = 'Y';
    for(;  i < j;  i++, j--)
    {
        if(str[i] != str[j])
        {
            flag = 'N';
            break;
        }
    }
    if(flag == 'Y') printf("%s是一个回文\n", str);
    else printf("%s不是一个回文\n", str);
    return 0;
}
```

- 如何测试本程序?
- 汉字回文?

测试数据：

输入一个字符串：a

a是一个回文

输入一个字符串：aa

aa是一个回文

输入一个字符串：ab

ab不是一个回文

输入一个字符串：aba

aba是一个回文

输入一个字符串：abb

abb不是一个回文

输入一个字符串：abccba

abccba是一个回文

输入一个字符串：abcdefghijkl

abcdefghijkl不是一个回文

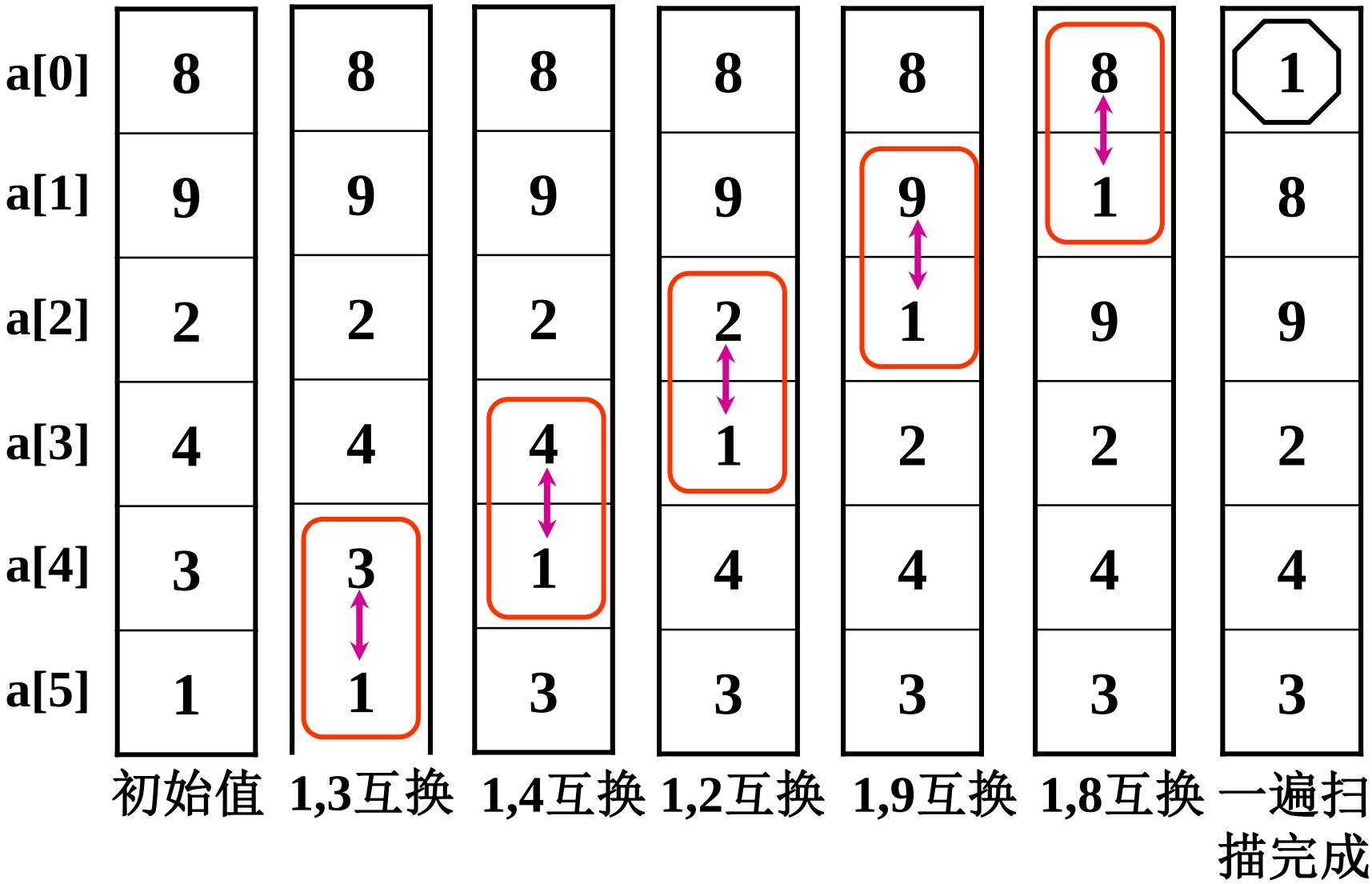
示例3: 冒泡排序

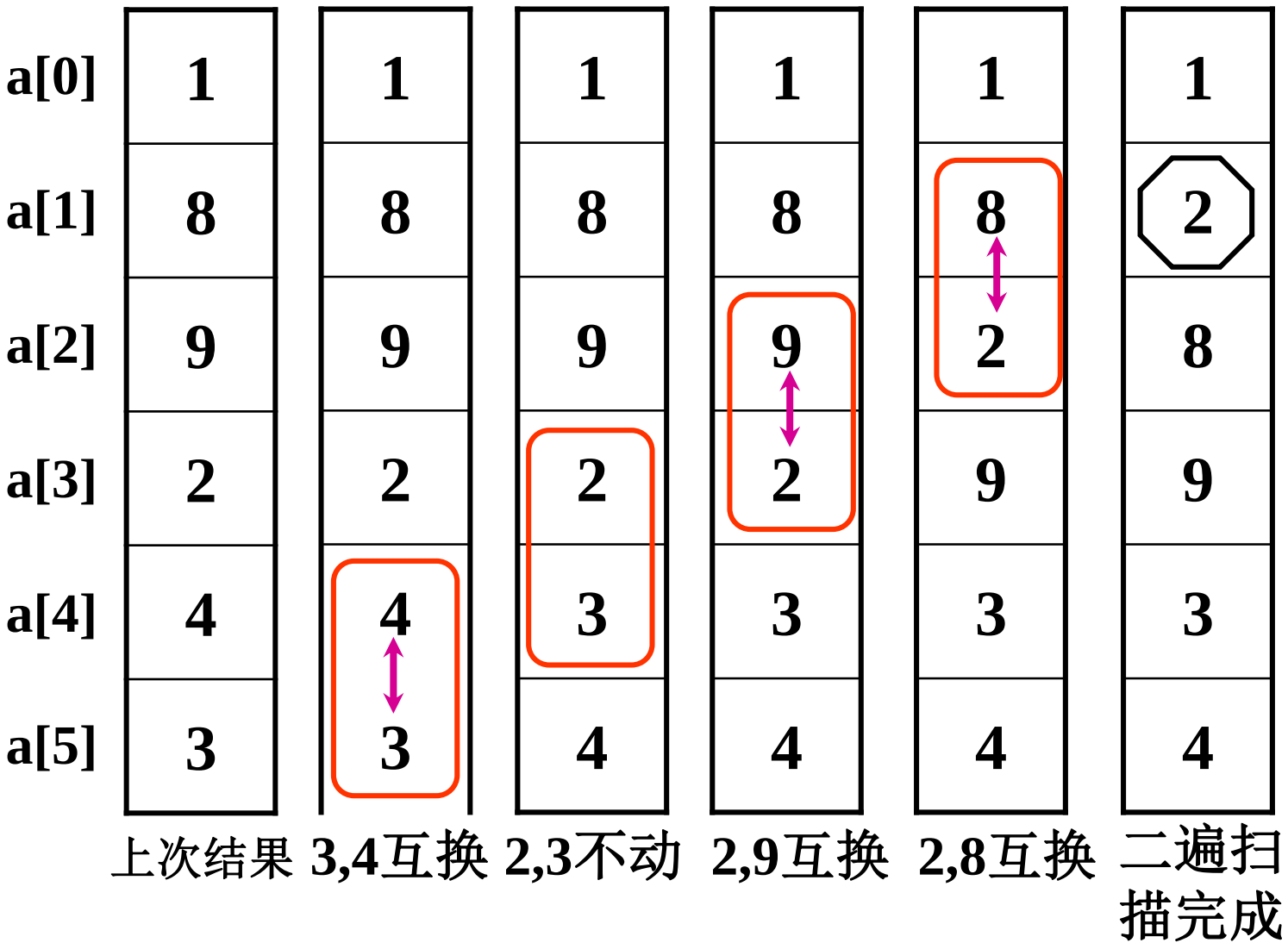
问题描述：

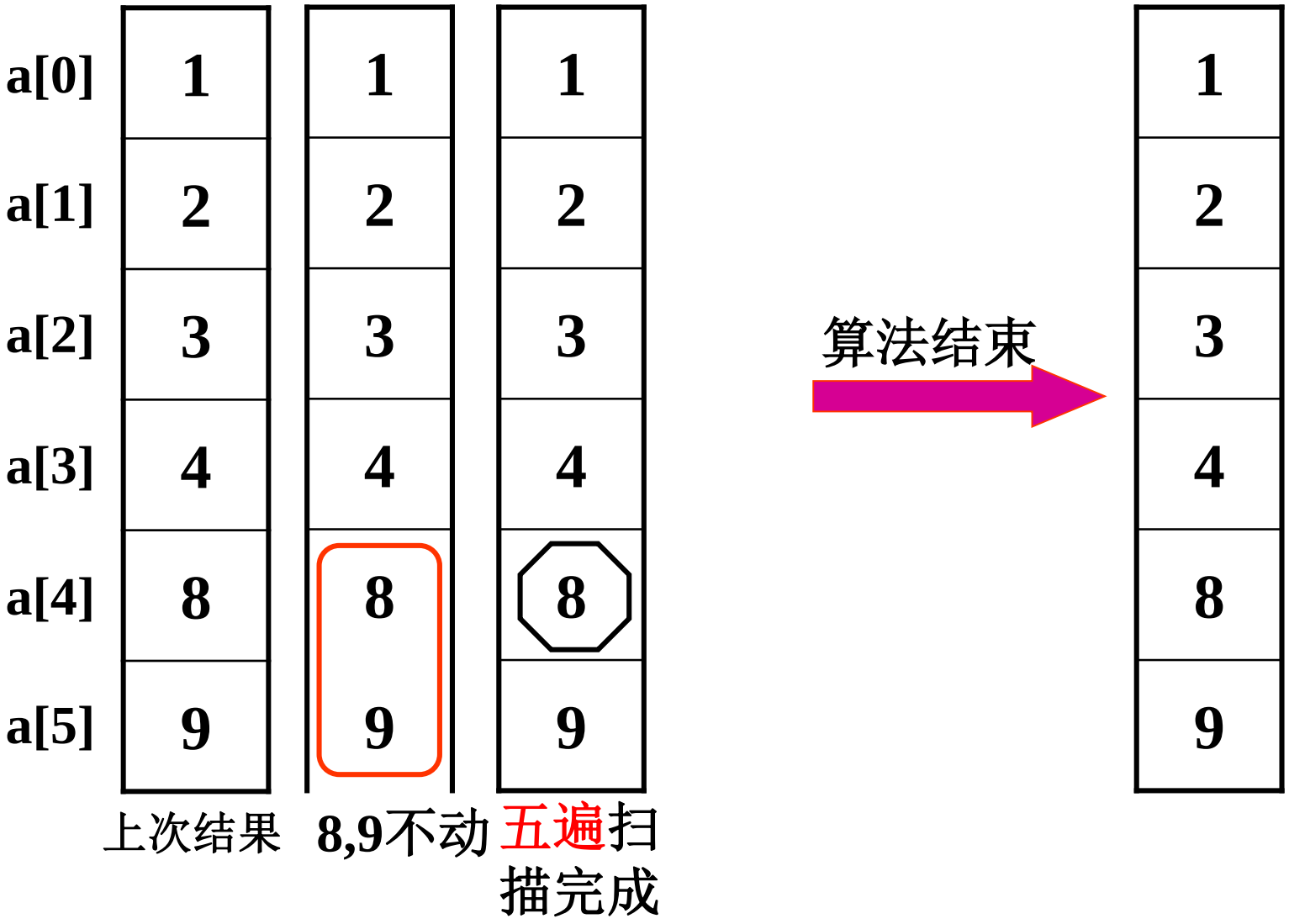
输入一组数据，然后把它们按照升序 (从小到大, ↑) 或降序 (从大到小, ↓) 排列。最后输出排序后的结果。

典型排序算法：
冒泡排序、选择排序、shell排序、快速排序、堆排序等

“冒泡排序”是一种最简单和实用的排序算法


a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
8 9 2 4 3 1
8 9 2 4 3 1
8 9 2 4 1 3
8 9 2 1 4 3
8 9 1 2 4 3
8 1 9 2 4 3
1 8 9 2 4 3
初始值
1,3互换
1,4互换
1,2互换
1,9互换
1,8互换
一遍扫描完成


a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
1 8 9 2 4 3
1 8 9 2 4 3
1 8 9 2 3 4
1 8 9 2 3 4
1 8 2 9 3 4
1 2 8 9 3 4
上次结果
3,4互换
2,3不动
2,9互换
2,8互换
二遍扫描完成


a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
1
2
3
4
8
9
1
2
3
4
8
9
1
2
3
4
8
9
算法结束
1
2
3
4
8
9
上次结果
8,9不动
五遍扫描完成

问题分析

1. 需要多少轮循环？每轮循环得到的最小值保存在何处？
2. 在每轮循环中，需要比较多少次？每次比较哪两个元素？

1、循环次数

从图中可以看出最小的一个数经过第一遍扫描就交换到a[0]中。如果将a[5]视为水底，a[0]视为水面：

最轻的(最小的)一个数 1 最先浮到水面，交换到a[0];
次轻的 2 经过第二遍扫描后交换到a[1];
再轻的 3 经过第三遍扫描后交换到a[2];

…

依此类推，有6个数，前5个数到位需要5遍扫描，而最大的数自然落在a[5]中。因此，6个数只需5遍扫描，每遍扫描得到的最小值，被交换到a[j]当中，若数组长度为n，则 j的取值范围为：　　j = 0, 1, 2, n-2

2、比较次数　　核心问题是i和j的范围！

再来看在每遍扫描中，相邻两个数组元素的比较次数。

（1）当 j = 0 时，需将 a[5]与a[4]比较、a[4]与a[3]比、a[3]与a[2]比、a[2]与a[1]比、a[1]与a[0]比。在比较这5次后，最小数到达a[0]。如果抽象为“a[i]与a[i-1]比”，则 i = 5, 4, 3, 2, 1;

（2）当 j = 1时，即第二遍搜索，需比较4次，之后次小的一个数到达了a[1]。这时a[1]就不必再与a[0]进行比较了。因此 i = 5, 4, 3, 2;

（3）当 j = 2时，即第三遍搜索，i =5, 4, 3；当 j = 3 时，i = 5, 4；当 j = 4 时，i = 5。

若数组长度n，则i 的取值范围：i = n-1, n-2, …, j+1

冒泡排序算法设计：

为了表述方便，定义以下 3 个变量：

n —— 待排序的数的个数，这里 **n = 6**

j —— 扫描遍数，**j = 0, 1, 2, …, n-2**

i —— 第 **j** 遍扫描中待比较元素的下标，即 **a[i]与a[i-1]比，其中i = n-1, n-2,…, j+1**

```
int main( )
{
    int i, j, n, temp, a[6];

    n  =  6;
    for (i = 0;  i < n;  i = i+1)
    {
        printf("请输入待排序的数a[%d]=", i);
        scanf ("%d", &a[i]);
    }
    for(j = 0; j <= n-2; j++)          // 冒泡排序，外层循环
        for(i = n-1; i >= j+1; i--)    // 内层循环
        {
            if(a[i] < a[i-1])          // 把小的数往上冒
            {
                temp = a[i];
                a[i] = a[i-1];
                a[i-1] = temp;
             }
        }
    for(i = 0; i < n; i++)  printf("%d ", a[i]);

    return 0;
}
```

```
for(j = 0; j < n-1; j++)
   for(i = n-1; i > j; i--)
```

单一排序指标！

```
for(j = 1; j <= n; j++)
   for(i = n-1; i >= 1; i--)
```

程序输出结果

```
请输入待排序的数a[0]=8
请输入待排序的数a[1]=9
请输入待排序的数a[2]=2
请输入待排序的数a[3]=4
请输入待排序的数a[4]=3
请输入待排序的数a[5]=1
1 2 3 4 8 9
```

时间复杂度

- 冒泡排序是一种用时间换空间的排序方法；
- 最坏情况是把顺序的排列变成逆序；
- **(n-1) + (n-2) + ... + 1 = n * (n - 1) / 2；**
- 时间复杂度：$O(N^2)$；

冒泡排序again

问题描述：

“软件工程（I）”期末考试结束了，将要进行成绩排名。排名的规则如下：

- 以考试成绩为排名依据(分数为整数)；
- 对学生A，若有K个学生的成绩比他/她高，则其排名为K+1；
- 若学生A和B的成绩相同，则他们的排名也相同，但学号小的放在前面。

样例输入

```
5
1006 95
1002 94
1007 100
1000 95
1001 100
```

样例输出

```
1 1001 100
1 1007 100
3 1000 95
3 1006 95
5 1002 94
```

算法思路

1. 基本思路：

   (1) 先按成绩排序(↓)；

   (2) 如果成绩相同再按学号进行排序 (↑)；

   (3) 最后再计算排名。

2. 如何实现多指标排序？

3. 如何计算排名？

实现方法(1)

```
int score[110],stu[110],id[110];


int cmp(int x,int y)
{
    ……
}
int main()
{
    ……
    return 0;
}
```

实现方法(2)

```
struct student
{
    int NO;     // 排名
    int ID;     // 学号
    int score; // 成绩
};
void sortScore(student *s, int n);
void sortID(student *s, int m);
int main()
{
    ……
    return 0;
}
```

思考题（1）

问题描述：

编写一个程序，从键盘输入一个字符串（长度不超过100个字符），然后将该字符串按如下要求处理：

（1）将大于原首字符的各字符按原来相互间的顺序关系都集中保存在原首字符的左边；

（2）将小于等于原首字符的各字符（按ASCII码值）升序都集中保存在原首字符的右边。

经过上述处理之后，将会得到一个新的字符串，然后将该字符串打印出来。

思考题（1）

说明：（1）可以使用字符数组存储输入的字符串；（2）由于输入的字符串当中可能包含有空白字符（如空格、Tab键），所以应该使用gets函数来读取这个字符串，不要用scanf函数；（3）输入的字符可能包含大小写或标点符号；（4）提示：可以使用gets、puts、strcpy、strcat等字符串处理函数。

样例输入：

abcdABCD1234

样例输出：

bcda1234ABCD

思考题（2）

版本号排序：

软件的版本号只由非负整数和 ‘.’组成，比如：1.3、0.1、3.0.61、3.1.12.5、3.1.12.4。

编写一个程序，程序第一行输入版本号的个数n（1≤n≤100），第二行输入n个字符串，每个字符串表示一个版本号（长度小于100），版本号之间用空格隔开，请将n个版本号从小到大排序并输出。

思考题（2）

样例输入：

5

1.3 0.1 3.0.61 3.1.12.5 3.1.12.4

样例输出：

0.1 1.3 3.0.61 3.1.12.4 3.1.12.5

一种实现方法

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "string.h"
#define M 100
#define N 100
bool CompareVersion(char v1[], char v2[]);
void Swap(char a[], char b[]);
void Sort(char str[][N], int n);
void Print(char str[][N], int n);
int main()
{
        int i, n;        char str[M][N];
        scanf("%d", &n);
        for (i = 0; i < n; i++)   scanf("%s", str[i]);
        Sort(str, n);
        Print(str, n);
        return 0;
}
```

```
void Sort(char str[][N], int n) //冒泡排序
{
        char min[N];
        int i, j;
        for (i = 0; i < n - 1; i++)
        {
                for (j = i; j < n - 1; j++)
                {
                        if (CompareVersion(str[j], str[j+1]))
                        {
                                Swap(str[j], str[j+1]);
                        }
                }
        }
}
bool CompareVersion(char v1[], char v2[])
{
}
```

示例4: 找数

问题描述：

在下列数据网格中寻找一个整数(<32000)，方法是：(1)在某一行从左往右或从右往左找；(2)在某一列从上往下或从下往上找。如整数3451在该网格中出现了两次，即(3, 1, 下)和(3, 3, 右)。需考虑回绕的情形，如5412出现在(1, 5, 右)。

1	2	3	4	5	4
2	5	4	3	2	1
3	5	3	4	5	1
4	4	3	4	5	2
5	3	4	3	4	3
1	4	5	2	3	4

问题分析

1. 数据网格如何存放？
2. 该整数有多少位？
3. 假设有M位，从某个位置出发，如何得到从左往右、从右往左、从上往下和从下往上的M个数字？
4. 在得到了M个数字之后，如何把它合成为一个整数？

假设有M位，数组为a，当前位置为（i, j）

- 从左往右：**(j+k) % 6 (k = 0, 1, …, M-1)**
- 从右往左：**(j-k+6) % 6 (k = 0, 1, …, M-1)**
- 从上往下：**(i+k) % 6 (k = 0, 1, …, M-1)**
- 从下往上：**(i-k+6) % 6 (k = 0, 1, …, M-1)**

```
1 2 3 4 5 4
2 5 4 3 2 1
3 5 3 4 5 1
4 4 3 4 5 2
5 3 4 3 4 3
1 4 5 2 3 4
```

```
#include <stdio.h>
int main()
{
    int a[6][6] = { 1, 2, 3, 4, 5, 4,
                    2, 5, 4, 3, 2, 1,
                    3, 5, 3, 4, 5, 1,
                    4, 4, 3, 4, 5, 2,
                    5, 3, 4, 3, 4, 3,
                    1, 4, 5, 2, 3, 4};
    int i, j, k;
    int value, temp, N=6, M;

    scanf("%d", &value);
    temp = value;
    M = 0;
    while(temp != 0){
        M++;
        temp /= 10;
    }
```

```
    for(i = 0; i < N; i++)
    for(j = 0; j < N; j++)
    {
        temp = 0;
        for(k = 0; k < M; k++)
            temp = temp * 10 + a[i][(j+k)%N];
        if(temp == value) printf("%d %d 右\n",i+1,j+1);
        temp = 0;
        for(k = 0; k < M; k++)
            temp = temp * 10 + a[i][(j-k+N)%N];
        if(temp == value) printf("%d %d 左\n",i+1,j+1);
        temp = 0;
        for(k = 0; k < M; k++)
            temp = temp * 10 + a[(i-k+N)%N][j];
        if(temp == value) printf("%d %d 上\n",i+1,j+1);
        temp = 0;
        for(k = 0; k < M; k++)
            temp = temp * 10 + a[(i+k)%N][j];
        if(temp == value) printf("%d %d 下\n",i+1,j+1);
    }
    return 0;
}
```

测试数据：

```
411
1 6 下
```

```
3454
1 3 右
```

```
3451
3 1 下
3 3 右
```

```
154
3 6 左
6 1 上
```

```
5435
3 5 左
4 5 下
```

```
1 2 3 4 5 4
2 5 4 3 2 1
3 5 3 4 5 1
4 4 3 4 5 2
5 3 4 3 4 3
1 4 5 2 3 4
```

示例5: DNA序列

问题背景：

人类基因组计划的第一阶段于2000年6月26日胜利结束，我国科研工作者圆满地完成了其中的1%的测序工作。

众所周知，组成DNA的碱基有四种：腺嘌呤(A)、鸟嘌呤(G)、胞嘧啶(C)、胸腺嘧啶(T)。对任意两个人，其染色体上的DNA序列大部分相同，但总会有少数碱基对不同。这种不同是由基因的变异引起，但每个人的变异位置不尽相同。这样，对于大部分位点来说，很可能的情形是：大部分人在该位点上的碱基是一致的（没有发生变异），少数人具有不同的碱基（发生了变异）。

这就给科学家一个启示：在测序过程中，若仅仅使用一个人的样本，在很多位点上测出的结果就不具有代表性；而如果能测出多个人的序列，则有可能“整合”出一段具有人类共性的序列出来，这样更有利于研究。

如，若要测人的某一段DNA序列，得到4人的样本：

AAAGGCCT
AGAGCTCT
AAGGATCT
AAACTTCT　**AAAGATCT**

整合规则：(1)取出在每个位置上出现次数最多的碱基作为整合后该位置上的碱基；(2)若某个位置出现次数最多的碱基不止一种，则优先选择A，其次C、G和T。

问题描述：

对一组DNA序列进行整合。

输入：N行(2<=N<=10)，每一行是一个DNA序列，具有相同的长度(2到100之间)；

输出：一个字符串，即整合后的序列。

问题分析：

一、数据结构

（1）这组DNA序列如何存放？

（2）如何记录在每个位置上各种碱基的出现次数？

二、算法

（1）如何实现字符到数组下标的映射？

（2）如何统计各位置上每种碱基的出现次数

（3）如何找到出现次数最多的碱基？

（4）若次数相同，如何实现ACGT优先顺序

```
4
AAAGGCCT
AGAGCTCT
AAGGATCT
AAACTTCT
```

```
#include <stdio.h>
#include <string.h>
int main()
{
    char DNA[10][101], Result[101];
    char Jianji[4] = {'A', 'C', 'G', 'T'};
    double Num[4], Max;
    int i, j, N, len;

    scanf("%d", &N);
    fflush(stdin);
    for(i=0; i<N; i++) scanf("%s", DNA[i]);
    len = strlen(DNA[0]);
```

```
4
AAAGGCCT
AGAGCTCT
AAGGATCT
AAACTTCT
```

```
for(j = 0; j < len; j++)
{
    Num[0] = 0.3;  Num[1] = 0.2;
    Num[2] = 0.1;  Num[3] = 0;
    for(i = 0; i < N; i++)
    {
        switch(DNA[i][j])
        {
        case 'A':   Num[0] += 1;  break;
        case 'C':   Num[1] += 1;  break;
        case 'G':   Num[2] += 1;  break;
        case 'T':   Num[3] += 1;  break;
        }
    }
```

```
4
AAAGGCCT
AGAGCTCT
AAGGATCT
AAACTTCT
```

```
            Max = 0;
            for(i = 0; i < 4; i++)
            {
                if(Num[i] > Max)
                {
                    Max = Num[i];
                    Result[j] = Jianji[i];
                }
            }
        }
        Result[j] = 0;
        printf("整合结果: %s\n", Result);
        return 0;
}
```

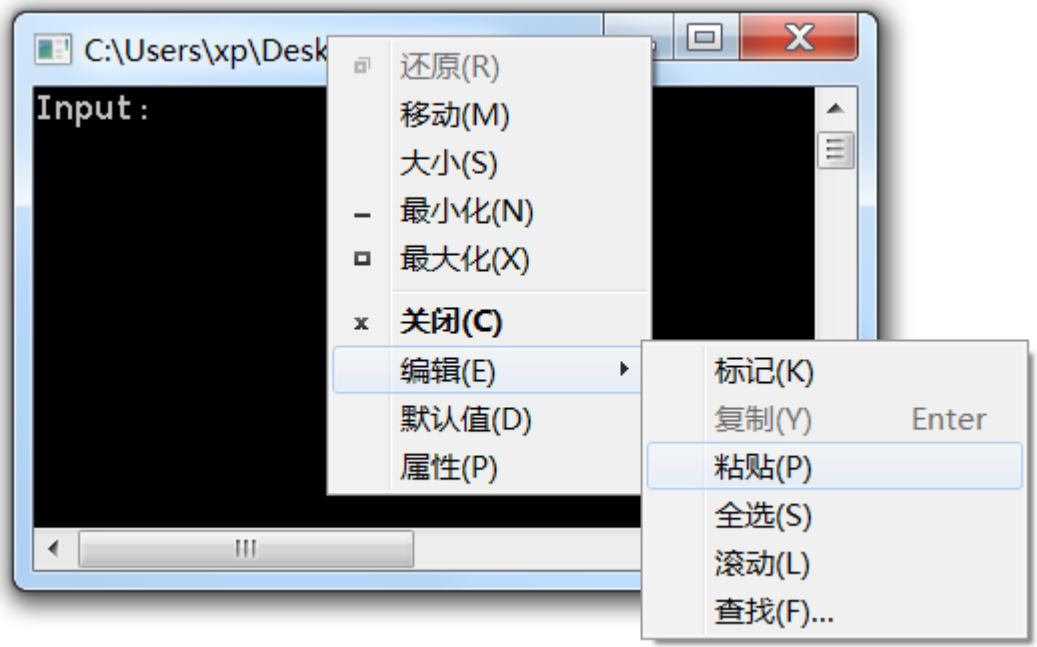
运行结果：

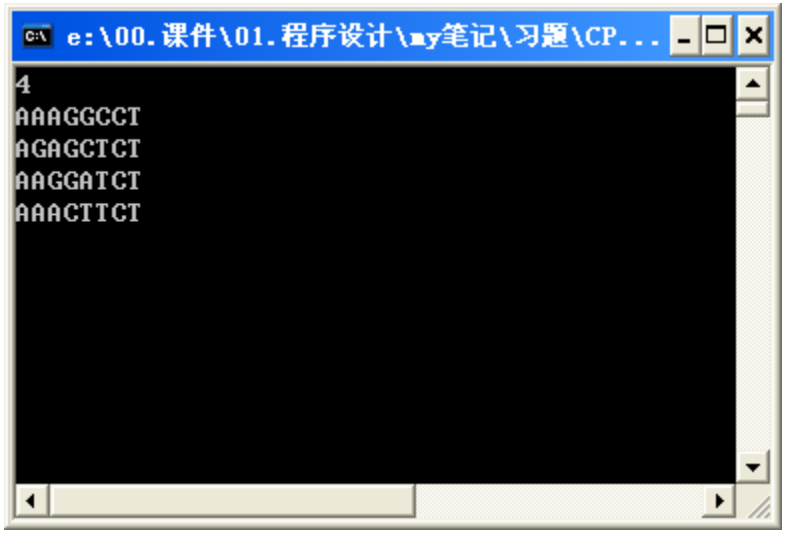
```
4
AAAGGCCT
AGAGCTCT
AAGGATCT
AAACTTCT
整合结果：
AAAGATCT
```

	A	C	G	T
第0列Num:	4.3	0.2	0.1	0.0
第1列Num:	3.3	0.2	1.1	0.0
第2列Num:	3.3	0.2	1.1	0.0
第3列Num:	0.3	1.2	3.1	0.0
第4列Num:	1.3	1.2	1.1	1.0
第5列Num:	0.3	1.2	0.1	3.0
第6列Num:	0.3	4.2	0.1	0.0
第7列Num:	0.3	0.2	0.1	4.0

技巧1: 调试数据

- **注意：键盘输入的值可以写在记事本里(或者copy剪切板)，然后右键输入窗口标题处，选择编辑，粘贴，这样可以减少调试中反复输入数据。**



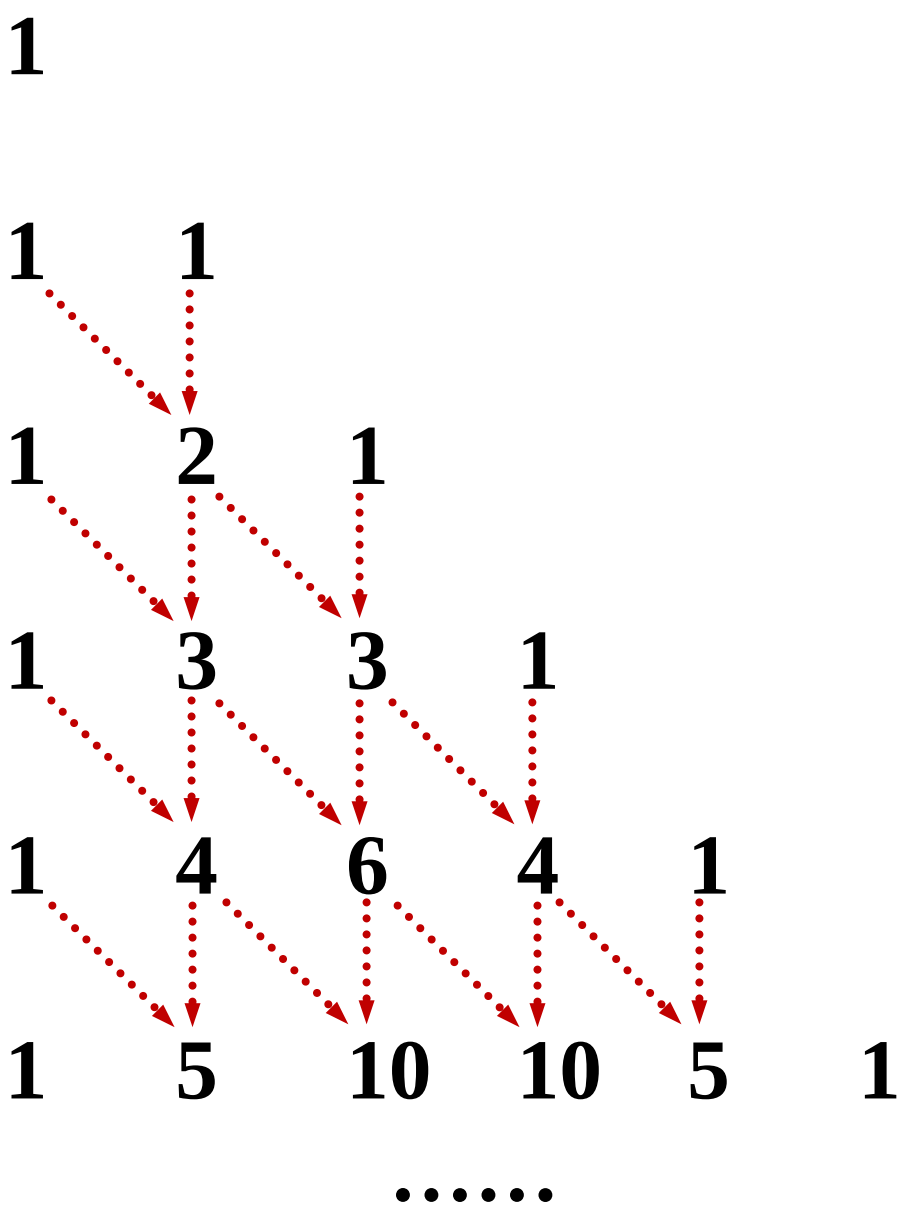


示例6: 杨辉三角形

问题描述： 打印杨辉三角形的前十行。

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
1   6   15  20  15  6   1
1   7   21  35  35  21  7   1
1   8   28  56  70  56  28  8   1
1   9   36  84  126 126 84  36  9   1
```

何谓杨辉三角形？即满足如图所示的叠加关系。



方法1：二维数组

- 定义一个二维数组a[10][10]，用来存放杨辉三角形。存放方式：每一行的数组元素对应于三角形的一行数据；
- 对于边角上的数据，设置为1；
- 对于其他的元素，使用如下的公式来计算生成：a[i][j] = a[i-1][j-1] + a[i-1][j]; 其中a[i][j]表示第i 行第 j 列的数组元素；
- 最后，把生成的杨辉三角形打印出来。

```
int  i,  j,  a[10][10];
a[0][0]  =  1;
for( i  =  1;  i  <=  9;  i++ )  // 计算每一行的数据
{
      a[i][0] = 1;          // 第 i 行的第1个元素
      a[i][i] = 1;          // 第 i 行的最后一个元素，对角线
      for( j  =  1;  j  <  i;  j++ )
            a[i][j]  =  a[i-1][j-1]  +  a[i-1][j];
}
for(i = 0; i <= 9; i++) // 打印杨辉三角形
{
      for(j = 0;  j <= i;  j ++)   printf("%-4d", a[i][j]);
      printf("\n");
}
```

```
1
1  1
1  2  1
1  3  3  1
1  4  6  4  1
1  5  10 10 5  1
1  6  15 20 15 6  1
1  7  21 35 35 21 7  1
1  8  28 56 70 56 28 8  1
1  9  36 84 126 126 84 36 9  1
```

此方法浪费了近一半的存储空间（45/100）！

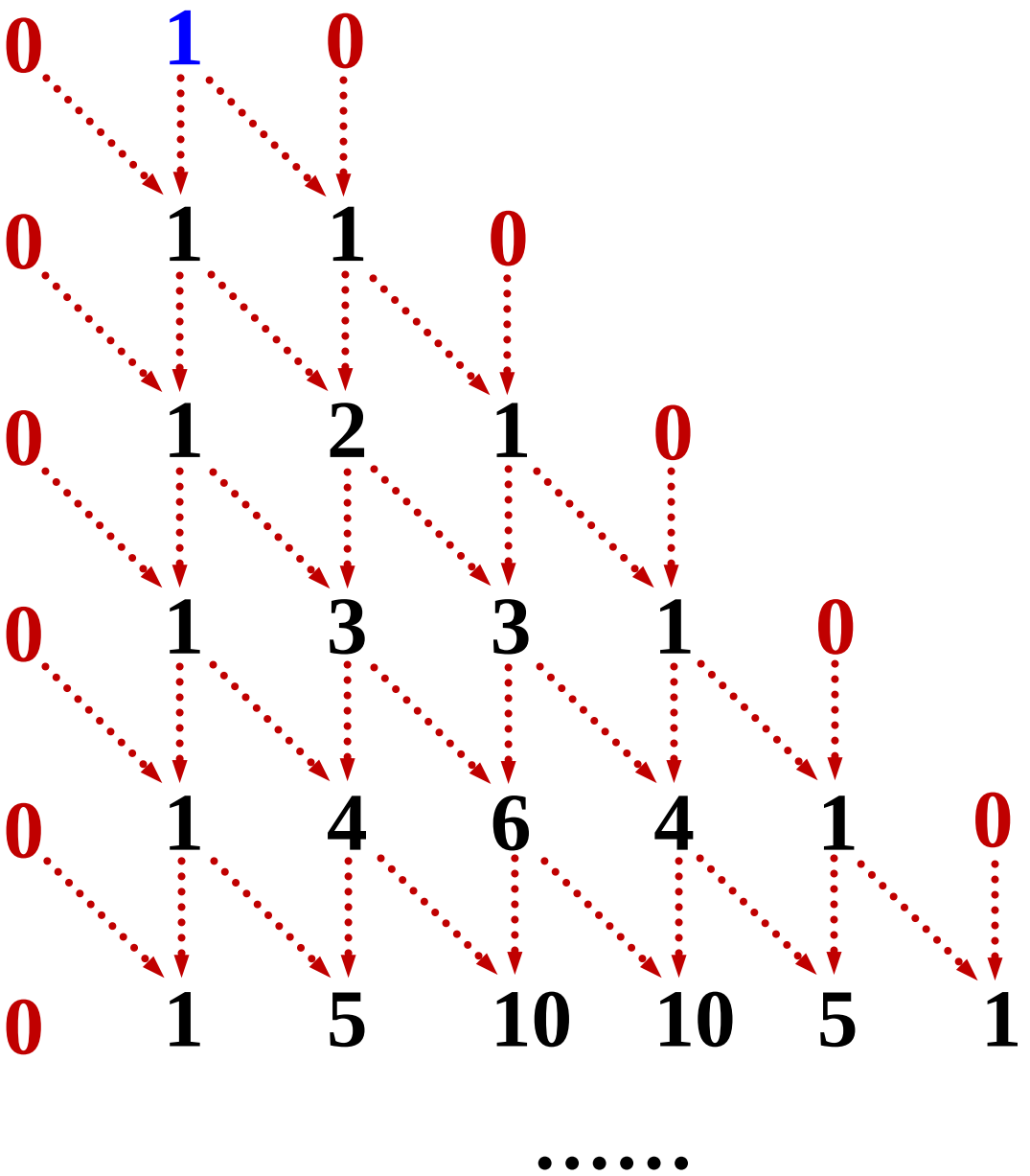
方法2：一维数组

能否用一维数组来解决这个问题？

如果能，怎么做？

```
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5    10   10   5    1
1    6    15   20   15   6    1
……
```

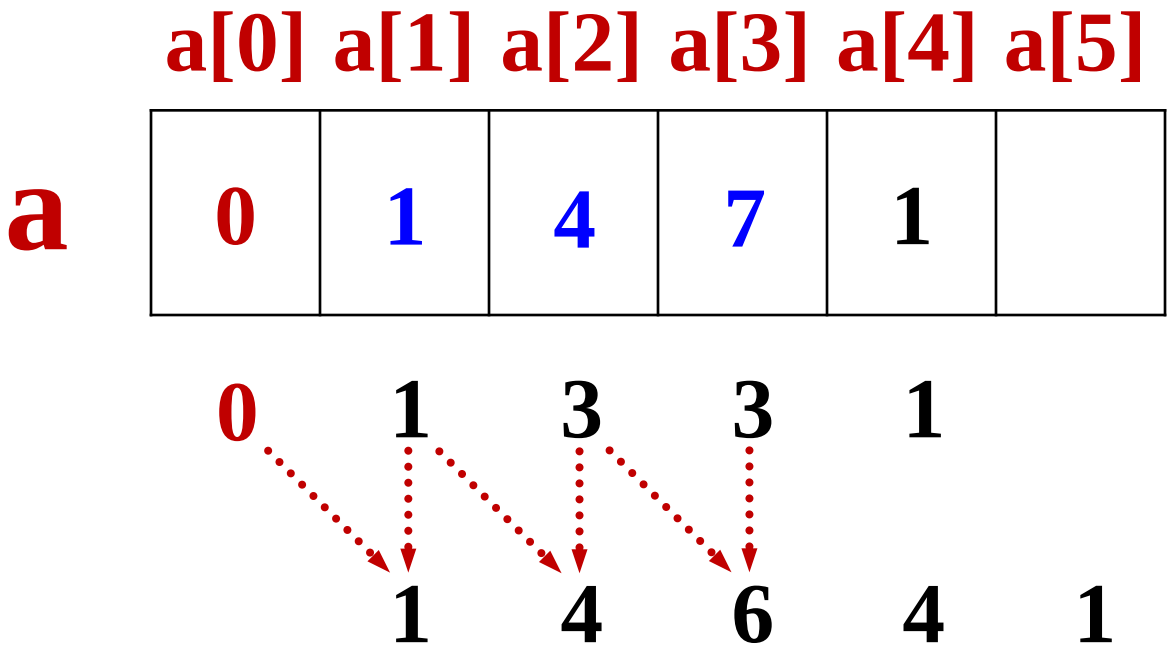
叠加关系的拓展：



程序框架

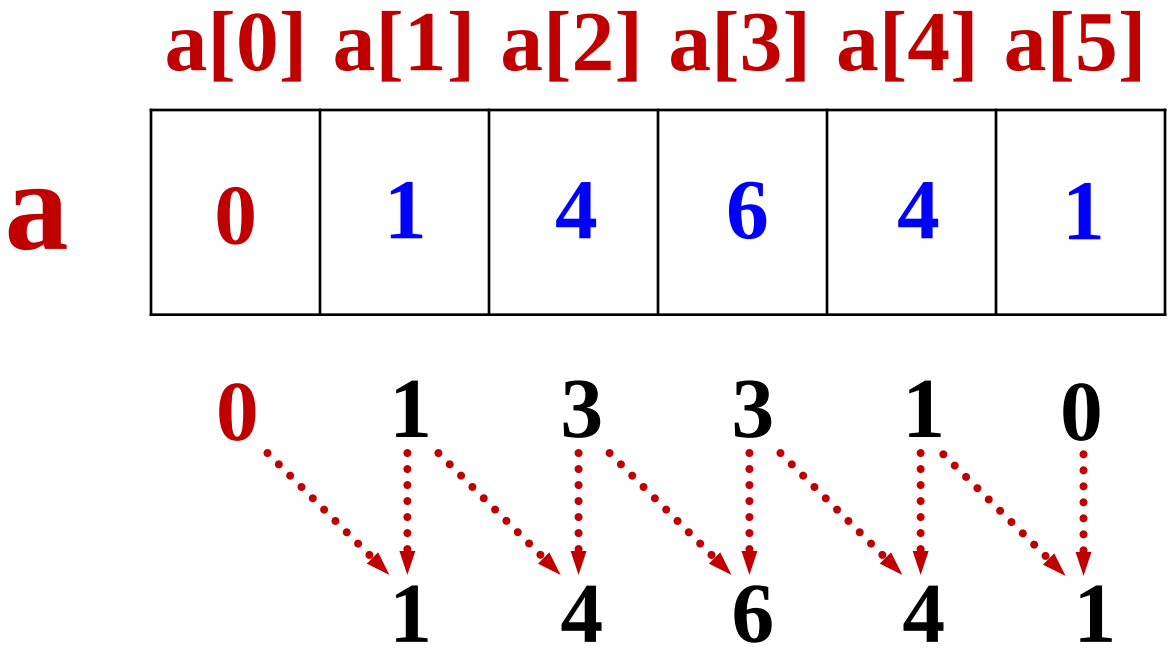
```
…
int  i,  j,  a[11];
for( i  =  1;   i  <=  10;   i++ )  // 计算每行数据并打印
{
        …
        for( j  = … )
        {
                a[j]  =  a[j]  +  a[j-1]; // 叠加关系
        }
        for(j = 1;  j <= i;  j++)    printf("%-4d",  a[j]);
        printf("\n");
}
```

下一行数据的计算（从左到右？）




```
for( j = 1; j <= i; j++ )   // 从左到右？
    a[j] = a[j] + a[j-1];
```

下一行数据的计算（从右到左！）




```
for( j = i;  j >= 1;  j-- )   // 从右到左！
    a[j] = a[j] + a[j-1];
```

算法设计：

Step1. 定义数组a[11]，和两个循环变量 i, j

Step2. a[0] = 0; i = 1;

Step3. 如果 i 等于1，a[i] = 1；否则 a[i] = 0;

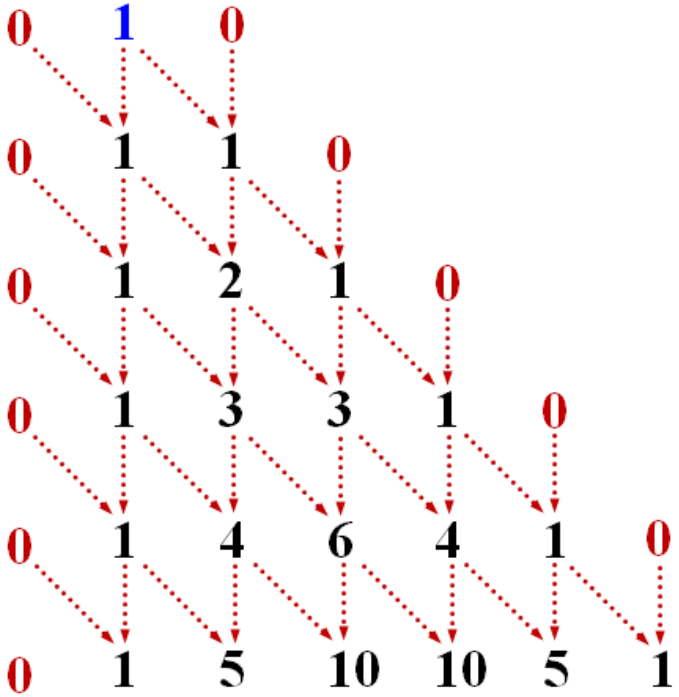
Step4. 从 j = i 直到 j =1，进行叠加运算 a[j] += a[j-1];

这样，就把第 i –1 行的数据元素更新为第 i 行的数据元素。

Step5. 显示第 i 行的数据元素；

Step6. 把 i ++，如果 i <= 10，转Step3；否则算法结束

```
#include <stdio.h>
int main()
{
    int i, j, a[11];
    a[0] = 0;
    for( i = 1; i <= 10; i++ )
    {
        if (i == 1)  a[i] = 1;
        else  a[i] = 0;
        for( j = i; j >= 1; j-- )
            a[j] = a[j] + a[j-1];
        for (j = 1; j <= i; j++)
            printf("%-4d", a[j]);
        printf("\n");
    }
    return 0;
}
```



Lecture 5 - Summary

- **Topics covered:**
  - The concept of array
  - Defining arrays
  - Initializing arrays
  - Character arrays
  - Multidimensional arrays
  - Variable length arrays
- **应用示例:**
  - 字符串处理、冒泡排序算法