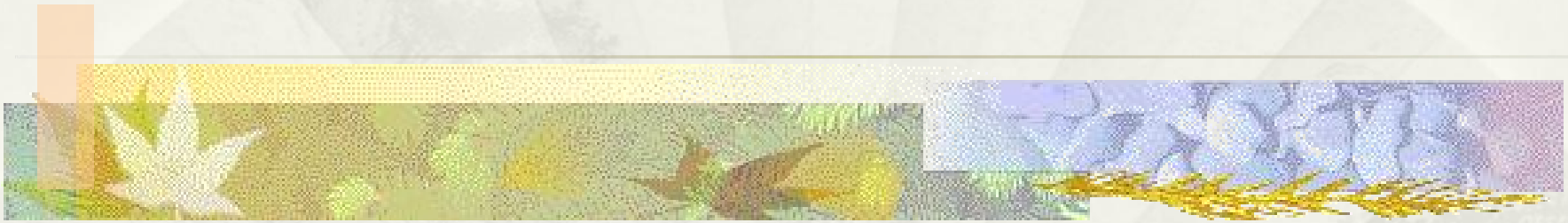
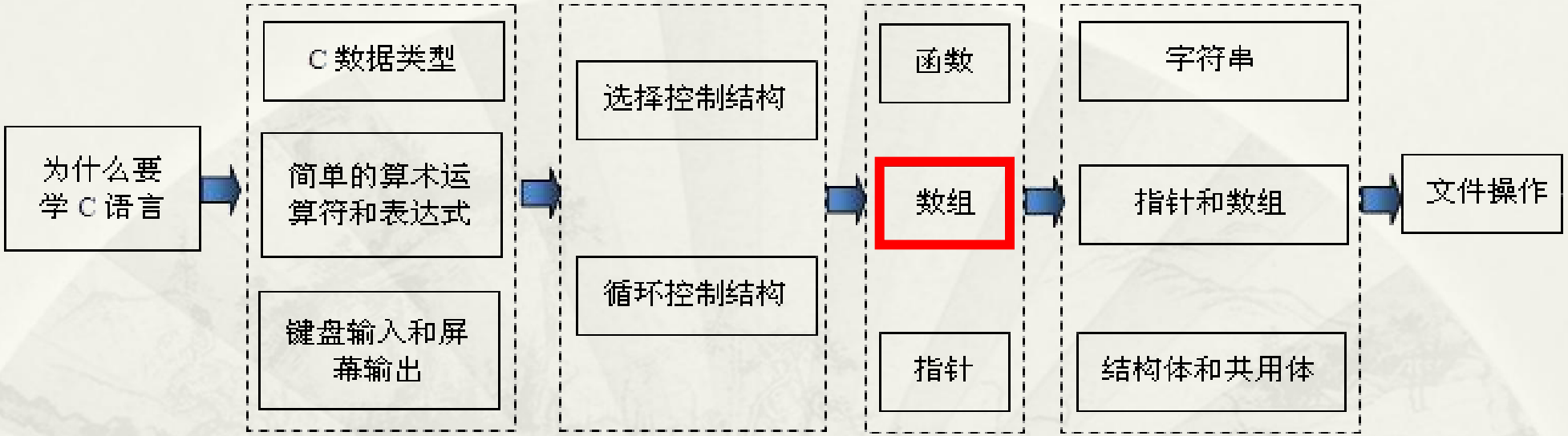
哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

规格严格 功夫到家

第8章 数组



哈尔滨工业大学（深圳）
计算机科学与技术学院
刘洋
Liu.yang@hit.edu.cn



简单的数据结构 → 复杂的数据结构

第8章　　学习内容

- 对数组名特殊含义的理解
- 数组的定义和初始化
- 向函数传递一维数组
- 向函数传递二维数组
- 排序、查找、求最值等算法

为什么使用数组(Array)?

- 【例8. 1】要读入并存储10人的成绩，然后求平均成绩
- 需定义10个不同名的整型变量，需使用多个**scanf()**

```
int score1, score2, … score10;
scanf("%d", &score1);
scanf("%d", &score2);
......
```

- 而数组仅用一个**scanf()**并利用循环语句读取

```
int score[10],i;
for (i=0; i<10; i++)
{
      scanf("%d", &score[i]);
}
```

8.1一维数组的定义和初始化

- 一维数组的定义

```
int a[10];
```

数据类型

代表元素个数

下标从0开始
数组名a代表首地址

- 定义一个有10个**int**型元素的一维数组
  - 系统分配连续的10个**int**型存储空间给此数组
- 为什么数组下标从0开始？
  - 使编译器简化，且运算速度少量提高
- 如果希望下标从1到10而非从0到9，怎么办？

8.1一维数组的定义和初始化

- 一维数组的定义

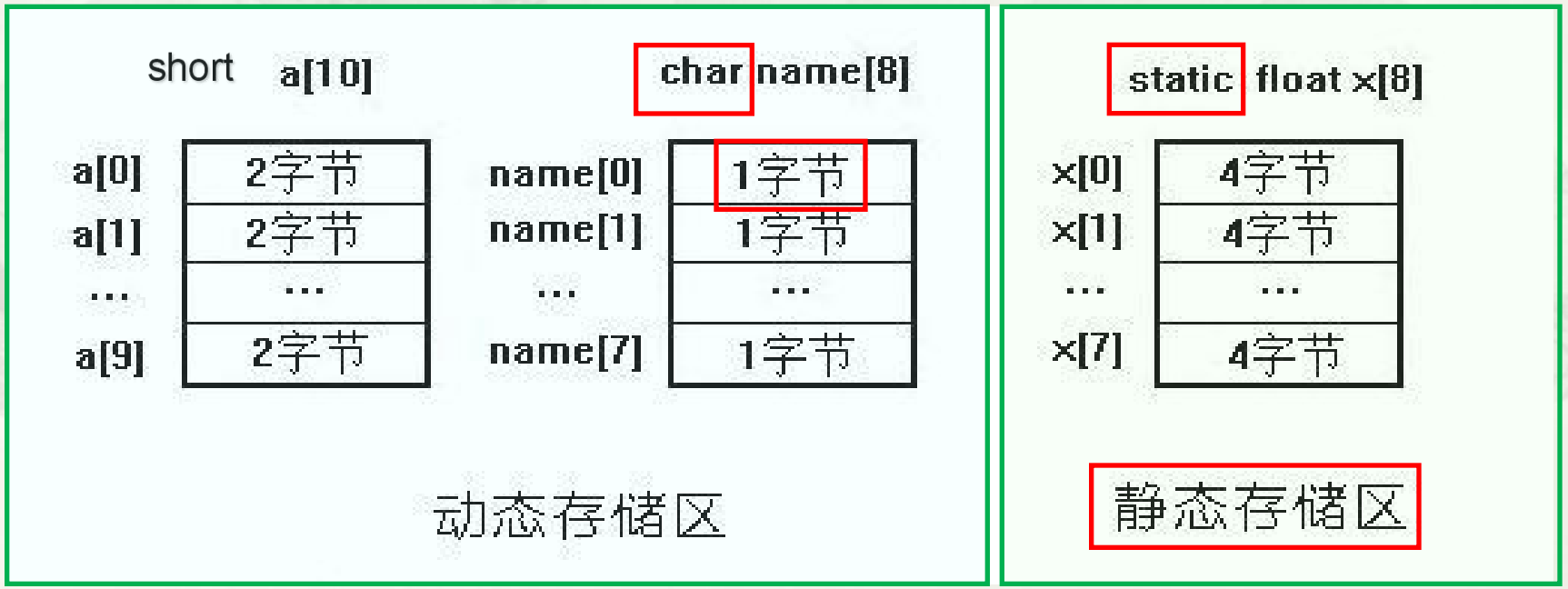
```
int a[10];
```

- 数组大小必须是值为正的常量，不能为变量，一旦定义，不能改变大小
- 大小最好用宏定义，以适应未来的变化

```
#define N 10
int a[N];
```

8.1一维数组的定义和初始化

- 根据数组的数据类型，为每一元素安排相应字节数的存储单元
- 根据数组的存储类型，将其安排在内存的动态、静态存储区或寄存器区



8.1一维数组的定义和初始化

```
int a[10];
```

引用时下标允许是int型变量或表达式

- 一维数组的引用

  数组名[下标]

- 允许快速随机访问
  - 允许使用a[i]这样的形式访问每个元素
  - 可以像使用普通变量一样使用

    a[0],a[1],...,a[9]

8.1一维数组的定义和初始化

- 未初始化的数组元素值是什么？
  - 静态数组和全局数组自动初始化为0值
  - 否则，是随机数

- 一维数组的初始化

```
int a[5] = { 12, 34, 56, 78, 9 };
int a[5] = { 0 };
int a[] = { 11, 22, 33, 44, 55 };
```

8.1一维数组的定义和初始化

如何使两个数组的值相等？

```
int a[4] = {1,2,3,4};
int b[4];
b = a;
```

数组名表示数组的首地址，
不代表整个数组元素值！

解决方法

- 方法1：逐个元素赋值

```
b[0]=a[0];
b[1]=a[1];
b[2]=a[2];
b[3]=a[3];
```

- 方法2：通过循环语句赋值

```
int i;
for (i=0;i<4;i++)
{
    b[i] = a[i];
}
```

8.1一维数组的定义和初始化

- 更高效的数组初始化方法

```
memset(a, 0, sizeof(a));
```

- 用sizeof(a)来获得数组a所占的内存字节数
- 更高效的数组赋值方法

```
memcpy(b, a, sizeof(a));
```

- 需要包含相应的头文件：

```
#include <string.h>
```

8.1一维数组的定义和初始化

- 【例8.2】编程实现显示用户输入的月份（不考虑闰年）拥有的天数

```
1   #include <stdio.h>
2   #define  MONTHS 12
3   int main()
4   {
5       int days[MONTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};
6       int month;
7       do{
8          printf("Input a month:");
9          scanf("%d", &month);
10      }while(month < 1 || month > 12); /* 处理不合法数据的输入 */
11      printf("The number of days is %d\n", days[month-1]);
12      return 0;
13  }
```

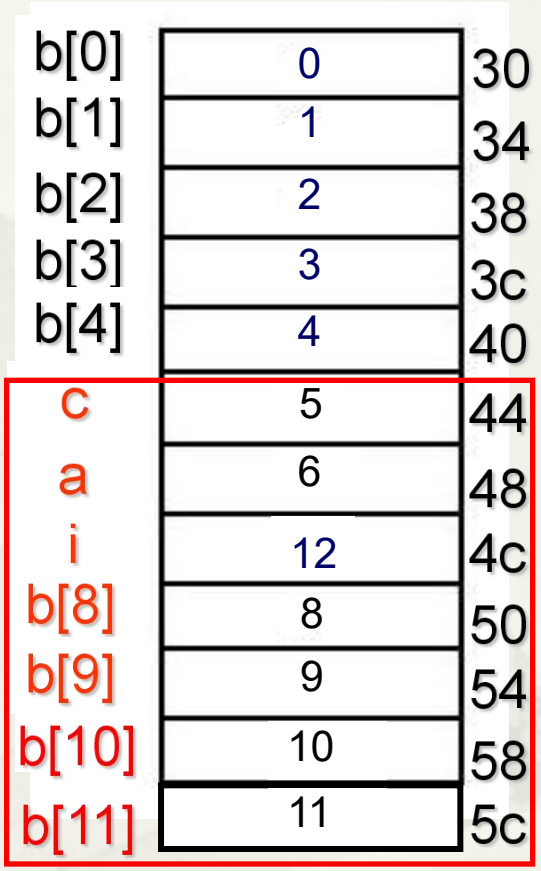
为什么要处理非法的数据输入？

8.1一维数组的定义和初始化

- 访问数组元素时，下标越界是大忌！
  - 编译程序不检查下标越界，导致运行时错误
  - 下标越界，将访问数组以外的空间
  - 那里的数据是未知的，不受我们掌控，可能带来严重后果
  - 后果有多严重呢？

【例8.3】当下标值小于0或超过数组长度时会出现什么情况？

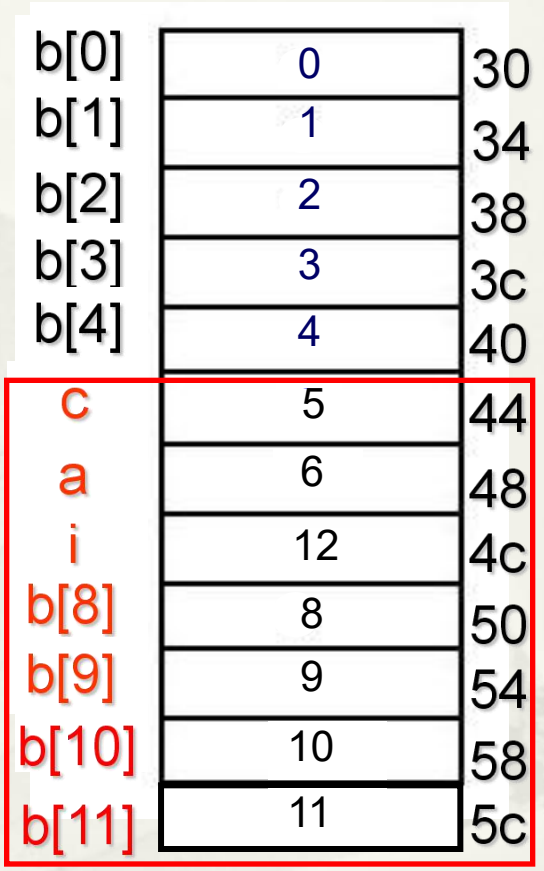
```
1   #include <stdio.h>
2   int main()
3   {
4       int a = 1, c = 2, b[5], i;
5       printf("%p, %p, %p\n", b, &a, &c);
6       for (i=0; i<=11; i++)
7       {
8           b[i] = i;
9           printf("%d  ", b[i]);
10      }
11      printf("\na=%d, c=%d\n", a, c);
12      return 0;
13  }
```



变量c和a的值，因数组下标越界而被悄悄破坏了

【例8.3】当下标值小于0或超过数组长度时会出现什么情况？

```
1   #include <stdio.h>
2   int main()
3   {
4       int a = 1, c = 2, b[5], i;
```



输出的地址值是系统相关的

8.2二维数组的定义和初始化

- 一维数组： `int a[5];`
  - 用一个下标确定各元素在数组中的顺序
  - 可用排列成一行的元素来表示

a[0]	a[1]	a[2]	a[3]	a[4]

- 二维数组: `int b[2][3];`
  - 用两个下标确定各元素在数组中的顺序
  - 用排列成i行，j列的元素来表示
  - 逻辑结构，非物理结构
  - 内存中线性保存

b[0][0]	b[0][1]	b[0][2]
b[1][0]	b[1][1]	b[1][2]

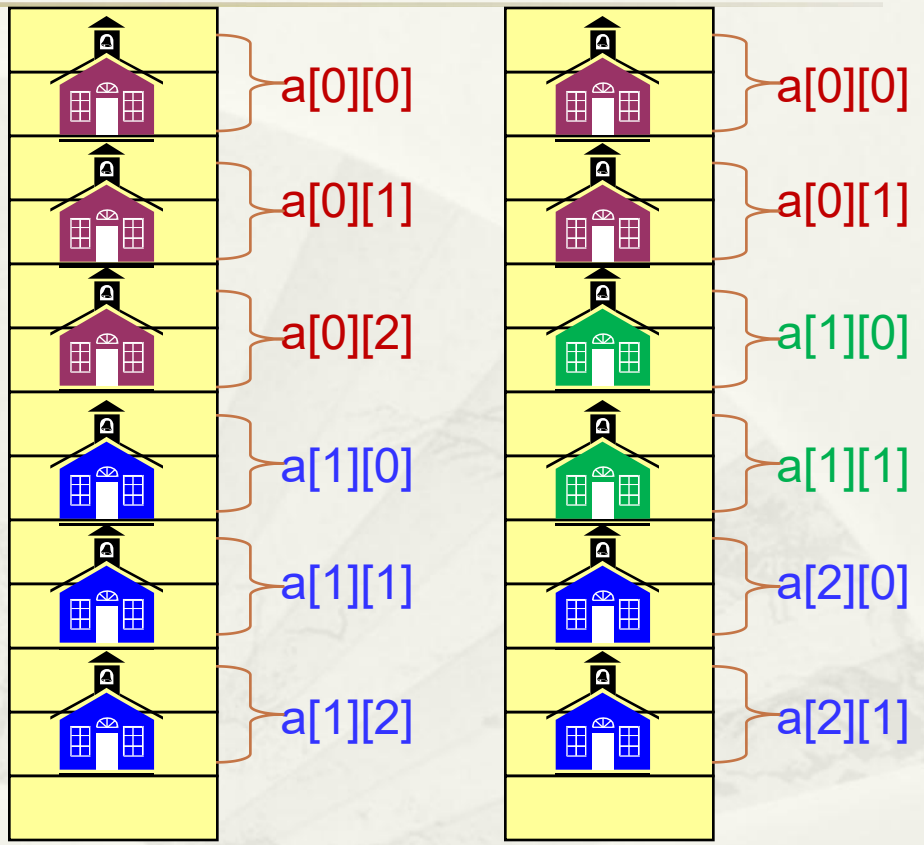
- n维数组: `int c[3][2][4];`
  - 用n个下标来确定各元素在数组中的顺序

8.2二维数组的定义和初始化

存放顺序：按行存放，线性存储

先顺序存放第0行，再存放第1行

```
short a[2][3];
```



若short a[3][2];则…

已知每行列数才能正确读出数组元素

所以初始化时，第二维长度不能省略

8.2二维数组的定义和初始化

【例】以下程序的运行结果是什么？

```
int main()
{
    int a[][3]={{1,2,3},{4,5},{6},{0}};
    printf("%d,%d,%d\n",a[1][1],a[2][1],a[3][1]);
    return 0;
}
```

1	2	3
4	5	0
6	0	0
0	0	0

结果：5，0，0

【例】若int a[ ][3]={1, 2, 3, 4, 5, 6, 7};
则a数组的第一维大小是多少？

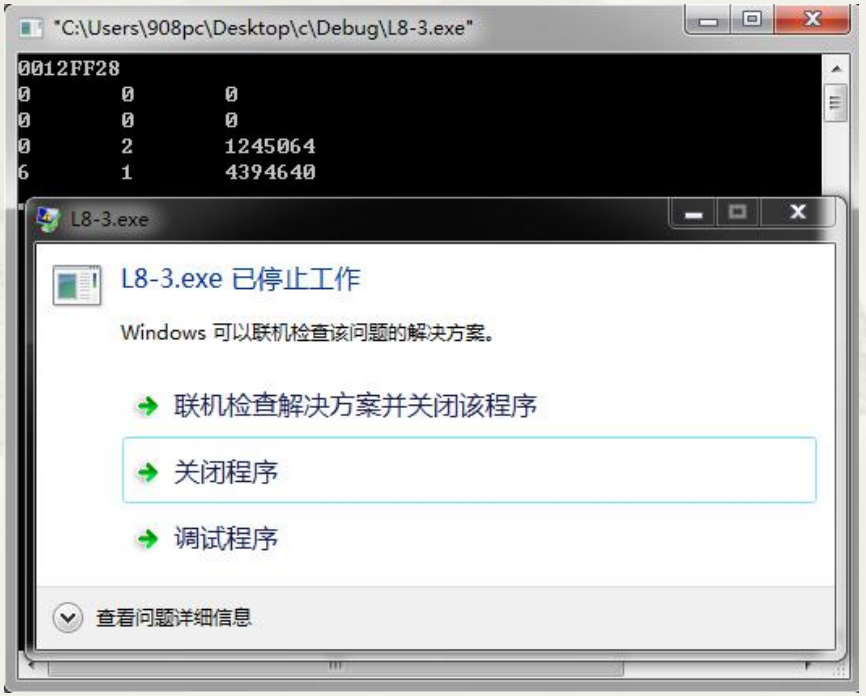
1	2	3
4	5	6
7	0	0

8.2二维数组的定义和初始化

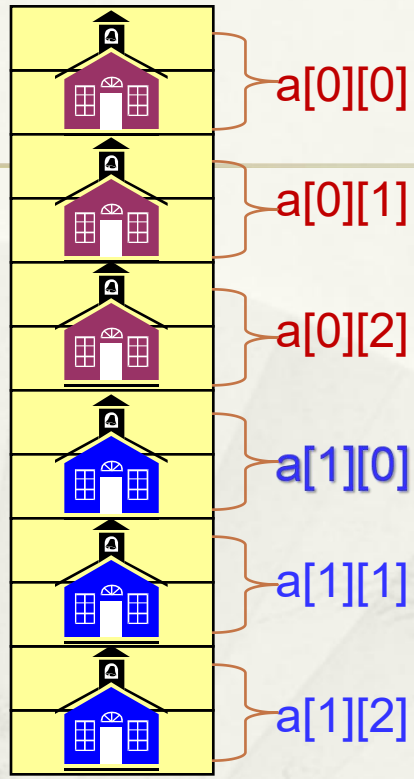
二维数组下标越界

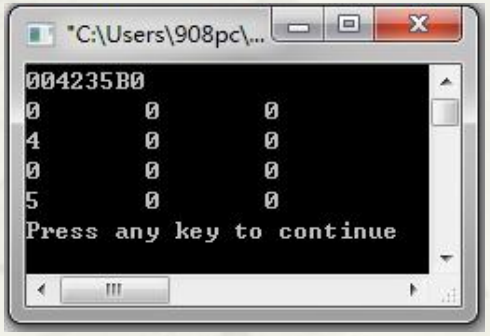
```
#include <stdio.h>
int main()
{
    int i, j;
    int a[2][3] = {0};
    printf("%p\n", a);
    a[3][0] = 6;
    for (i=0; i<4; i++)
    {
        for (j=0; j<3; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[1][1]	a[1][2]
a[3][0]	a[1][1]	a[1][2]



```
#include <stdio.h>
int main()
{
    int i, j;
    int a[2][3] = {0};
    printf("%p\n", a);
    a[1][0] = 4;
    for (i=0; i<2; i++)
    {
        for (j=0; j<3; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    a[0][3] = 5;
    for (i=0; i<2; i++)
    {
        for (j=0; j<3; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```



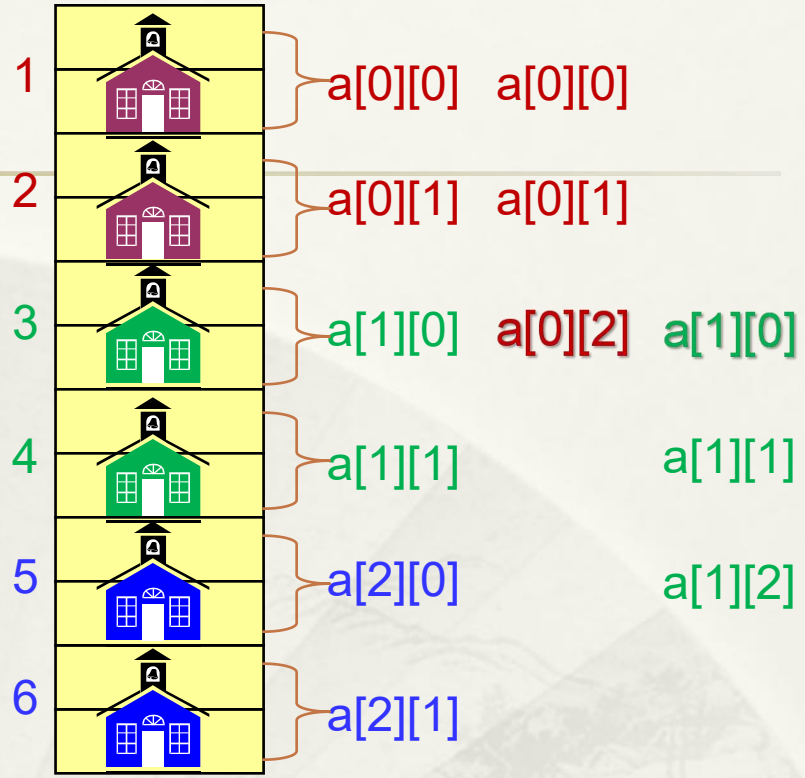


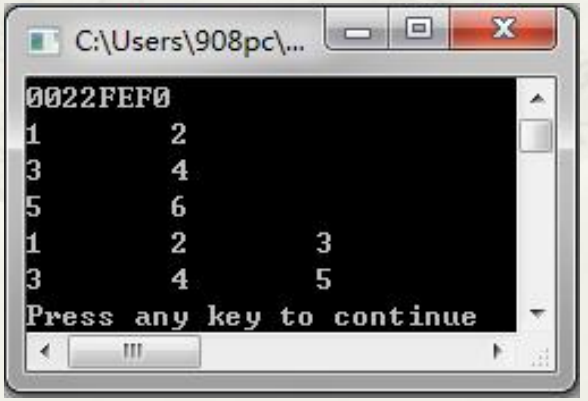
a[0][3]和a[1][0]指的是同一元素，不检查下标越界，a[0][3]的写法也合法，但隐患严重

```c
#include <stdio.h>
int main()
{
    int i, j;
    int a[3][2] = {1,2,3,4,5,6};
    printf("%p\n", a);

    for (i=0; i<3; i++)
    {
        for (j=0; j<2; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }

    for (i=0; i<2; i++)
    {
        for (j=0; j<3; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

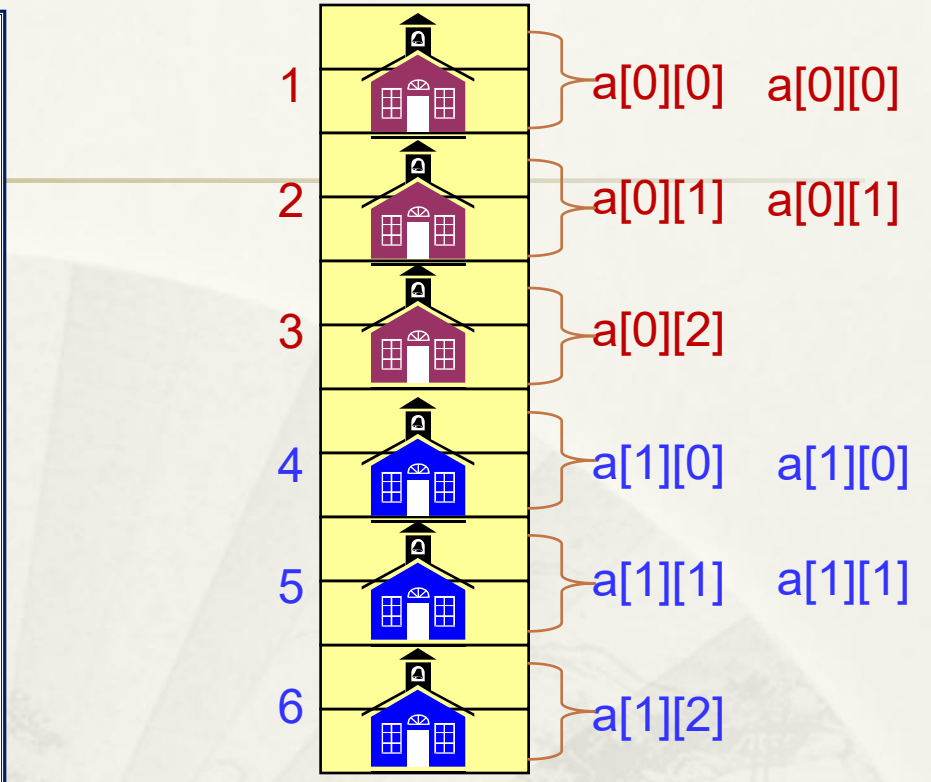


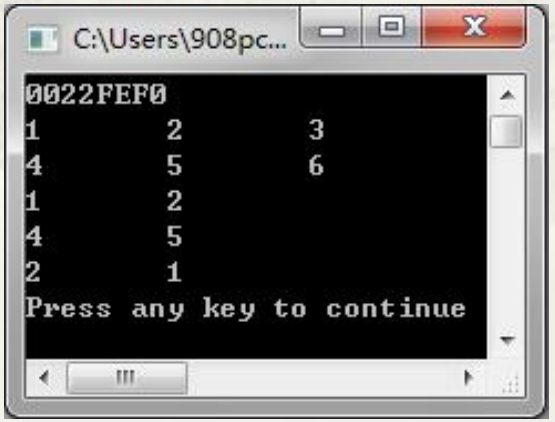


```
#include <stdio.h>
int main()
{
    int i, j;
    int a[2][3] = {1,2,3,4,5,6};
    printf("%p\n", a);

    for (i=0; i<2; i++)
    {
        for (j=0; j<3; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }

    for (i=0; i<3; i++)
    {
        for (j=0; j<2; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

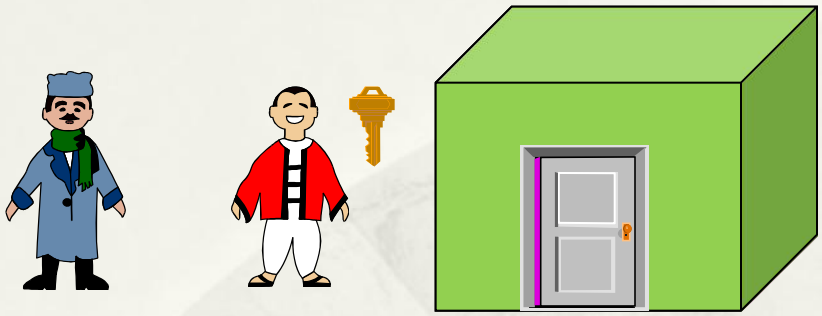




【例8.4】从键盘输入某年某月（包括闰年），编程输出该年的该月拥有的天数

```
1   #include <stdio.h>
2   #define  MONTHS 12
3   int main()
4   {
5       int days[2][MONTHS] = {{31,28,31,30,31,30,31,31,30,31,30,31},
6                              {31,29,31,30,31,30,31,31,30,31,30,31}};
7       int year, month;
8       do{
9           printf("Input year,month:");
10          scanf("%d,%d", &year, &month);
11      } while(month < 1 || month > 12);  /* 处理不合法数据的输入 */
12      if (((year%4 == 0) && (year%100 != 0)) || (year%400 == 0))/*闰年*/
13          printf("The number of days is %d\n", days[1][month-1]);
14      else  /*非闰年*/
15          printf("The number of days is %d\n", days[0][month-1]);
16      return 0;
17  }
```

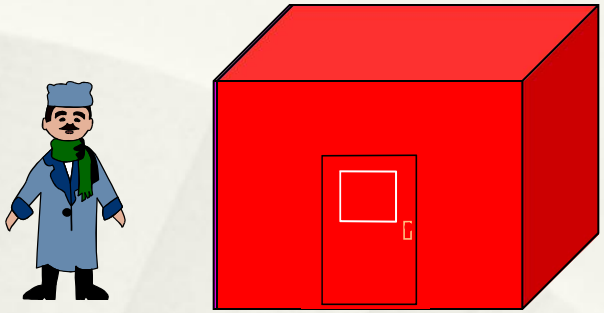
8.3 向函数传递一维数组



数组作函数参数

——按地址调用

传递数组的首地址，实参与形参数组占同一段内存单元



普通变量作函数参数

——按值调用

传递变量值的副本，实参与形参变量占不同的内存单元

8.3 向函数传递一维数组

安全否？

【例8.5】计算平均分：计数控制—键盘输入学生人数

```
1   #include <stdio.h>
2   #define N 40
3   int Average(int score[], int n);
4   void ReadScore(int score[], int n);
5   int main()
6   {
7       int score[N], aver, n;
8       printf("Input n:");
9       scanf("%d", &n);
10      ReadScore(score, n);
11      aver = Average(score, n);
12      printf("Average score is %d\n",aver);
13
14  }
```

用不带下标的数组名做函数实参

```
int Average(int score[], int n)
{
    int i, sum = 0;
    for (i=0; i<n; i++)
    {
        sum += score[i];
    }
    return  sum / n;
}
```

```
void ReadScore(int score[], int n)
{
    int i;
    printf("Input score:");
    for (i=0; i<n; i++)
    {
        scanf("%d", &score[i]);
    }
}
```

8.3 向函数传递一维数组

【例8.6】计算平均分：标记控制—负值作为输入结束标记

```
1   #include <stdio.h>
2   #define N 40
3   int Average(int score[], int n);
4   int ReadScore(int score[]);
5   int main()
6   {
7       int score[N], aver, n;
8       n = ReadScore(score);
9       printf("Total students are %d\n",n);
10      aver = Average(score, n);
11      printf("Average score is %d\n",aver);
12      return 0;
13  }
```

返回学生人数

```
int ReadScore(int score[])
{
    int i = -1;
    do{
        i++;
        printf("Input score:");
        scanf("%d", &score[i]);
    }while (score[i] >= 0);
    return i;
}
```

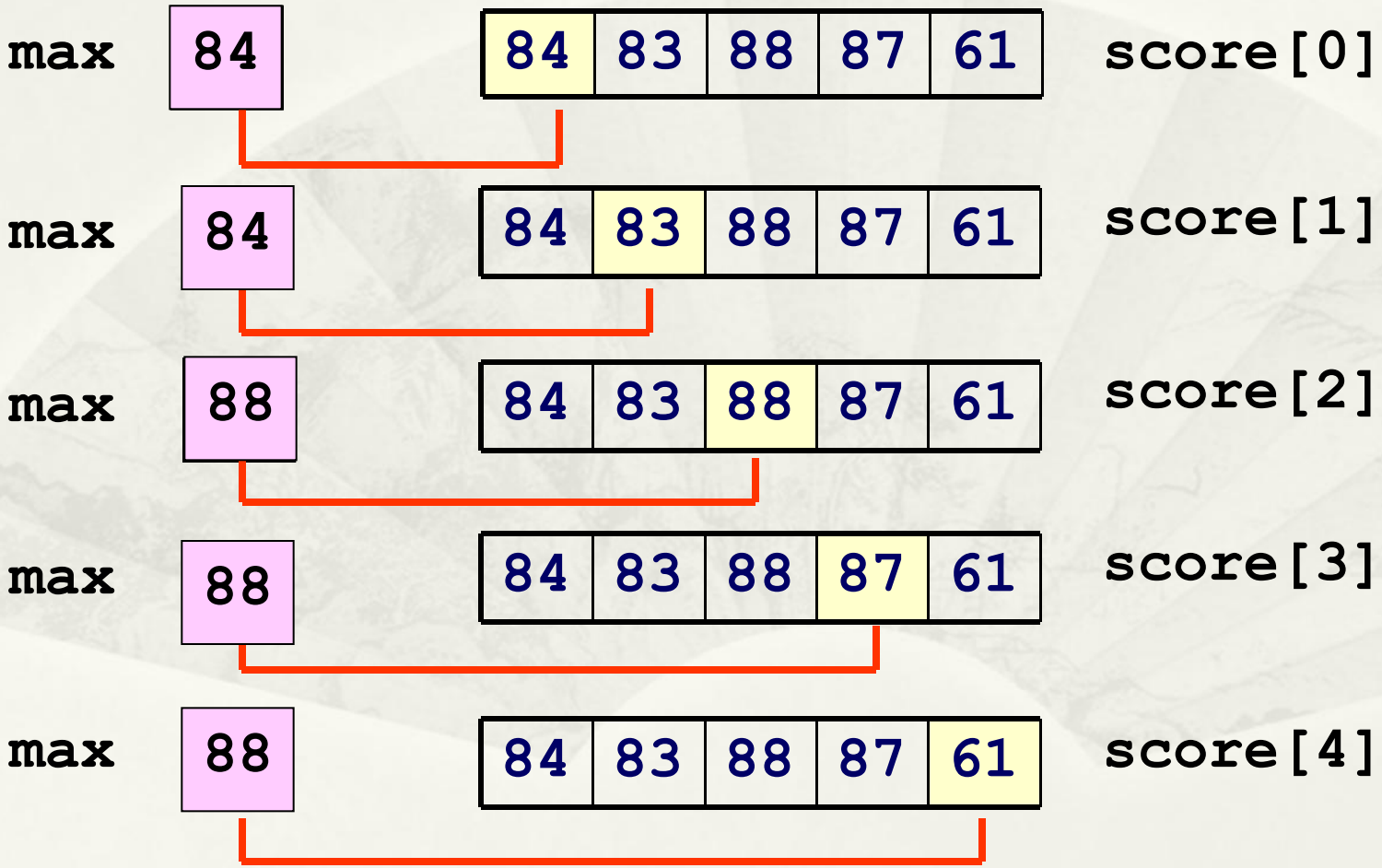
8.3 向函数传递一维数组

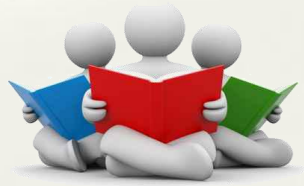
【例8.7】计算最高分

```
#include <stdio.h>
#define  N  40
int ReadScore(int score[]);
int FindMax(int score[], int n);
int main()
{
   int  score[N], max, n;
   n = ReadScore(score);
   printf("Total students are %d\n", n);
   max = FindMax(score, n);
   printf("The highest score is %d\n", max);
   return 0;
}
```

8.3 向函数传递一维数组

【例8.7】计算最高分 `int score[5]={84, 83, 88, 87, 61};`



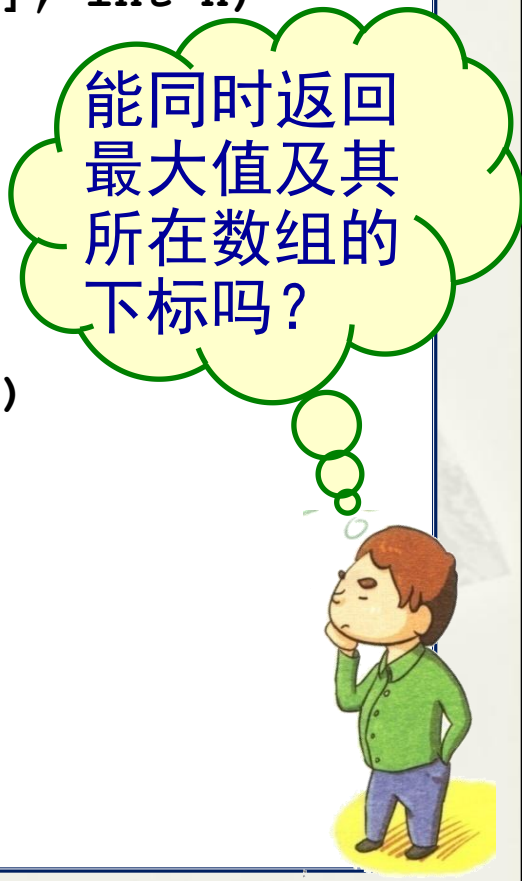


8.3 向函数传递一维数组

【例8.7】计算最高分

- 假设第一个学生成绩为最高
  `max = score[0];`
- 对所有学生成绩进行比较
  若 `score[i] > max`
  则令 `max = score[i]`
- 返回最高分`max`

```
int FindMax(int score[], int n)
{
   int max, i;
   max = score[0];
   for (i=1; i<n; i++)
   {
      if (score[i] > max)
      {
         max = score[i];
      }
    }
    return max;
}
```



8.3 向函数传递一维数组

【例8.7】计算最高分

- 假设第一个学生成绩为最高

  `maxNum = 0;`
- 对所有学生成绩进行比较

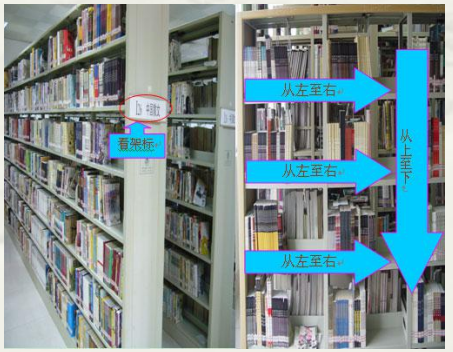
  若`score[i]>score[maxNum]`

  则令 `maxNum = i`
- 返回最高分所在的下标`maxNum`

```
int FindMax(int score[], int n)
{
  int maxNum, i;
  maxNum = 0;
  for (i=1; i<n; i++)
  {
    if (score[i] > score[maxNum])
    {
       maxNum = i;
    }
  }
  return maxNum;
}
```

8.4排序和查找

- 排序与查找是常用的操作
  - 对扑克牌进行排序
  - 青年歌手大奖赛成绩排名
  - 从图书馆的书架上查找某一本图书
  - 百度、谷歌等网站的搜索引擎就需要使用查找匹配算法完成搜索任务

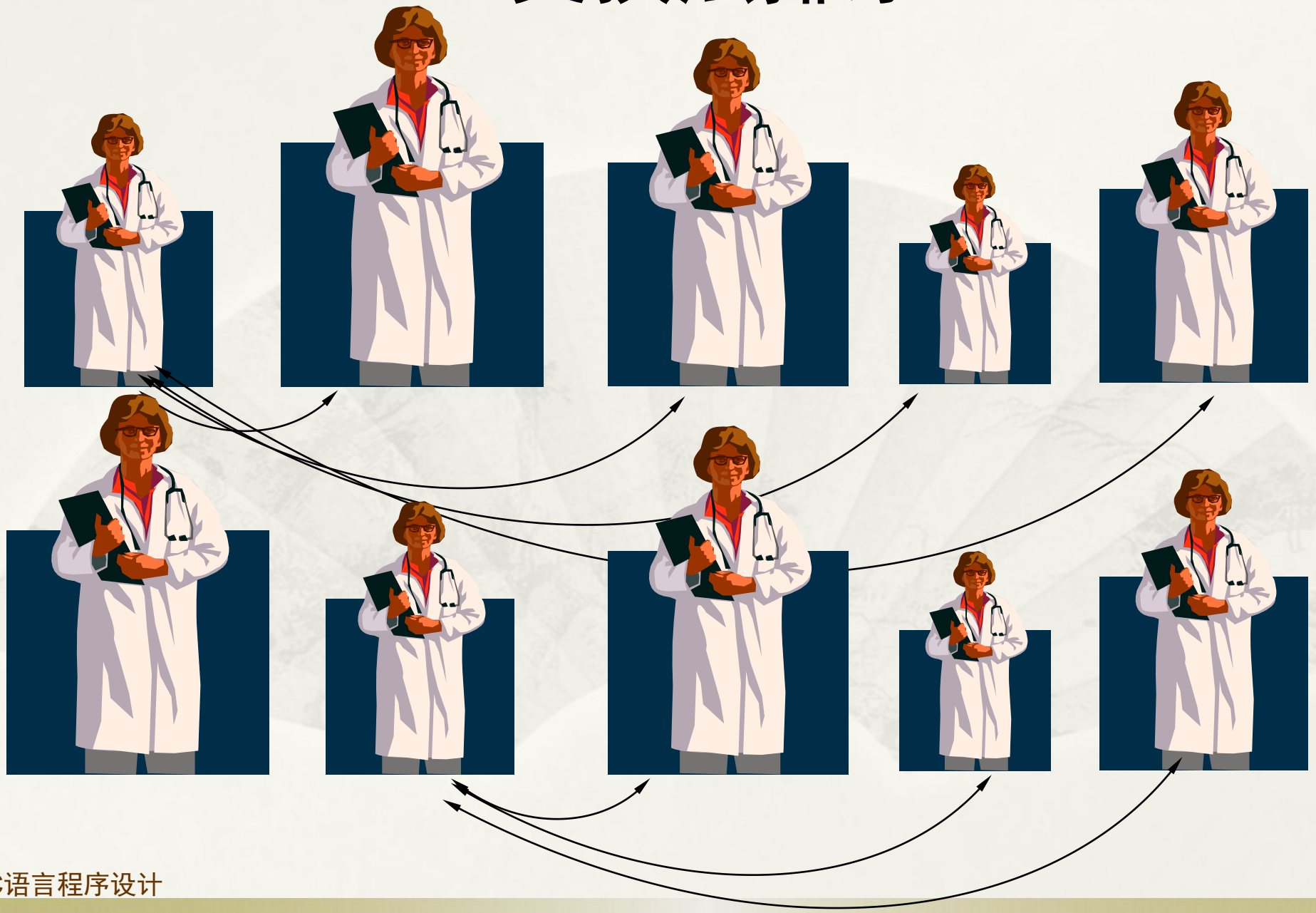


form http://www.baidu.com/

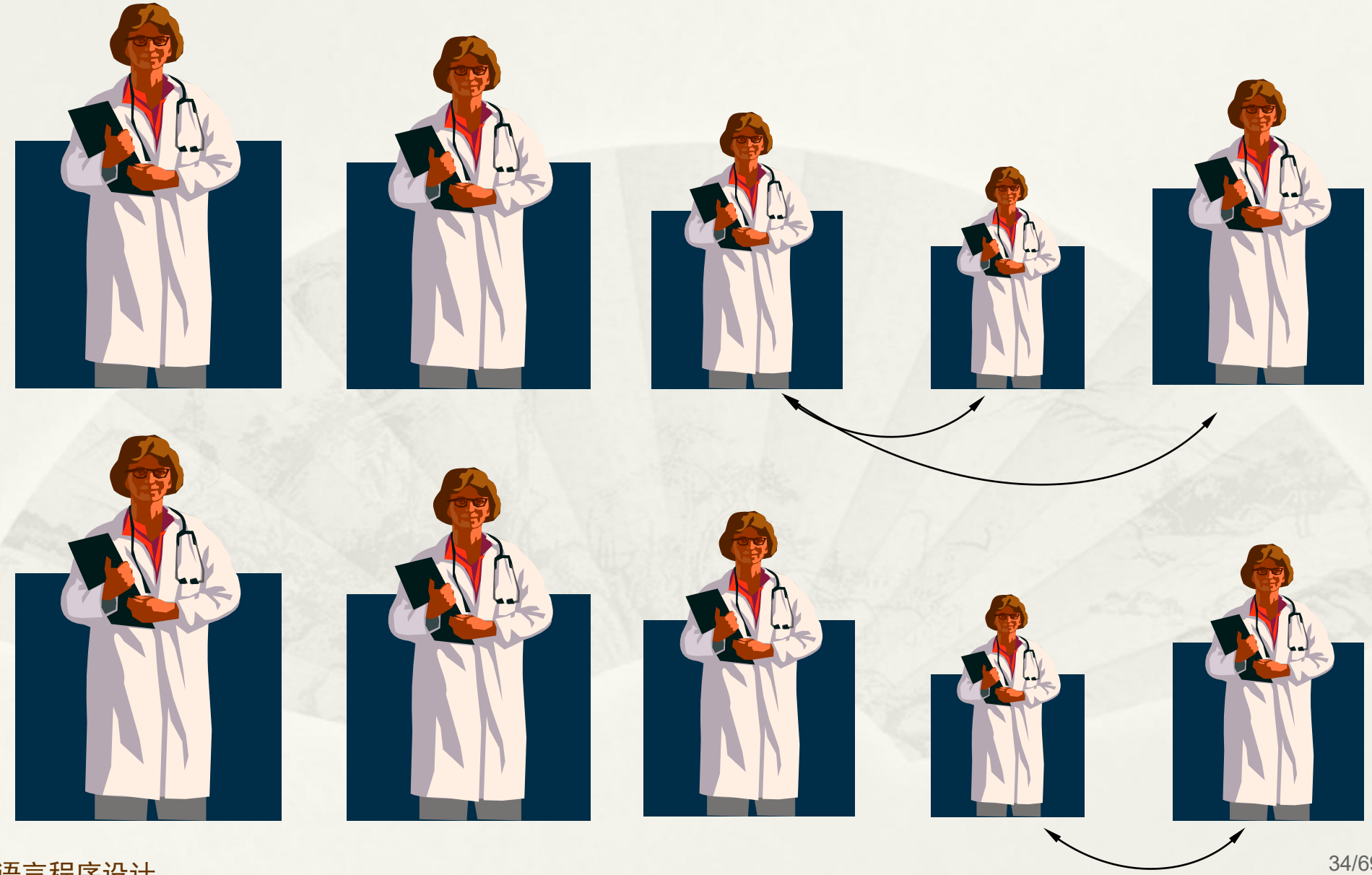
8.4排序和查找

- 排序（Sort）算法
  - 交换法排序（Exchange Sort）
  - 选择法排序（Selection Sort）
  - 冒泡排序（Bubble Sort）
  - 插入排序（Insertion Sort）

交换法排序

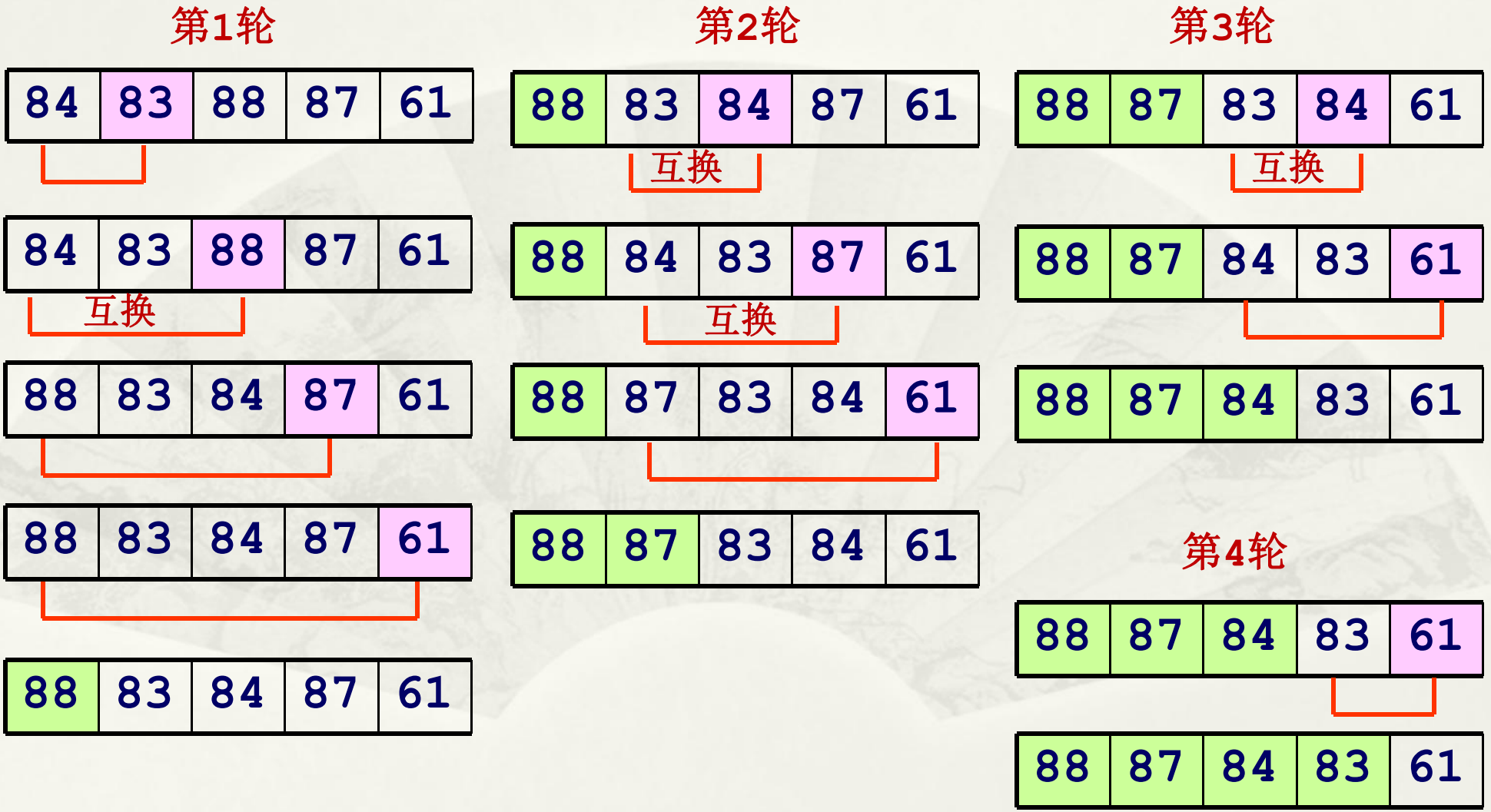


交换法排序



交换法排序

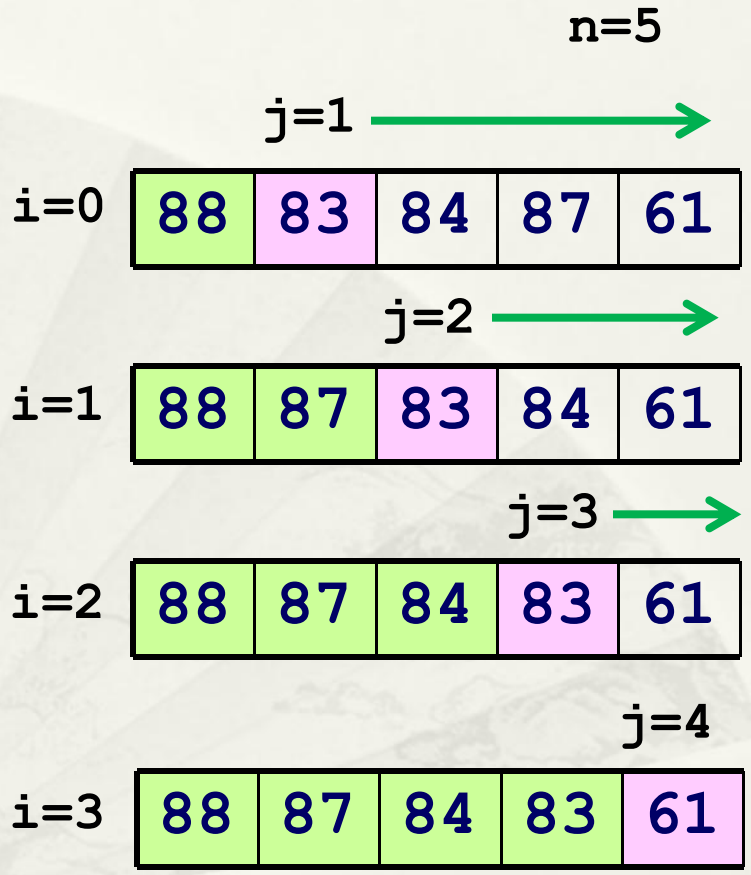
```
int a[5] = {84, 83, 88, 87, 61}; //Unsorted
```



交换法排序

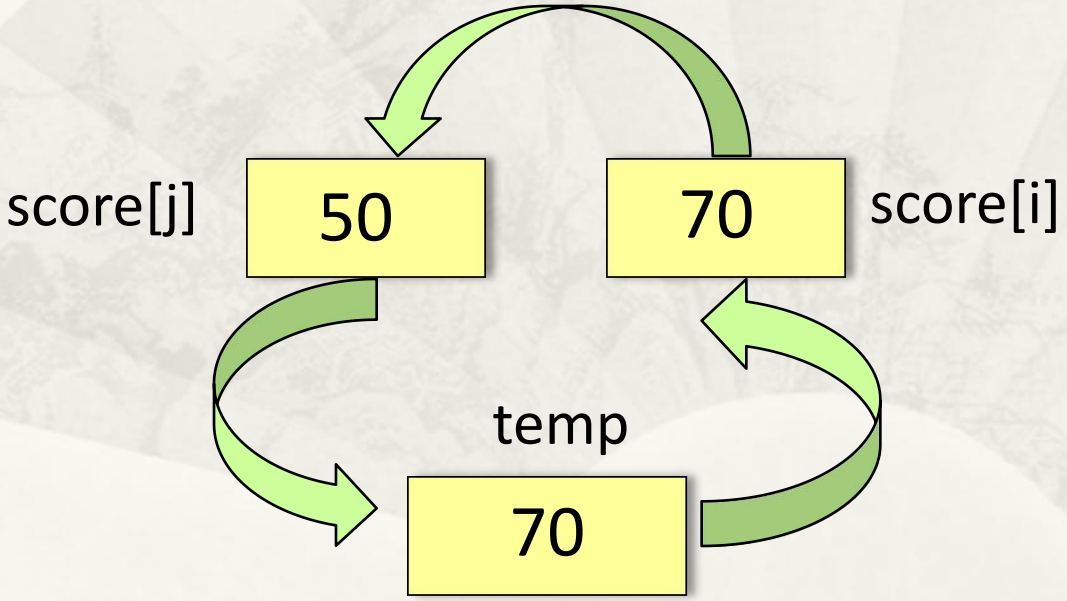
【例8.8】用交换法对成绩降序排序

```
for (i=0; i<n-1; i++)
{
    for (j=i+1; j<n; j++)
    {
       if (score[j] > score[i])
        "交换score[j]和score[i]"
    }
}
```



如何实现两数交换？

```
temp = score[j];
score[j] = score[i];
score[i] = temp;
```



如何实现两数交换？

- 能否不借助中间变量完成两数交换呢？
- 这个方法安全吗？

```
a = a + b;
b = a - b;
a = a - b;
```

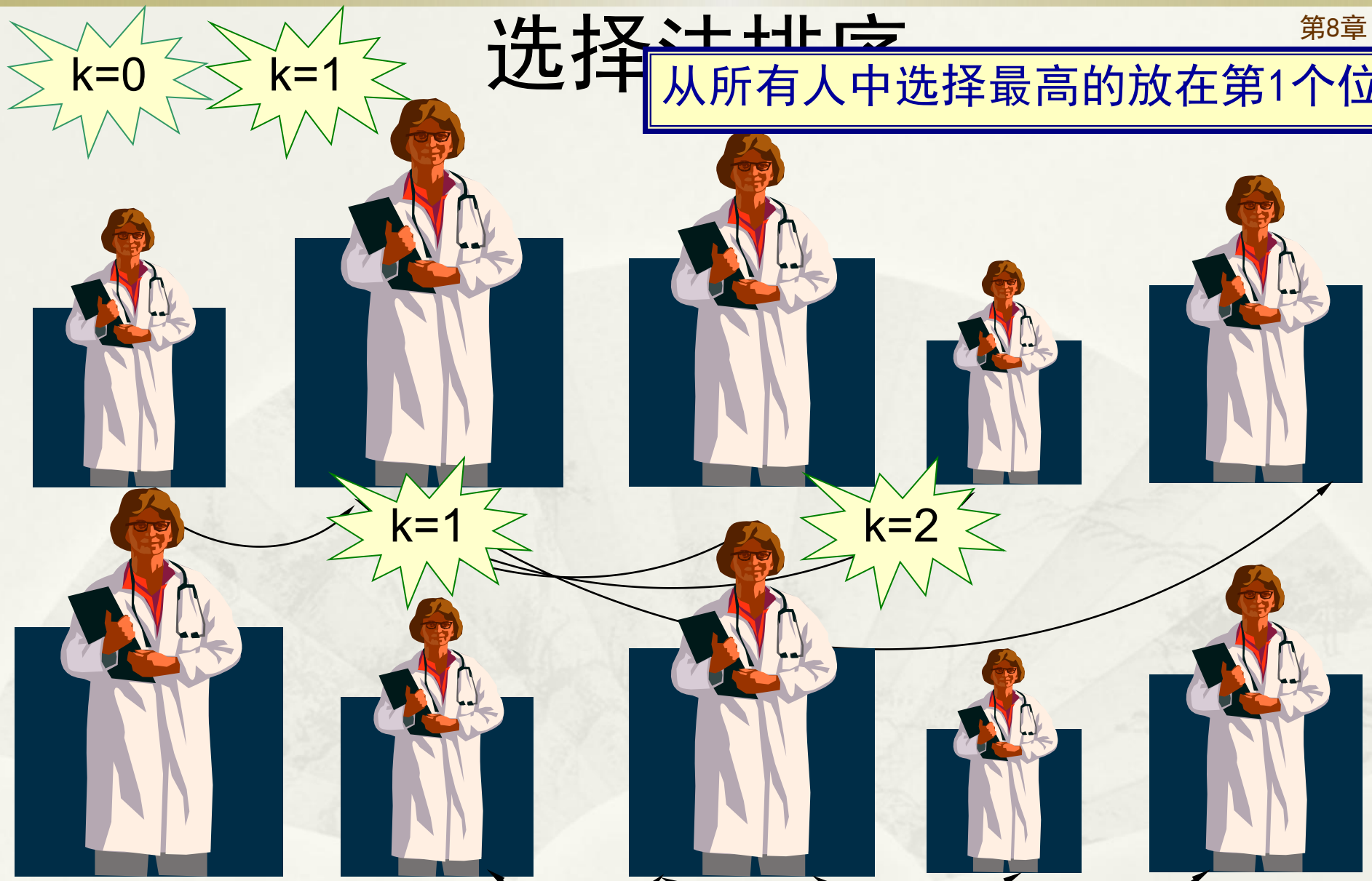
```
b = a * b;
a = b / a;
b = b / a;
```

【例8.8】用交换法对成绩降序排序

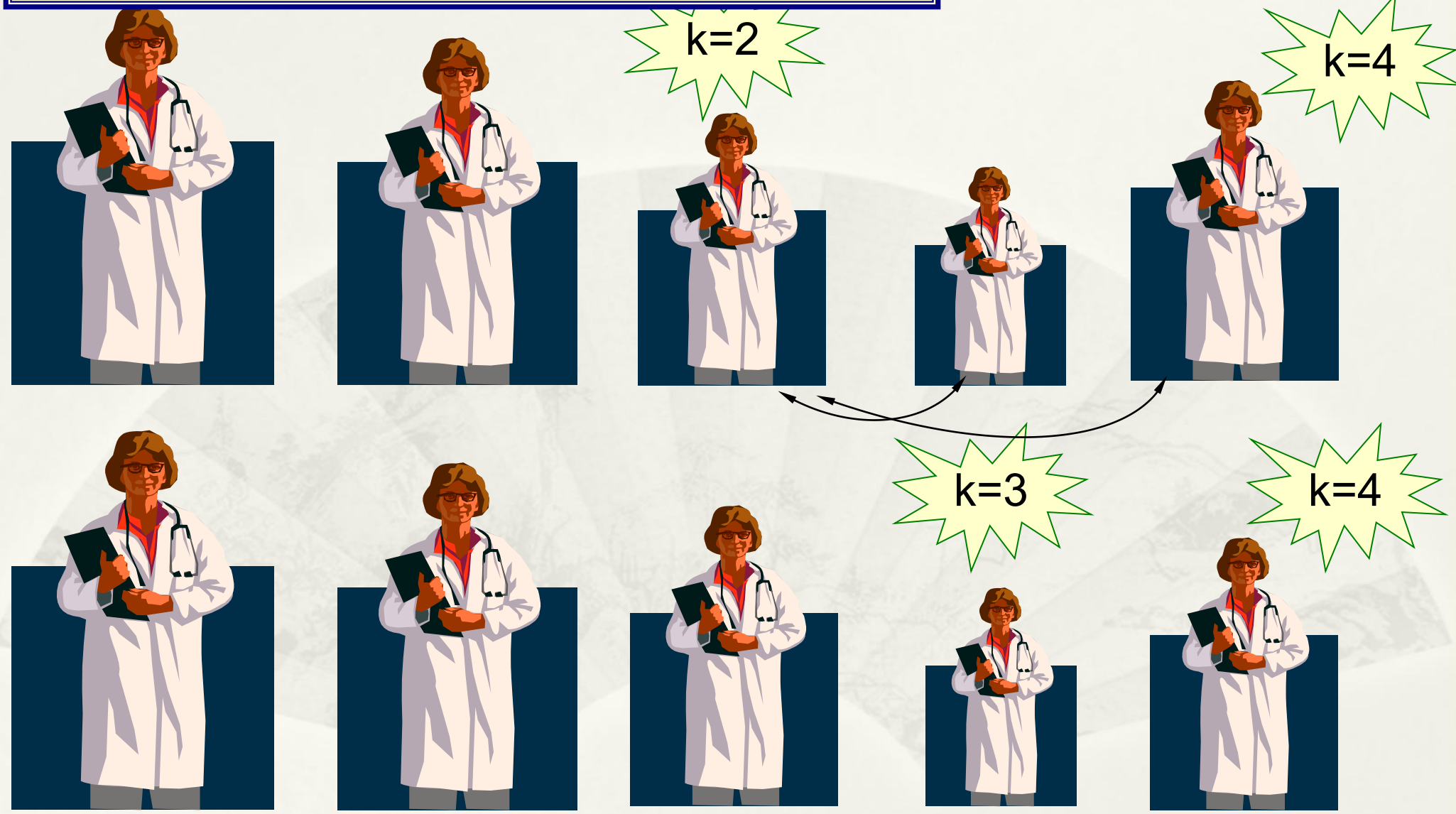
```
void DataSort(int score[], int n) /*交换法排序*/
{
   int i, j, temp;
   for (i=0; i<n-1; i++)
   {
       for (j=i+1; j<n; j++)
       {
           if (score[j] > score[i]) /*从高到低*/
           {
               temp = score[j];
               score[j] = score[i];
               score[i] = temp;
           }
       }
   }
}
```

成绩降序排序

选择法排序



从剩余的所有人中选择最高的放在第3个位置



从剩余的所有人中选择最高的放在第4个位置

选择法排序

```
int a[7] = {2, 9, 5, 4, 8, 1, 6}; //Unsorted
```

选择法排序

寻找最高分所在下标k的过程

```
for (i=0; i<n-1; i++)
{
    k = i;
    for (j=i+1; j<n; j++)
    {
        if (score[j] > score[k])
            记录此轮比较中最高分所在元素的下标   k = j;
    }
    若k中记录的最高分位置不在下标i处，则
        "交换成绩score[k]和score[i]"
        "交换学号num[k]和num[i]"
}
```

```
void DataSort(int score[], long num[], int n) /*选择法*/
{
    int i, j, k, temp1;
    long temp2;
    for (i=0; i<n-1; i++)
    {
        k = i;
        for (j=i+1; j<n; j++)
        {
            if (score[j] > score[k])
            {
                k = j;      /*记录最大数下标位置*/
            }
        }
        if (k != i)         /*若最大数不在下标位置i*/
        {
            temp1 = score[k];
            score[k] = score[i];
            score[i] = temp1;
            temp2 = num[k];
            num[k] = num[i];
            num[i] = temp2;
        }
    }
}
```

成绩降序排序

```
void DataSort(int score[], long num[], int n) /*选择法*/
{
    int i, j, k, temp1;
    long temp2;
    for (i=0; i<n-1; i++)
    {
        k = i;
        for (j=i+1; j<n; j++)
        {
            if (num[j] < num[k])
            {
                k = j;      /*记录最大数下标位置*/
            }
        }
        if (k != i)         /*若最大数不在下标位置i*/
        {
            temp1 = score[k];
            score[k] = score[i];
            score[i] = temp1;
            temp2 = num[k];
            num[k] = num[i];
            num[i] = temp2;
        }
    }
}
```

学号升序排序

8.4排序和查找

- 查找（Search）算法
  - 顺序查找，也称线性查找（Linear Search）
  - 折半查找，也称对分搜索（Binary Search）

顺序查找原理

Key　　List

Key	List							
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8

事先不必排序

哈，找到了！

在最坏情况下，查找次数等于总的数据量大小。
平均情况需要比较一半的数组元素。

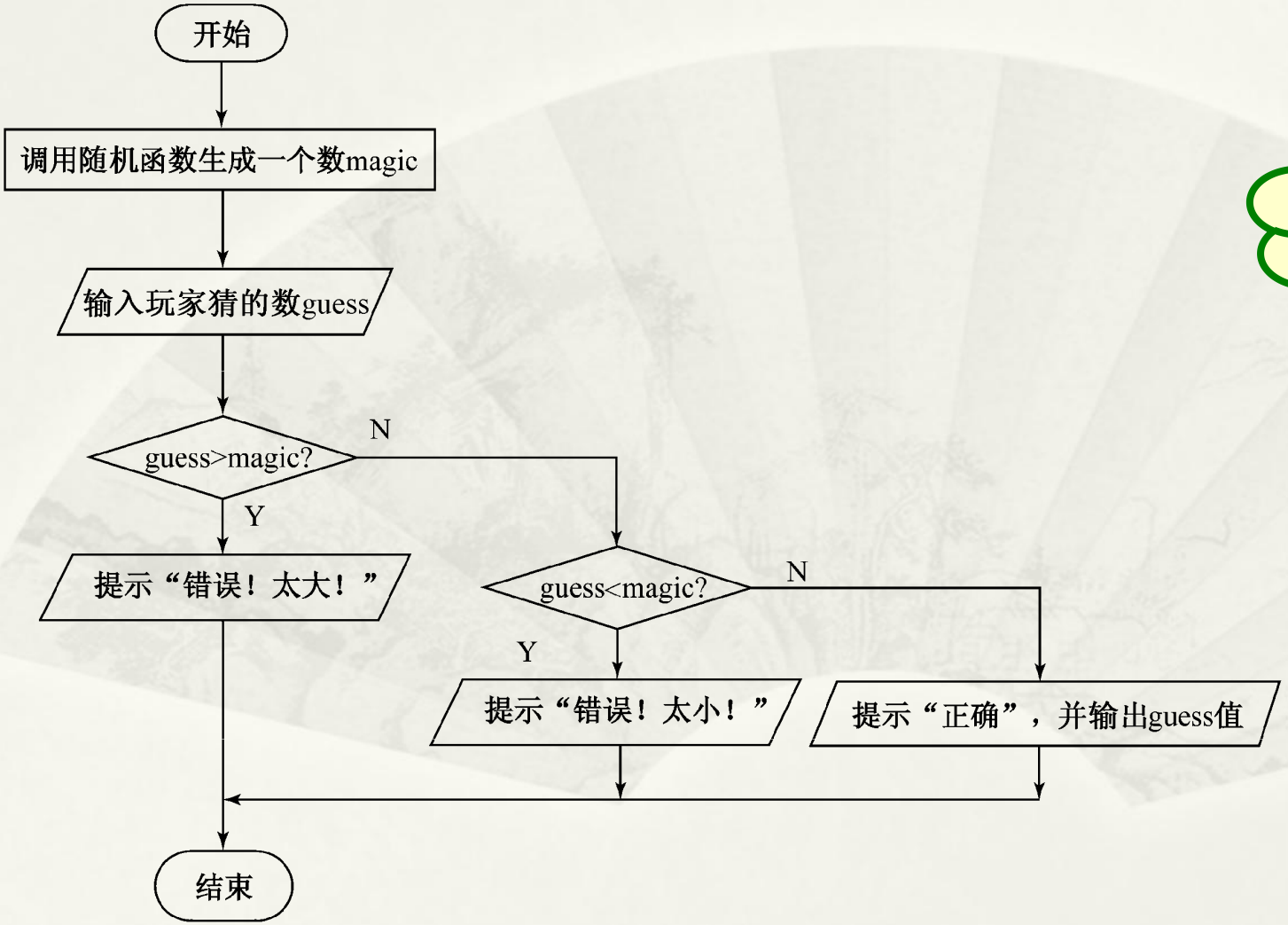
```
int  LinSearch(long num[], long x, int n)
{
    int i;
    for (i=0; i<n; i++)
    {
        if (num[i] == x)
        {
            return i;
        }
    }
    return -1;
}
```

找不到时返回-1

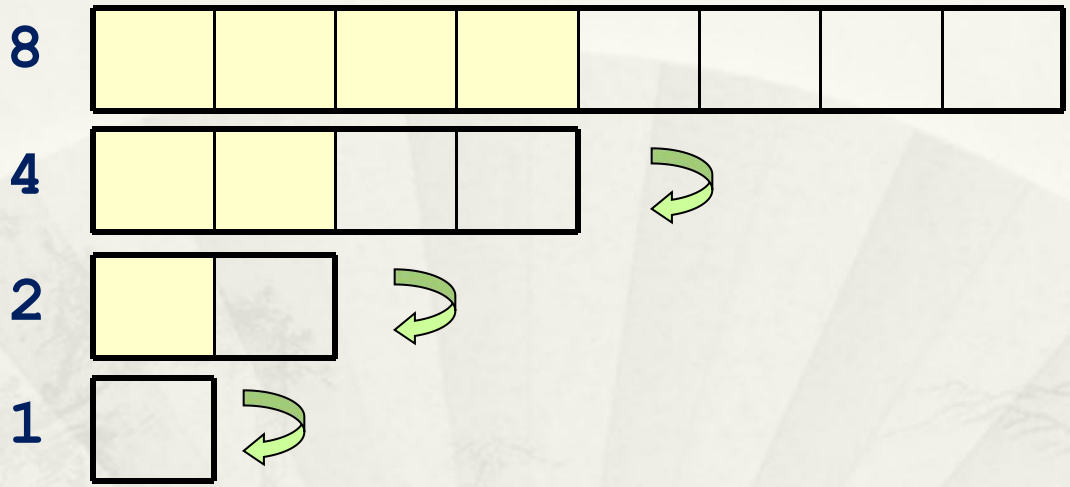
【例8.10】顺序查找某学号对应的学生的成绩

```
#define N 40
int main()
{
  int score[N], n, pos;
  long num[N], x;
  n = ReadScore(score, num);
  printf("Input the searching ID:");
  scanf("%ld", &x);
  pos = LinSearch(num, x, n);
  if (pos != -1)
  {
    printf("score=%d\n",score[pos]);
  }
  else
  {
    printf("Not found!\n");
  }
  return 0;
}
```

猜数游戏中的问题求解方法



折半查找

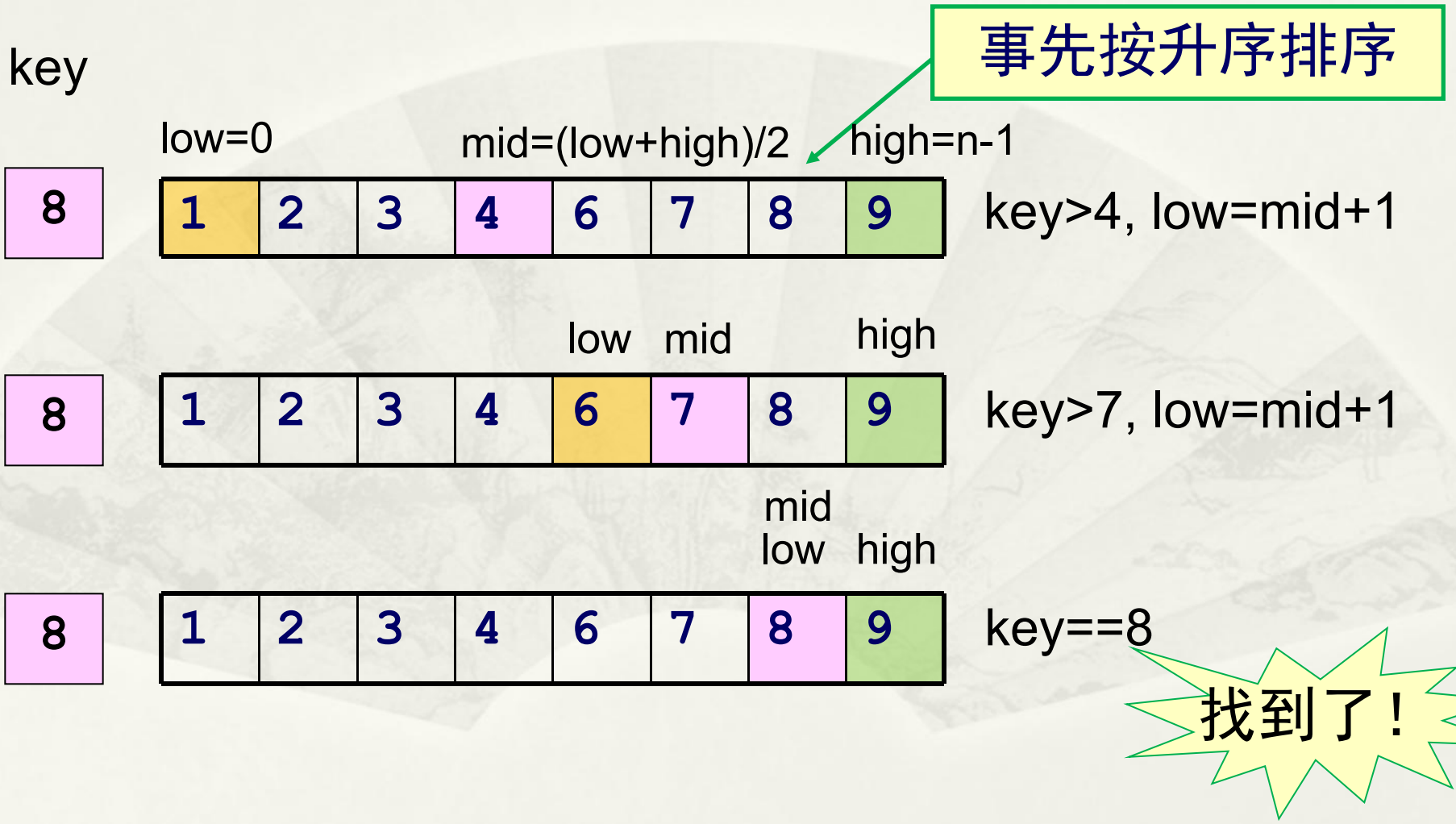


1024 ⇨512 ⇨256 ⇨128 ⇨64 ⇨32 ⇨16 ⇨ 8 ⇨4 ⇨2 ⇨1

用2除一次相当于折半查找算法中的一次比较。

最多所需的比较次数是第一个大于数组元素个数的2的幂次数。

折半查找原理


事先按升序排序
key
low=0
mid=(low+high)/2
high=n-1
8
1 2 3 4 6 7 8 9
key>4, low=mid+1
low mid high
8
1 2 3 4 6 7 8 9
key>7, low=mid+1
mid
low high
8
1 2 3 4 6 7 8 9
key==8
找到了！

折半查找原理

key


low
mid
high
9
1 2 3 4 6 7 10 12
key>4, low=mid+1
low mid high
9
1 2 3 4 6 7 10 12
key>7, low=mid+1
mid
low high
9
1 2 3 4 6 7 10 12
key<10, high=mid-1
high low
9
1 2 3 4 6 7 10 12
low<high为假
没找到！

【例8.11】折半查找学号

```
int  BinSearch(long num[], long x, int n)
{
   int  low, high, mid;
   low = 0;
   high = n - 1;
   while (low <= high)
   {
       mid = (high + low) / 2;
       if (x > num[mid])
       {
             low = mid + 1;
       }
       else  if (x < num[mid])
       {
             high = mid - 1;
       }
       else
       {
             return mid;
       }
   }
   return -1;
}
```

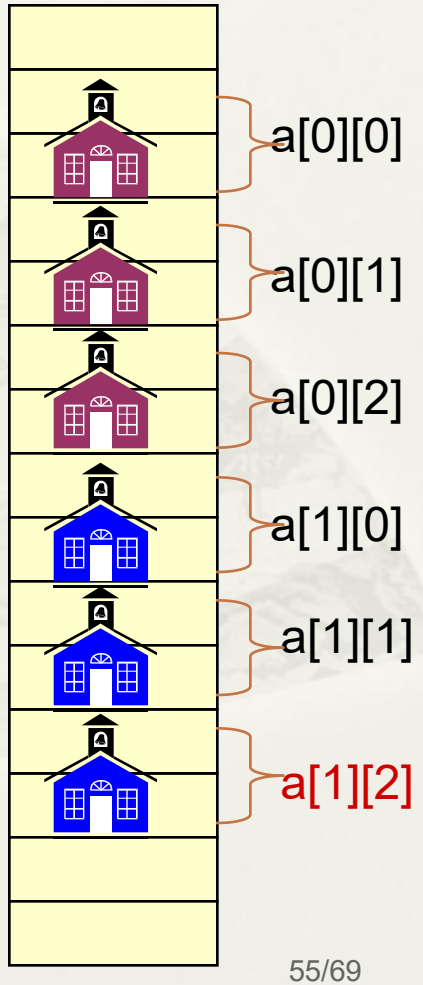
并非吹毛求疵，鸡蛋里挑骨头

- `mid = (high + low) / 2;`
- 如果数组很大，low和high之和大于有符号整数的极限值（在limits.h中定义）
  - 就会发生数值溢出，使mid成为一个负数
- 防止溢出的解决方案
  - 修改计算中间值的方法，用减法代替加法
  - `mid = low + (high - low) / 2;`

8.5向函数传递二维数组

```
short a[2][3];
```

向函数传递二维数组的首地址



- 在声明函数的二维数组形参时，为什么不能省略数组第二维的长度（列数）呢？
- 元素`a[i][j]`在数组中相对于第一个元素的位置：

```
i * 3 + j
```

元素地址：首地址+偏移量

偏移1*3+2个元素

数组的应用

- 保存n个学生一门课程的成绩
  - 用一维数组
  - `int Average(int score[], int n);`
  - 通常不指定数组的长度，用另一个形参来指定数组的大小
- 保存n个学生的m门课程的成绩
  - 用二维数组
  - `void Average(int score[][COURSE_N], float aver[], int n);`
  - 可省略数组第一维的长度，不能省略第二维的长度
  - 数组**aver**可保存每个学生的平均分，或每门课程的平均分

例8.12 计算每个学生的总分和平均分

```
void AverforStud(int score[][COURSE_N], int sum[],
                 float aver[], int n)
{
   int  i, j;

   for (i=0; i<n; i++)   //先遍历每个学生
   {
       sum[i] = 0;
       for (j=0; j<COURSE_N; j++) //遍历每门课程
       {
             sum[i] = sum[i] + score[i][j];
       }
       aver[i] = (float) sum[i] / COURSE_N;
   }
}
```

例8.12 计算每门课程的总分和平均分

```
void AverforCourse(int score[][COURSE_N], int sum[],
                   float aver[], int n)
{
   int  i, j;

   for (j=0; j<COURSE_N; j++) //先遍历每门课程
   {
       sum[j] = 0;
       for (i=0; i<n; i++)     //遍历每个学生
       {
             sum[j] = sum[j] + score[i][j];
       }
       aver[j] = (float) sum[j] / n;
   }
}
```

数组的其他应用

- 除了一维、二维数据表、矩阵运算等，还有其他应用吗？
- 如何判断一个整数x是否是素数（Prime Number）？
  - 不能被1和x以外的其他数整除的正整数
  - 试商法，用2~x-1之间的整数去试商看能否被整除
  - 用2~sqrt(x)之间的整数去试商看能否被整除即可

```
int IsPrime(int x)
{
    int i, flag = 1;
    int squareRoot = sqrt(x);
    if (x <= 1) flag = 0;
    for (i=2; i<=squareRoot && flag; i++)
    {
        if (x%i == 0)  flag = 0;
    }
    return flag;
}
```

求100以内的所有素数

```
int main()
{
    int i;
    for (i=1; i<=100; i++)
    {
        if (IsPrime(i)) printf("%d\n", i);
    }
    return 0;
}
```

```
int IsPrime(int x)
{
    int i, flag = 1;
    int squareRoot = sqrt(x);
    if (x <= 1) flag = 0;
    for (i=2; i<=squareRoot && flag; i++)
    {
        if (x%i == 0)  flag = 0;
    }
    return flag;
}
```

筛法求100以内的所有素数

初始化：令a[2]=2,a[3]=3,.....,a[N]=N (N值为100)

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

筛2的倍数

2	3	0	5	0	7	0	9	0	11	0	13	0	15	0	17

筛3的倍数

2	3	0	5	0	7	0	0	0	11	0	13	0	0	0	17

筛5的倍数

2	3	0	5	0	7	0	0	0	11	0	13	0	0	0	17

依次从a中筛掉2的倍数，3的倍数，5的倍数，.....，sqrt(N)的倍数；

既筛掉所有素数的倍数，直到a中仅剩下素数为止（剩下的不为0的数不是任何数的倍数）

筛法求100以内的所有素数

- 自顶向下、逐步求精设计算法
- step 1：设计总体算法
  - 初始化数组a，使a[2]=2, a[3]=3,....., a[N]=N
  - 对i=2,3,.....,sqrt(N)分别做：　“筛掉a中所有a[i]的倍数”
  - 输出数组中余下的数（a[i]!=0的数）
- step 2：对“筛掉a中所有的a[i]的倍数”求精
  - 对数组a中下标i后j对应的所有数分别做：
  - 若“该数a[j]是a[i]的倍数”，则“筛掉该数a[j]”

筛法求100以内的所有素数

- 自顶向下、逐步求精设计算法
- step 2：对“筛掉a中所有的a[i]的倍数”求精
  - 对数组a中下标i后j对应的所有数分别做：
  - 若“该数a[j]是a[i]的倍数”，则“筛掉该数a[j]”
- step 3：对若“该数a[j]是a[i]的倍数”，则“筛掉该数a[j]”求精

```
if (a[i]!=0 && a[j]!=0 && a[j]%a[i]==0)
{
    a[j] = 0;
}
```

筛法求100以内的所有素数

```
int i, j, a[N+1];
for (i=2; i<=N; i++)
{
   a[i] = i;
}
for (i=2; i<=sqrt(N); i++)
{
   for (j=i+1; j<=N; j++)
   {
       if (a[i]!=0 && a[j]!=0 && a[j]%a[i]==0)
       {
              a[j] = 0;
       }
   }
}
```

鲁智深吃馒头

- 据说，鲁智深一天中午匆匆来到开封府大相国寺，想蹭顿饭吃，当时大相国寺有99个和尚，只做了99个馒头
- 智清长老不愿得罪鲁智深，便把他安排在一个特定位置，之后对所有人说：
- 从我开始报数（围成一圈），第5个人可以吃到馒头（并退下）
- 退下的人的下一位开始从1报数，第5个人可以吃到馒头（并退下）
- 按此方法，所有和尚都吃到了馒头，唯独鲁智深没有吃上
- 请问他在那个位置？
- 能否借鉴筛法求出剩下的最后一个人的位置？

- 50位的n!计算？
- 为什么结果会这样？

```c
#include <stdio.h>
double  Fact(unsigned int  n);
int main()
{
    int  m;
    for (m=1; m<=40; m++)
    {
        printf("%d! = %.0f\n", m, Fact(m));
    }
    return 0;
}
double  Fact(unsigned int  n)
{
    unsigned int  i;
    double result = 1;
    for (i=2; i<=n; i++)
        result *= i;
    return result;
}
```

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 51090942171709440000
22! = 1124000727777607700000
23! = 25852016738884978000000
24! = 620448401733239410000000
25! = 15511210043330986000000000
26! = 403291461126605650000000000
27! = 10888869450418352000000000000
28! = 304888344611713840000000000000
29! = 8841761993739700800000000000000
30! = 265252859812191030000000000000000
31! = 8222838654177922400000000000000000
32! = 263130836933693520000000000000000000
33! = 8683317618811885900000000000000000000
34! = 295232799039604120000000000000000000000
35! = 10333147966386144000000000000000000000000
36! = 371993326789901180000000000000000000000000
37! = 13763753091226343000000000000000000000000000
38! = 523022617466601040000000000000000000000000000
39! = 20397882081197442000000000000000000000000000000
40! = 815915283247897680000000000000000000000000000000
```

- 挑战性的作业：习题8.18
- 挑战类型表示的极限——50位的n!计算？
  - 大数的存储问题

```
Enter a number to be calculated:
40
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = 51090942171709440000
22! = 1124000727777607680000
23! = 25852016738884976640000
24! = 620448401733239439360000
25! = 15511210043330985984000000
26! = 403291461126605635584000000
27! = 10888869450418352160768000000
28! = 304888344611713860501504000000
29! = 8841761993739701954543616000000
30! = 265252859812191058636308480000000
31! = 8222838654177922817725562880000000
32! = 263130836933693530167218012160000000
33! = 8683317618811886495518194401280000000
34! = 295232799039604140847618609643520000000
35! = 10333147966386144929666651337523200000000
36! = 371993326789901217467999448150835200000000
37! = 13763753091226345046315979581580902400000000
38! = 523022617466601111760007224100074291200000000
39! = 20397882081197443358640281739902897356800000000
40! = 815915283247897734345611269596115894272000000000
```

小结

- 如何定义数组
- 如何向函数传递一维数组
  - 传递数组的首地址，形参数组和实参数组共享同一段内存
  - 通常不指定数组的长度，用另一个形参来指定数组的大小
- 如何向函数传递二维数组
  - 可省略数组第一维的长度，不能省略第二维的长度
- 常用算法：排序、查找、求最值


Thank you for
your attention !
SuXiaoHong


Q&A

抖音

抖音号: dydkuq2u6f2z

论程序员的罗辑思维有多强大……