

## § 5. 利用数组处理批量数据

### 5.1. 基本概念

#### 5.1.1. 引入

在表示若干相同类型、含义的元素时，引入数组，避免了用多个变量表示的不统一性

#### 5.1.2. 数组的组成

数组名+下标

(标识在整个数组中所处的位置，即序号)

#### 5.1.3. 要求

数组中元素的类型相同

例3: 输入3个人的成绩，先打印平均值再打印成绩

```
int main()
{
    int s1, s2, s3;
    cin >> s1;    //假设输入正确
    cin >> s2;    //假设输入正确
    cin >> s3;    //假设输入正确
    cout << "avg=" << (s1+s2+s3)/3.0 << endl;
    cout << "第1个:" << s1 << endl;
    cout << "第2个:" << s2 << endl;
    cout << "第3个:" << s3 << endl;
    return 0;
}
```

输入/输出无法  
用循环, 因为  
每次变量不同

10000人 ?  
人数不定 ?

例1: 输入3个人的成绩，打印平均值

```
int main()
{
    int i, s, sum = 0;
    for(i=0; i<3; i++) {
        cin >> s;    //假设输入都正确
        sum += s;
    }
    cout << "avg=" << sum/3.0 << endl;
    return 0;
}
```

S中只保留了  
最后一次的输入

例2: 输入3个人的成绩，打印每个人的成绩及平均值

```
int main()
{
    int i, s, sum = 0;
    for(i=0; i<3; i++) {
        cin >> s;    //假设输入都正确
        cout << "第" << i+1 << "个=" << s << end;
        sum += s;
    }
    cout << "avg=" << sum/3.0 << endl;
    return 0;
}
```

S中只保留了  
最后一次的输入

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.1. 定义

数据类型 数组名[正整型常量表达式]

```
int student[10];
```

```
long a[15*2];
```

★ 包括整型常量、整型符号常量和整型只读变量

```
#define N 10      const int n=10;
```

```
int a[N]; ✓      int a[n]; ✓
```

```
int k=10;
```

```
int a[k]; ✗
```

- 早期C/C++标准中，数组大小必须是常量，新的标准已允许为变量
- 为统一，此处仍要求为常量

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int n;
    cin >> n;
    int a[n];
}
```

VS2017编译	: ✗
其余三编译器	: ✓

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.2. 使用

- ★ 必须先定义，后使用，数组名的命名规则同变量名
- ★ 数组的大小在声明时应确定，不能动态定义，在内存中顺序存放，占用一块连续的空间
- ★ 若数组定义中整型常量表达式为n，表示有n个元素，则称数组长度应为n
- ★ 每个数组元素在内存中占用一个数据类型所占用的字节数，数组元素的表示方法为 **数组名[下标]**，下标范围从 **[0..n-1]**，表示为整型常量或整型表达式

```
#include <iostream>
using namespace std;
int main()
{ long a[10];
  cout << sizeof(a) << endl;
  cout << sizeof(a[3]) << endl;
}
```

40  
4

例：若有数组定义long a[10]  
则：数组长度为10  
数组下标表示范围[0..9]  
每个数组元素占4个字节  
sizeof(long)=4  
数组共占用内存中连续的40个字节  
内存存放为：

a[0]	2000 2003	
a[1]	2004 2007	
a[2]	2008 2011	
a[3]	2012 2015	
a[4]	2016 2019	
a[5]	2020 2023	
a[6]	2024 2027	
a[7]	2028 2031	
a[8]	2032 2035	
a[9]	2036 2039	

- ★ 即使允许使用变量定义数组，数组定义后大小仍为固定值，定义变量值的改变不影响数组

```
//用除VS2017以外的三编译器编译运行
#include <iostream>
using namespace std;
int main()
{
  int n;
  cin >> n; //假设输入15
  int a[n];
  cout << sizeof(a) << endl;
  n = 10;
  cout << sizeof(a) << endl;
  return 0;
}
```

60

60

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.1. 定义

#### 5.2.2. 使用

★ 数组的使用只能逐个引用其中元素，不能整体使用，引用时数组下标的表示为**常量或带变量的表达式**

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

```
a[0]=15;
```

```
float c; long d; c+d*a[5];
```

```
int k=3; a[k*4-3]=18;
```

```
10+a[0]+a[1]-a[2];
```

```
int a[5], i;
```

```
✗ cout << a; printf("%d", a);
```

```
✗ printf("%d%d%d%d%d", a);
```

```
✓ cout<<a[0]<<a[1]<<a[2]<<a[3]<<a[4];
```

```
for(i=0;i<5;i++)
```

```
✓ cout << a[i];
```

错误是指无法得到预期结果，编译本身没错，得到的是另外值(地址)



```
#include <iostream>
using namespace std;
int main()
{ int a[10], i;
  for(i=0; i<10; i++)
    cout << a[i] << endl;
  cout << endl;
  cout << a << endl;
  return 0;
}
```

10个不确定值  
一个6/8位的16进制数  
(VS2017下值每次不定)

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.1. 定义

#### 5.2.2. 使用

★ 数组的使用只能逐个引用其中元素，不能整体使用，引用时数组下标的表示为**常量或带变量的表达式**

★ 引用时，若数组下标超出范围，C/C++不会报错，因此编程者要控制下标在合理的范围内

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

```
3*5+a[10];
```

 × (引用操作)

```
a[5+2*3]=16;
```

 × (修改操作)

注意：编译都不会报错，是执行时错误  
=> 数组下标越界即使执行时不错，也是严重错误

```
#include <iostream>
using namespace std;
int main()
{ int k, a[10];
  k = 10+a[10]*2;
  cout << k << endl;
} //能执行，不确定值
```

```
#include <iostream>
using namespace std;
int main()
{ int k, a[10];
  k = 10+a[1000]*2;
  cout << k << endl;
} //不确定值 or 被系统终止
```

```
#include <iostream>
using namespace std;
int main()
{ int a[10];
  a[10] = 123;
  cout << a[10] << endl;
} //输出123后被系统终止
```

```
#include <iostream>
using namespace std;
int main()
{ int a[10];
  a[1000] = 123;
  cout << a[1000] << endl;
} //直接被操作系统终止
```

VS2017下执行的错误表现  
其余编译器自行观察

```
#include <iostream>
using namespace std;

int main()
{
  int k=321, a[10];
  cout << "k=" << k << endl;
  a[12] = 123;
  cout << "a[12]=" << a[12] << endl;
  cout << "k=" << k << endl;
}
```

VS2017下：  
a[12]实际占了k的位置，  
但系统不报错，访问a[12]  
也正确，但实际是严重错误  
换为a[11]后观察其它编译器

VS2017的结果  
k=321  
a[12]=123  
k=123

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.3. 数组在定义时初始化

##### 5.2.3.1. 初始化的作用

保证使用前有一个确定值

##### 5.2.3.2. 全部初始化

A. `int a[10]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};` →

a[0]	2000 2003	1
a[1]	2004 2007	2
a[2]	2008 2011	3
a[3]	2012 2015	4
a[4]	2016 2019	5
a[5]	2020 2023	6
a[6]	2024 2027	7
a[7]	2028 2031	8
a[8]	2032 2035	9
a[9]	2036 2039	10

`int a[10]={7, -3, 9, 64, 76, -23, 78, 123, -76, 263};` →

a[0]	2000 2003	7
a[1]	2004 2007	-3
a[2]	2008 2011	9
a[3]	2012 2015	64
a[4]	2016 2019	76
a[5]	2020 2023	-23
a[6]	2024 2027	78
a[7]	2028 2031	123
a[8]	2032 2035	-76
a[9]	2036 2039	263

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

错误，因为初始化的个数超过了数组的大小

B. `int a[]={0, 1, 2, 3, 4, 5, 6, 7, 8, 9};`

数组长度自动定义为10（初始化元素的个数）

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.3. 数组在定义时初始化

##### 5.2.3.1. 初始化的作用

##### 5.2.3.2. 全部初始化

##### 5.2.3.3. 部分初始化

```
int a[10]={5, -2, 7};
```

$a[0]=5$   $a[1]=-2$   $a[2]=7$

$a[3]-a[9]$  自动为0

```
int a[1000]={0}; //全0
```

长度不可省略

```
int a[1000]={1}; //a[0]为1, 其余全0
```

注意: 不是全1

```
int a[1000]; 希望a[0] - a[999] 为 0 - 999
```

方法1: `int a[1000] = {0, 1, 2, ..., 999};` 语句太长, 一般不用

方法2: `int a[1000], i;`

```
for(i=0; i<1000; i++)
```

```
    a[i] = i;
```

a[0]	2000 2003	5
a[1]	2004 2007	-2
a[2]	2008 2011	7
a[3]	2012 2015	0
a[4]	2016 2019	0
a[5]	2020 2023	0
a[6]	2024 2027	0
a[7]	2028 2031	0
a[8]	2032 2035	0
a[9]	2036 2039	0

不是定义时初始化, 而是通过  
执行语句进行赋值操作

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.3. 数组在定义时初始化

##### 5.2.3.1. 初始化的作用

##### 5.2.3.2. 全部初始化

##### 5.2.3.3. 部分初始化

★ 若一个都不初始化，则数组元素的值，当数组为自动变量时：**不确定**  
当数组为静态局部、静态全局、外部全局：**0**

```
int a[1000];    全部为0  
main()  
{   int b[100]; 不确定  
}
```



## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.4. 应用

思考:

- 1、人数不定如何处理?
- 2、如何检查输入的合理性?

例3: 输入3个人的成绩, 先打印平均值再打印成绩

```
int main()
{
    int s1, s2, s3;
    cin >> s1;    //假设输入正确
    cin >> s2;    //假设输入正确
    cin >> s3;    //假设输入正确
    cout << "avg=" << (s1+s2+s3)/3.0 << endl;
    cout << "第1个:" << s1 << endl;
    cout << "第2个:" << s2 << endl;
    cout << "第3个:" << s3 << endl;
    return 0;
}
```

输入/输出无法  
用循环, 因为  
每次变量不同

例3: 输入3个人的成绩, 先打印平均值再打印成绩

```
int main()
{
    int s[3], i, sum=0;
    for (i=0; i<3; i++) {
        cin >> s[i];    //假设输入正确
        sum += s[i];
    }
    cout << "avg=" << sum/3.0 << endl;
    for (i=0; i<3; i++)
        cout << "第" << i+1 << "个:" << s[i] << endl;
    return 0;
}
```

循环方式

例3: 输入3个人的成绩, 先打印平均值再打印成绩

```
#define NUM 10000    // const int NUM=10000;
int main()
{
    int s[NUM], i, sum=0;
    for (i=0; i<NUM; i++) {
        cin >> s[i];    //假设输入正确
        sum += s[i];
    }
    cout << "avg=" << sum/float(NUM) << endl;
    for (i=0; i<NUM; i++)
        cout << "第" << i+1 << "个:" << s[i] << endl;
    return 0;
}
```

10000人, 更合理  
方便修改及维护

例3: 输入3个人的成绩, 先打印平均值再打印成绩

```
int main()
{
    int s[10000], i, sum=0;
    for (i=0; i<10000; i++) {
        cin >> s[i];    //假设输入正确
        sum += s[i];
    }
    cout << "avg=" << sum/10000.0 << endl;
    for (i=0; i<10000; i++)
        cout << "第" << i+1 << "个:" << s[i] << endl;
    return 0;
}
```

10000人

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.4. 应用

P.127 例5.2 数组求斐波那契数列

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    int i;
    int f[20]={1,1};
    for(i=2; i<20; i++)
        f[i]=f[i-1]+f[i-2];
    for(i=0; i<20; i++) {
        if (i%5==0)
            cout << endl;
        cout << setw(8) << f[i];
    }
    cout << endl;
    return 0;
}
```

空行

1	1	2	3	5
8	13	21	34	55
89	144	233	377	610
987	1597	2584	4181	6765

和P.78 例3.13相比

1、3.13边计算边输出

本例先计算后输出

2、3.13输出后值不保留

本例值全部保留

如果不希望空第一行，  
如何实现？

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.4. 应用

P. 127-128 例5.3 冒泡法排序

```
int main()
{
    .....
    for(j=0; j<9; j++)
    {
        for(i=0; i<9-j; i++)
        {
            if (a[i] > a[i+1]) {
                t=a[i];
                a[i]=a[i+1];
                a[i+1]=t;
            }
        }
    }
    .....
}
```

两数  
交换  
方法

P. 128的程序定义了a[11]，但  
只用a[1]-a[10]，a[0]废弃  
本程序用a[0]-a[9]，有所不同  
for (j=1; j<=9; j++)  
 for (i=1; i<=10-j; i++)

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.4. 应用

P. 127-128 例5.3 冒泡法排序

```
int main()
{
    .....
    for(j=0; j<9; j++)
    {
        for(i=0; i<9-j; i++)
        {
            if (a[i] > a[i+1]) {
                t=a[i];
                a[i]=a[i+1];
                a[i+1]=t;
            }
        }
    }
    .....
}
```

两数  
交换  
方法

P. 128的程序定义了a[11]，但  
只用a[1]-a[10]，a[0]废弃  
本程序用a[0]-a[9]，有所不同  
for (j=1; j<=9; j++)  
for (i=1; i<=10-j; i++)

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次，外循环j=0

内循环 0 1 2 3 4 5 6 7 8

0	3
1	7
2	29
3	23
4	12
5	82
6	1
7	72
8	8
9	5

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.4. 应用

P. 127-128 例5.3 冒泡法排序

```
int main()
{
    .....
    for(j=0; j<9; j++)
    {
        for(i=0; i<9-j; i++)
        {
            if (a[i] > a[i+1]) {
                t=a[i];
                a[i]=a[i+1];
                a[i+1]=t;
            }
        }
    }
    .....
}
```

两数  
交换  
方法

P. 128的程序定义了a[11]，但  
只用a[1]-a[10]，a[0]废弃  
本程序用a[0]-a[9]，有所不同  
for (j=1; j<=9; j++)  
for (i=1; i<=10-j; i++)

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次，外循环j=0

内循环 0 1 2 3 4 5 6 7 8

0	3	-
1	7	-
2	29	
3	23	
4	12	
5	82	
6	1	
7	72	
8	8	
9	5	

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.4. 应用

P. 127-128 例5.3 冒泡法排序

```
int main()
{
    .....
    for(j=0; j<9; j++)
    {
        for(i=0; i<9-j; i++)
        {
            if (a[i] > a[i+1]) {
                t=a[i];
                a[i]=a[i+1];
                a[i+1]=t;
            }
        }
    }
    .....
}
```

两数  
交换  
方法

P. 128的程序定义了a[11]，但  
只用a[1]-a[10]，a[0]废弃  
本程序用a[0]-a[9]，有所不同  
for (j=1; j<=9; j++)  
for (i=1; i<=10-j; i++)

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次，外循环j=0

内循环 0 1 2 3 4 5 6 7 8

0	3	-
1	7	- -
2	29	-
3	23	
4	12	
5	82	
6	1	
7	72	
8	8	
9	5	

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.4. 应用

P. 127-128 例5.3 冒泡法排序

```
int main()
{
    .....
    for(j=0; j<9; j++)
    {
        for(i=0; i<9-j; i++)
        {
            if (a[i] > a[i+1]) {
                t=a[i];
                a[i]=a[i+1];
                a[i+1]=t;
            }
        }
    }
    .....
}
```

两数  
交换  
方法

P. 128的程序定义了a[11]，但  
只用a[1]-a[10]，a[0]废弃  
本程序用a[0]-a[9]，有所不同  
for (j=1; j<=9; j++)  
for (i=1; i<=10-j; i++)

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次，外循环j=0

内循环 0 1 2 3 4 5 6 7 8

0	3	-						
1	7	-	-					
2	29		-	23				
3	23			29				
4	12							
5	82							
6	1							
7	72							
8	8							
9	5							

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.4. 应用

P. 127-128 例5.3 冒泡法排序

```
int main()
{
    .....
    for(j=0; j<9; j++)
    {
        for(i=0; i<9-j; i++)
        {
            if (a[i] > a[i+1]) {
                t=a[i];
                a[i]=a[i+1];
                a[i+1]=t;
            }
        }
    }
    .....
}
```

两数  
交换  
方法

P. 128的程序定义了a[11]，但  
只用a[1]-a[10]，a[0]废弃  
本程序用a[0]-a[9]，有所不同  
for (j=1; j<=9; j++)  
for (i=1; i<=10-j; i++)

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次，外循环j=0

内循环 0 1 2 3 4 5 6 7 8

0	3	-						
1	7	-	-					
2	29		-	23				
3	23			29	12			
4	12				29			
5	82							
6	1							
7	72							
8	8							
9	5							



## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.4. 应用

P. 127-128 例5.3 冒泡法排序

```
int main()
{
    .....
    for(j=0; j<9; j++)
    {
        for(i=0; i<9-j; i++)
        {
            if (a[i] > a[i+1]) {
                t=a[i];
                a[i]=a[i+1];
                a[i+1]=t;
            }
        }
    }
    .....
}
```

两数  
交换  
方法

P. 128的程序定义了a[11]，但  
只用a[1]-a[10]，a[0]废弃  
本程序用a[0]-a[9]，有所不同  
for (j=1; j<=9; j++)  
for (i=1; i<=10-j; i++)

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次，外循环j=0

内循环 0 1 2 3 4 5 6 7 8

0	3	-						
1	7	-	-					
2	29		-	23				
3	23			29	12			
4	12				29	-		
5	82					-		
6	1							
7	72							
8	8							
9	5							

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.4. 应用

P. 127-128 例5.3 冒泡法排序

```
int main()
{
    .....
    for(j=0; j<9; j++)
    {
        for(i=0; i<9-j; i++)
        {
            if (a[i] > a[i+1]) {
                t=a[i];
                a[i]=a[i+1];
                a[i+1]=t;
            }
        }
    }
    .....
}
```

两数  
交换  
方法

P. 128的程序定义了a[11]，但  
只用a[1]-a[10]，a[0]废弃  
本程序用a[0]-a[9]，有所不同  
for (j=1; j<=9; j++)  
for (i=1; i<=10-j; i++)

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次，外循环j=0

内循环 0 1 2 3 4 5 6 7 8

0	3	-						
1	7	-	-					
2	29		-	23				
3	23			29	12			
4	12				29	-		
5	82					-	1	
6	1						82	
7	72							
8	8							
9	5							

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.4. 应用

P. 127-128 例5.3 冒泡法排序

```
int main()
{
    .....
    for(j=0; j<9; j++)
    {
        for(i=0; i<9-j; i++)
        {
            if (a[i] > a[i+1]) {
                t=a[i];
                a[i]=a[i+1];
                a[i+1]=t;
            }
        }
    }
    .....
}
```

两数  
交换  
方法

P. 128的程序定义了a[11]，但  
只用a[1]-a[10]，a[0]废弃  
本程序用a[0]-a[9]，有所不同  
for (j=1; j<=9; j++)  
for (i=1; i<=10-j; i++)

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次，外循环j=0

内循环 0 1 2 3 4 5 6 7 8

0	3	-						
1	7	-	-					
2	29		-	23				
3	23			29	12			
4	12				29	-		
5	82					-	1	
6	1						82	72
7	72							82
8	8							
9	5							

## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.4. 应用

P. 127-128 例5.3 冒泡法排序

```
int main()
{
    .....
    for(j=0; j<9; j++)
    {
        for(i=0; i<9-j; i++)
        {
            if (a[i] > a[i+1]) {
                t=a[i];
                a[i]=a[i+1];
                a[i+1]=t;
            }
        }
    }
    .....
}
```

两数  
交换  
方法

P. 128的程序定义了a[11]，但  
只用a[1]-a[10]，a[0]废弃  
本程序用a[0]-a[9]，有所不同  
for (j=1; j<=9; j++)  
for (i=1; i<=10-j; i++)

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次，外循环j=0

内循环 0 1 2 3 4 5 6 7 8

0	3	-						
1	7	-	-					
2	29		-	23				
3	23			29	12			
4	12				29	-		
5	82					-	1	
6	1						82	72
7	72							82 8
8	8							
9	5							



## § 5. 利用数组处理批量数据

### 5.2. 一维数组的定义和引用

#### 5.2.4. 应用

P. 127-128 例5.3 冒泡法排序

```
int main()
{
    .....
    for(j=0; j<9; j++)
    {
        for(i=0; i<9-j; i++)
        {
            if (a[i] > a[i+1]) {
                t=a[i];
                a[i]=a[i+1];
                a[i+1]=t;
            }
        }
    }
    .....
}
```

两数  
交换  
方法

P. 128的程序定义了a[11]，但  
只用a[1]-a[10]，a[0]废弃  
本程序用a[0]-a[9]，有所不同  
for (j=1; j<=9; j++)  
for (i=1; i<=10-j; i++)

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次，外循环j=0

内循环 0 1 2 3 4 5 6 7 8

0	3	-							3
1	7	-	-						7
2	29		-	23					23
3	23			29	12				12
4	12				29	-			29
5	82					-	1		1
6	1						82	72	72
7	72							82	8
8	8								82
9	5								5

第1次外循环后，最大数82在最后，第2次  
外循环就不必再比较该数了

## § 5. 利用数组处理批量数据

### 5.3. 二维数组的定义和引用

#### 5.3.1. 定义

数据类型 数组名[正整常量1][正整常量2]

行 列

```
int a[5][7];
```

```
long s[2*8][9];
```

★ 对应为一张二维表

★ 包括整型常量、整型符号常量和整型只读变量

```
#define M 5          const int m=5,n=10;
#define N 10         int a[m][n]; ✓
int a[M][N] ✓
```

```
int i=5, j=10;
int a[i][j]; ✗
```

- 早期C/C++标准中，数组大小必须是常量，新的标准已允许为变量
- 为统一，此处仍要求为常量

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int m, n;
    cin >> m >> n;
    int a[m][n];
}
```

VS2017编译 : ✗  
三编译器编译: ✓

## § 5. 利用数组处理批量数据

### 5.3. 二维数组的定义和引用

#### 5.3.1. 定义

#### 5.3.2. 使用

- ★ 必须先定义，后使用，数组名的命名规则同变量名
- ★ 数组的大小在声明时应确定，不能动态定义，在内存中存放时**先行后列**，占用一块连续的空间
- ★ 若数组定义中整型常量表达式为m, n，则表示有m\*n个元素，数组长度应为m\*n，也称数组有m行n列
- ★ 每个数组元素占用一个**数据类型**所占用的字节数，数组元素的表示方法为**数组名[行下标][列下标]**，行下标的范围从**[0..m-1]**，列下标的范围从**[0..n-1]**，表示为整型常量或整型表达式

例：有数组定义int a[3][4]，则数组有3行4列，元素排列为：

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

共有3\*4=12个数组元素，数组长度为12  
每个数组元素占4个字节  
整个数组占用3\*4\*sizeof(int)=48个字节  
行下标的范围：0-2  
列下标的范围：0-3  
内存存放为：

P. 129 表5.1 的解释错，  
应为：s[2][3]表示整数73

a[0][0]	2000 2003	
a[0][1]	2004 2007	
a[0][2]	2008 2011	
a[0][3]	2012 2015	
a[1][0]	2016 2019	
a[1][1]	2020 2023	
a[1][2]	2024 2027	
a[1][3]	2028 2031	
a[2][0]	2032 2035	
a[2][1]	2036 2039	
a[2][2]	2040 2043	
a[2][3]	2044 2047	

```
#include <iostream>
using namespace std;
int main()
{ int a[3][4];
  cout << sizeof(a) << endl;
  cout << sizeof(a[1][2]) << endl;
}
```

48  
4



## § 5. 利用数组处理批量数据

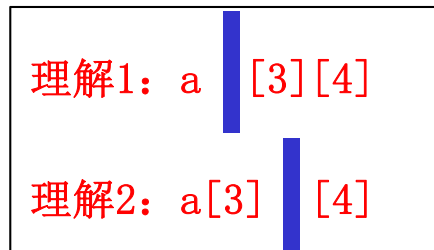
### 5.3. 二维数组的定义和引用

#### 5.3.1. 定义

#### 5.3.2. 使用

- ★ 二维数组可以看作是一个一维数组，  
它的每个数组元素都是一个一维数组

二维数组 `int a[3][4]`，  
理解为一维数组，有3(行)个元素  
每个元素又是一维数组，有4(列)个元素  
`a`是二维数组名  
`a[0]`, `a[1]`, `a[2]`是一维数组名  
若分别替换为*i, j, k*  
则：`int a[3][4]`;  
=> `int i[4], j[4], k[4]`;  
理解为三个包含四个元素的一维数组



a[0]	<code>a[0][0]</code>	2000 2003	
	<code>a[0][1]</code>	2004 2007	
	<code>a[0][2]</code>	2008 2011	
	<code>a[0][3]</code>	2012 2015	
a[1]	<code>a[1][0]</code>	2016 2019	
	<code>a[1][1]</code>	2020 2023	
	<code>a[1][2]</code>	2024 2027	
	<code>a[1][3]</code>	2028 2031	
a[2]	<code>a[2][0]</code>	2032 2035	
	<code>a[2][1]</code>	2036 2039	
	<code>a[2][2]</code>	2040 2043	
	<code>a[2][3]</code>	2044 2047	

```
#include <iostream>
using namespace std;
int main()
{ int a[3][4];
  cout << sizeof(a) << endl;
  cout << sizeof(a[0]) << endl;
  cout << sizeof(a[1][2]) << endl;
}
```

48  
16  
4

## § 5. 利用数组处理批量数据

### 5.3. 二维数组的定义和引用

#### 5.3.1. 定义

#### 5.3.2. 使用

- ★ 数组的使用只能逐个引用其中元素，不能整体使用，引用时数组下标的表示为**常量或带变量的表达式**

```
int a[10][20];
```

```
a[0][7]=15;
```

✓

```
float c; long d; c+d*a[5][3];
```

✓

```
int k=3; a[k*4-3][k/2+4]=18;
```

✓

```
10+a[0][9]+a[1][3]-a[2][2];
```

✓

```
int a[3][4], i, j;
```

```
cout << a;
```

✗

```
cout << a[0];
```

✗

```
cout << a[0][0] << a[1][2] << a[2][3];
```

✓

```
for(i=0;i<3;i++)
```

```
for(j=0;j<4;j++)
```

✓

```
cout << a[i][j];
```

错误是指无法得到预期结果，编译本身没错，得到的是另外值（地址）

- ★ 引用时，若数组下标超出范围，C/C++不会报错，因此编程者要控制下标在合理的范围内

```
int a[10][20];
```

```
3*5+a[10][5]; ✗ (引用操作)
```

```
3*5+a[5][20]; ✗ (引用操作)
```

```
a[5*3][21]=16; ✗ (修改操作)
```

注意：编译都不会报错，是执行时错误  
=> 数组下标越界即使执行时不错，也是严重错误

## § 5. 利用数组处理批量数据

### 5.3. 二维数组的定义和引用

#### 5.3.3. 数组在定义时初始化

##### 5.3.3.1. 全部初始化

A. `int a[3][4]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};`

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

`a[1][0]=5 a[1][1]=6 a[1][2]=7 a[1][3]=8`

`a[2][0]=9 a[2][1]=10 a[2][2]=11 a[2][3]=12`

`int a[3][4]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};`

错误，元素的个数超过了数组的大小

B. `int a[3][4]={ {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };`

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

`{5, 6, 7, 8},`

`{9, 10, 11, 12} };`

更直观

1	2	3	4
5	6	7	8
9	10	11	12

`int a[3][4]={ {1, 2, 3, 4, 5}, {5, 6, 7, 8}, {9, 10, 11, 12} };`

错误，元素的总个数超过了数组大小

`int a[3][4]={ {1, 2, 3, 4, 5}, {6, 7, 8}, {9, 10, 11, 12} };`

错误，元素的总个数虽然正确，但一行的个数超过了数组定义的行大小

<code>a[0][0]</code>	2000 2003	1
<code>a[0][1]</code>	2004 2007	2
<code>a[0][2]</code>	2009 2011	3
<code>a[0][3]</code>	2012 2015	4
<code>a[1][0]</code>	2016 2019	5
<code>a[1][1]</code>	2020 2023	6
<code>a[1][2]</code>	2024 2027	7
<code>a[1][3]</code>	2028 2031	8
<code>a[2][0]</code>	2032 2035	9
<code>a[2][1]</code>	2036 2039	10
<code>a[2][2]</code>	2040 2043	11
<code>a[2][3]</code>	2044 2047	12

## § 5. 利用数组处理批量数据

### 5.3. 二维数组的定义和引用

#### 5.3.3. 数组在定义时初始化

##### 5.3.3.1. 全部初始化

A. `int a[3][4]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};`

B. `int a[3][4]={ {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};`

C. `int a[][4]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};`

(行缺省=3) (列不可省略)

✗ `int a[3][]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};`

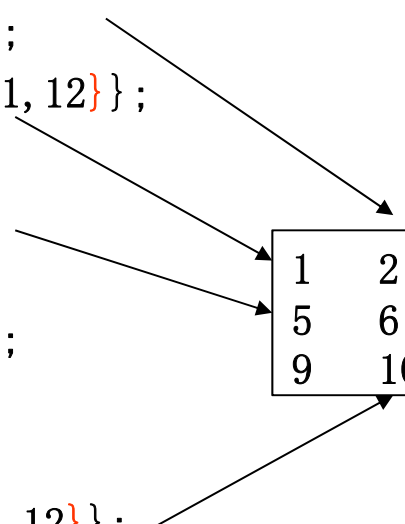
不允许缺省列

D. `int a[][4]={ {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};`

(行缺省=3) (列不可省略)

✗ `int a[3][]={ {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};`

不允许缺省列



1	2	3	4
5	6	7	8
9	10	11	12

问：为什么只能行缺省而不能列缺省？

答：二维表形式： $\left\{ \begin{array}{l} \text{行缺省, 行} = \text{总数/列} \\ \text{列缺省, 列} = \text{总数/行} \end{array} \right.$  无法理解

元素是一维数组的一维数组：

行缺省：元素大小(一维数组)已知，省略个数

列缺省：元素个数已知，省略大小 ✗

## § 5. 利用数组处理批量数据

### 5.3. 二维数组的定义和引用

#### 5.3.3. 数组在定义时初始化

##### 5.3.3.1. 全部初始化

##### 5.3.3.2. 部分初始化

A. `int a[3][4]={1, 5, 9};`

`a[0][0]=1 a[0][1]=5 a[0][2]=9`, 其余为0

1	5	9	0
0	0	0	0
0	0	0	0

B. `int a[3][4]={ {1}, {5}, {9} };`

`a[0][0]=1 a[1][0]=5 a[2][0]=9`, 其余为0

1	0	0	0
5	0	0	0
9	0	0	0

C. `int a[3][4]={ {1, 2}, {5, 6, 7}, {9, 10} };`

1	2	0	0
5	6	7	0
9	10	0	0

D. `int a[3][4]={ {1}, {5, 6} }`

1	0	0	0
5	6	0	0
0	0	0	0

E. `int a[3][4]={ {1}, {}, {5, 6} };`

1	0	0	0
0	0	0	0
5	6	0	0

## § 5. 利用数组处理批量数据

### 5.3. 二维数组的定义和引用

#### 5.3.3. 数组在定义时初始化

##### 5.3.3.2. 部分初始化

F. `int a[][4]={ {1}, {5}, {9} };`

3(省略)

G. `int a[][4]={ {1}, {5} };`

2(省略)

H. `int a[][4]={ {1}, {}, {5} };`

3(省略)

I. `int a[][4]={1, 2, 3, 4, 5, 6, 7, 8, 9};`

3(省略)

3 = 「初始化元素个数/列数」

J. `int a[][4]={1, 2, 3, 4, 5};`

2(省略) = 「初始化元素个数/列数」

如何验证?



```
#include <iostream>
using namespace std;
int main()
{
    int a1[][4]={ {1}, {5}, {9} };           //3
    int a2[][4]={ {1}, {5} };                 //2
    int a3[][4]={ {1}, {}, {5} };             //3
    int a4[][4]={1, 2, 3, 4, 5, 6, 7, 8, 9};   //3
    int a5[][4]={1, 2, 3, 4, 5};               //2

    cout << sizeof(a1) << endl;               //期望48
    cout << sizeof(a2) << endl;               //期望32
    cout << sizeof(a3) << endl;               //期望48
    cout << sizeof(a4) << endl;               //期望48
    cout << sizeof(a5) << endl;               //期望32
}
```

## § 5. 利用数组处理批量数据

### 5.3. 二维数组的定义和引用

#### 5.3.4. 应用

P. 131-133 例5.4 2x3 转置为 3x2

例5.5 求最大值及所在行列

```
int main()
{
    .....
    for(i=0; i<=1; i++) {
        for(j=0; j<=2; j++) {
            cout << a[i][j] << " ";
            b[j][i]=a[i][j]; //转置
        }
        cout << endl;
    }
    .....
}
```

```
int main()
{
    .....
    max=a[0][0]; row=0; col=0; //初始认为a[0][0]
    for(i=0; i<=2; i++)
        for(j=0; j<=3; j++)
            if (a[i][j] > max) {
                max=a[i][j]; //替换新的max
                row=i;       //同时记录下行、列值
                col=j;
            }
    .....
}
```

## § 5. 利用数组处理批量数据

### 5.3. 二维数组的定义和引用

补：多维数组的基本概念

★ 定义：数据类型 数组名[N1][..][..][..][..]

★ 内存中的排列：离数组名最远的维数变化最快，最近的维数变化最慢

★ 一般理解：三维/四维/.../ N维空间

★ 专业理解：N维数组是元素是N-1维数组的一维数组

=> N维数组是基本元素的类型为数据类型的一维数组的一维数组的...一维数组

★ 定义时初始化：允许N层括号嵌套，每层括号内的元素数量受各层对应维数的限制  
只能省略最靠近数组名的维数大小

```
int a[3][4][5][6][7];  
内存：a[0][0][0][0][0], ..., a1[0][0][0][0][6]  
      a[0][0][0][1][0], ..., a1[0][0][0][1][6]  
      ...  
      a[0][0][0][5][0], ..., a1[0][0][0][5][6]  
      ...  
      a[2][3][4][5][0], ..., a1[2][3][4][5][6]
```

定义时初始化：

```
int a[3][4][5][6][7] = { {}, {}, {} };  
=> { { {}, {}, {}, {} }, ... }  
=> { { { {}, {}, {}, {}, {} }, ... }, ... }  
=> { { { { {}, {}, {}, {}, {}, {} }, ... }, ... }, ... }  
=> { { { { { 2, 6, 1, 3, 9, 8, 12 }, ... }, ... }, ... }, ... }
```



## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

#### 5. 4. 1. 用数组元素做函数实参

#### ★ 形参为相应类型的简单变量

P. 133 例5.6 数组元素做参数求最大值及所在行列

```
int max_value(int x, int max)
{
    if (x > max)
        return x;
    else
        return max;
}

int main()
{
    ...
    max = a[0][0];
    for(i=0; i<=2; i++)
        for(j=0; j<=3; j++) {
            max=max_value(a[i][j], max);
            ...
        }
}
```

实参:二维数组元素(int型)  
形参:int型简单变量

## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

#### 5. 4. 1. 用数组元素做函数实参

#### 5. 4. 2. 用一维数组名做函数实参

### ★ 形参为相应类型的一维数组

P. 134 例5. 7 选择法排序

```
int main() { ... select_sort(a, 10); ...};
```

```
void select_sort(int array[], int n)
```

```
{  int i, j, k, t;
```

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

```
        k=i;
```

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

```
            if (array[j] < array[k])
```

```
                k=j;
```

```
        t=array[k];
```

```
        array[k]=array[i];
```

```
        array[i]=t;
```

```
    }
```

```
}
```

实参:一维数组名  
形参:一维数组

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次, 外循环i=0 k=0 内循环 1-9

0	1	2	3	4	5	6	7	8	9
3	7	29	23	12	82	1	72	8	5

## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

#### 5. 4. 1. 用数组元素做函数实参

#### 5. 4. 2. 用一维数组名做函数实参

### ★ 形参为相应类型的一维数组

P. 134 例5. 7 选择法排序

```
int main() { ... select_sort(a, 10); ...};
```

```
void select_sort(int array[], int n)
```

```
{  int i, j, k, t;
```

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

```
        k=i;
```

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

```
            if (array[j] < array[k])
```

```
                k=j;
```

```
        t=array[k];
```

```
        array[k]=array[i];
```

```
        array[i]=t;
```

```
    }
```

```
}
```

实参:一维数组名  
形参:一维数组

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次, 外循环i=0 k=0 内循环 1-9

	0	1	2	3	4	5	6	7	8	9
0	3	7	29	23	12	82	1	72	8	5
1	7	<	3	-						
2										
3										
4										
5										
6										
7										
8										
9										

## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

#### 5. 4. 1. 用数组元素做函数实参

#### 5. 4. 2. 用一维数组名做函数实参

### ★ 形参为相应类型的一维数组

P. 134 例5. 7 选择法排序

```
int main() { ... select_sort(a, 10); ...};
```

```
void select_sort(int array[], int n)
```

```
{  int i, j, k, t;
```

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

```
        k=i;
```

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

```
            if (array[j] < array[k])
```

```
                k=j;
```

```
        t=array[k];
```

```
        array[k]=array[i];
```

```
        array[i]=t;
```

```
    }
```

```
}
```

实参:一维数组名  
形参:一维数组

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次, 外循环i=0 k=0 内循环 1-9

	0	1	2	3	4	5	6	7	8	9
	3	7	29	23	12	82	1	72	8	5
1	7	< 3	-							
2	29	< 3	-							
3										
4										
5										
6										
7										
8										
9										

## § 5. 利用数组处理批量数据

## 5.4. 用数组名作函数参数

### 5.4.1. 用数组元素做函数实参

### 5.4.2. 用一维数组名做函数实参

## ★ 形参为相应类型的一维数组

### P. 134 例5.7 选择法排序

```
int main() { ... select_sort(a, 10); ...};
```

```
void select_sort(int array[], int n)
```

```
{   int i, j, k, t;
```

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

$$k=i;$$

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

```
if (array[j] < array[k])
```

$$k=j;$$

```
t=array[k];
```

```
array[k]=array[i];
```

```
array[i]=t;
```

}

}

实参:一维数组名  
形参:一维数组

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次, 外循环 $i=0$   $k=0$  内循环 1-9

0 1 2 3 4 5 6 7 8 9

3 7 29 23 12 82 1 72 8 5

1 7 < 3 -

2  $29 < 3$  –

3  $23 < 3$  -

4

5

6

7

8

9

## § 5. 利用数组处理批量数据

## 5.4. 用数组名作函数参数

### 5.4.1. 用数组元素做函数实参

### 5.4.2. 用一维数组名做函数实参

### ★ 形参为相应类型的一维数组

## P. 134 例5.7 选择法排序

```
int main() { ... select_sort(a, 10); ...};
```

```
void select_sort(int array[], int n)
```

```
{  int i, j, k, t;
```

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

$$k=i;$$

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

```
if (array[j] < array[k])
```

k=j:

```
t=array[k];
```

```
array[k]=array[i];
```

```
array[i]=t;
```

}

$$\}$$

实参:一维数组名  
形参:一维数组

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次, 外循环 $i=0$   $k=0$  内循环 1-9

0 1 2 3 4 5 6 7 8 9

3 7 29 23 12 82 1 72 8 5

 $17 < 3 -$ 

2  $29 < 3$  –

3  $23 < 3$  –

4  $12 < 3 -$

5

6

7

8

9

## § 5. 利用数组处理批量数据

## 5.4. 用数组名作函数参数

### 5.4.1. 用数组元素做函数实参

### 5.4.2. 用一维数组名做函数实参

### ★ 形参为相应类型的一维数组

## P. 134 例5.7 选择法排序

```
int main() { ... select_sort(a, 10); ...};
```

```
void select_sort(int array[], int n)
```

```
{  int i, j, k, t;
```

```
for (i=0; i<n-1; i++)\{
```

```
k=i;
```

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

```
if (array[j] < array[k])
```

k=j:

```
t=array[k];
```

```
array[k]=array[i];
```

```
array[i]=t;
```

$$\}$$
$$\}$$

实参:一维数组名  
形参:一维数组

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次, 外循环*i*=0    *k*=0    内循环 1-9

0 1 2 3 4 5 6 7 8 9

3 7 29 23 12 82 1 72 8 5

1 7 < 3 -

2  $29 < 3$  —

3 23 < 3 -

4  $12 < 3 -$

5  $82 < 3$  —

6

7

8

9

## § 5. 利用数组处理批量数据

## 5.4. 用数组名作函数参数

### 5.4.1. 用数组元素做函数实参

### 5.4.2. 用一维数组名做函数实参

### ★ 形参为相应类型的一维数组

### P. 134 例5.7 选择法排序

```
int main() { ... select sort(a, 10); ...};
```

```
void select_sort(int array[], int n)
```

```
{  int i, j, k, t;
```

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

$$k=i;$$

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

```
if (array[j] < array[k])
```

 $k=j;$ 

```
t=array[k];
```

```
array[k]=array[i];
```

```
array[i]=t;
```

}

}

实参:一维数组名  
形参:一维数组

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次, 外循环 $i=0$   $k=0$  内循环 1-9

0 1 2 3 4 5 6 7 8 9

3 7 29 23 12 82 1 72 8 5

$$17 < 3 -$$

2  $29 < 3$  –

3  $23 < 3 -$

4  $12 < 3 -$

5  $82 < 3 -$

6 1 < 3 k=6

7

8

9



## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

#### 5. 4. 1. 用数组元素做函数实参

#### 5. 4. 2. 用一维数组名做函数实参

### ★ 形参为相应类型的一维数组

P. 134 例5. 7 选择法排序

```
int main() { ... select_sort(a, 10); ...};
```

```
void select_sort(int array[], int n)
```

```
{  int i, j, k, t;
```

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

```
        k=i;
```

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

```
            if (array[j] < array[k])
```

```
                k=j;
```

```
        t=array[k];
```

```
        array[k]=array[i];
```

```
        array[i]=t;
```

```
    }
```

```
}
```

实参: 一维数组名  
形参: 一维数组

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次, 外循环i=0 k=0 内循环 1-9

	0	1	2	3	4	5	6	7	8	9
	3	7	29	23	12	82	1	72	8	5

1 7 < 3 -

2 29 < 3 -

3 23 < 3 -

4 12 < 3 -

5 82 < 3 -

6 1 < 3 k=6

7 72 < 1 -

8

9

## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

#### 5. 4. 1. 用数组元素做函数实参

#### 5. 4. 2. 用一维数组名做函数实参

### ★ 形参为相应类型的一维数组

P. 134 例5. 7 选择法排序

```
int main() { ... select_sort(a, 10); ...};
```

```
void select_sort(int array[], int n)
```

```
{ int i, j, k, t;
```

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

```
    k=i;
```

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

```
        if (array[j] < array[k])
```

```
            k=j;
```

```
    t=array[k];
```

```
    array[k]=array[i];
```

```
    array[i]=t;
```

```
}
```

```
}
```

实参: 一维数组名  
形参: 一维数组

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次, 外循环i=0 k=0 内循环 1-9

	0	1	2	3	4	5	6	7	8	9
	3	7	29	23	12	82	1	72	8	5

1 7 < 3 -

2 29 < 3 -

3 23 < 3 -

4 12 < 3 -

5 82 < 3 -

6 1 < 3 k=6

7 72 < 1 -

8 8 < 1 -

9

## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

#### 5. 4. 1. 用数组元素做函数实参

#### 5. 4. 2. 用一维数组名做函数实参

### ★ 形参为相应类型的一维数组

P. 134 例5.7 选择法排序

```
int main() { ... select_sort(a, 10); ...};
```

```
void select_sort(int array[], int n)
```

```
{ int i, j, k, t;
```

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

```
    k=i;
```

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

```
        if (array[j] < array[k])
```

```
            k=j;
```

```
    t=array[k];
```

```
    array[k]=array[i];
```

```
    array[i]=t;
```

```
}
```

```
}
```

实参：一维数组名  
形参：一维数组

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次，外循环i=0 k=0 内循环 1-9

	0	1	2	3	4	5	6	7	8	9
	3	7	29	23	12	82	1	72	8	5
1	7	<	3	-						
2	29	<	3	-						
3	23	<	3	-						
4	12	<	3	-						
5	82	<	3	-						
6	1	<	3	k=6						
7	72	<	1	-						
8	8	<	1	-						
9	5	<	1	-						

## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

#### 5. 4. 1. 用数组元素做函数实参

#### 5. 4. 2. 用一维数组名做函数实参

### ★ 形参为相应类型的一维数组

P. 134 例5.7 选择法排序

```
int main() { ... select_sort(a, 10); ...};
```

```
void select_sort(int array[], int n)
```

```
{ int i, j, k, t;
```

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

```
    k=i;
```

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

```
        if (array[j] < array[k])
```

```
            k=j;
```

```
    t=array[k];
```

```
    array[k]=array[i];
```

```
    array[i]=t;
```

```
}
```

```
}
```

实参：一维数组名  
形参：一维数组

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次，外循环i=0 k=0 内循环 1-9

	0	1	2	3	4	5	6	7	8	9
	3	7	29	23	12	82	1	72	8	5

1 7 < 3 -

2 29 < 3 -

3 23 < 3 -

4 12 < 3 -

5 82 < 3 -

6 1 < 3 k=6

7 72 < 1 -

8 8 < 1 -

9 5 < 1 -

循环结束 k=6 a[0] ⇔ a[6]

1	7	29	23	12	82	3	72	8	5
---	---	----	----	----	----	---	----	---	---

## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

#### 5. 4. 1. 用数组元素做函数实参

#### 5. 4. 2. 用一维数组名做函数实参

### ★ 形参为相应类型的一维数组

P. 134 例5. 7 选择法排序

```
int main() { ... select_sort(a, 10); ...};
```

```
void select_sort(int array[], int n)
```

```
{  int i, j, k, t;
```

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

```
        k=i;
```

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

```
            if (array[j] < array[k])
```

```
                k=j;
```

```
        t=array[k];
```

```
        array[k]=array[i];
```

```
        array[i]=t;
```

```
    }
```

```
}
```

实参:一维数组名  
形参:一维数组

设a[10]为{3, 7, 29, 23, 12, 82, 1, 72, 8, 5}

第1次, 外循环i=0 k=0 内循环 1-9

0	1	2	3	4	5	6	7	8	9
3	7	29	23	12	82	1	72	8	5

内循环结束 a[0] ⇔ a[6]

1	7	29	23	12	82	3	72	8	5
---	---	----	----	----	----	---	----	---	---

第1次外循环结束, a[0]中已经是最小数

第2次:k=1 内循环:2-9 结束后a[1]次小

N个数, 外循环N-1次即可

## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

#### 5. 4. 1. 用数组元素做函数实参

#### 5. 4. 2. 用一维数组名做函数实参

#### ★ 形参为相应类型的一维数组

这些内容与第4章中简单变量做实形参的概念不同，且不完全正确，真正的原因第6章指针中再详细解释

★ 实参传递时，将实参数组的首地址(数组名表示数组的首地址)传给形参，因此实、形参数组的内存地址重合(实参占用空间，形参不占用空间)

★ 形参数组值的改变会影响到实参(与简单参数不同)

★ 因为形参数组不分配空间，因此数组大小可不指定

★ 因为形参数组不分配空间，因此实形参的类型必须完全相同，否则编译错

## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

#### 5. 4. 1. 用数组元素做函数实参

#### 5. 4. 2. 用一维数组名做函数实参

#### ★ 形参为相应类型的一维数组

★ 实参传递时，将实参数组的首地址(数组名表示数组的首地址)传给形参，因此实、形参数组的内存地址重合(实参占用空间，形参不占用空间)

//验证数组名/普通变量做函数参数，实形参地址是否相同

```
#include <iostream>
using namespace std;
void fun(int x[], int y)
{
    cout << "x_size=" << sizeof(x) << endl;
    cout << "addr_x=" << x << endl; //数组名代表首地址
    cout << "addr_y=" << &y << endl; //普通变量加&
    cout << "x[2]=" << x[2] << endl;
}
int main()
{
    int a[10] = {7, -2, 108, 25}, w=19;
    cout << "a_size=" << sizeof(a) << endl;
    cout << "addr_a=" << a << endl; //数组名代表首地址
    cout << "addr_w=" << &w << endl; //普通变量加&
    cout << "a[2]=" << a[2] << endl;
    fun(a, w);
}
```

```
a_size=40
addr_a=16进制地址a
addr_w=16进制地址w
a[2]=108
x_size=4
addr_x=16进制地址(与a相同)
addr_y=16进制地址(与w不同)
x[2]=108
```

说明

1. 证明了形参未分配40字节的数组空间  
(为什么是4第6章会解释)
2. 证明了实/形参内存地址重合  
(地址相同/[2]的值相同)

## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

#### 5. 4. 1. 用数组元素做函数实参

#### 5. 4. 2. 用一维数组名做函数实参

#### ★ 形参为相应类型的一维数组

★ 实参传递时，将实参数组的首地址(数组名表示数组的首地址)传给形参，因此实、形参数组的内存地址重合(实参占用空间，形参不占用空间)

★ 形参数组值的改变会影响到实参(与简单参数不同)

//数组名及简单变量做函数参数，验证形参是否影响实参

```
#include <iostream>
using namespace std;
void fun(int x[], int y)
{
    x[2]=37; //修改形参数组某元素的值
    y=45;    //修改简单变量形参的值
}
int main()
{
    int a[10] = {7, -2, 18, 25}, w=19;
    cout << a[2] << ' ' << w << endl;
    fun(a, w);
    cout << a[2] << ' ' << w << endl;
}
```

18 19
37 19



## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

#### 5. 4. 1. 用数组元素做函数实参

#### 5. 4. 2. 用一维数组名做函数实参

#### ★ 形参为相应类型的一维数组

★ 实参传递时，将实参数组的首地址(数组名表示数组的首地址)传给形参，因此实、形参数组的内存地址重合(实参占用空间，形参不占用空间)

★ 形参数组值的改变会影响到实参(与简单参数不同)

★ 因为形参数组不分配空间，因此数组大小可不指定

```
#include <iostream>
using namespace std;
void f1(int x1[]) //形参数组不指定大小
{ cout << "x1_size=" << sizeof(x1) << endl;
}
void f2(int x2[10]) //形参数组大小与实参相同
{ cout << "x2_size=" << sizeof(x2) << endl;
}
void f3(int x3[1234]) //形参数组大小与实参不同
{ cout << "x3_size=" << sizeof(x3) << endl;
}
int main()
{ int a[10];
  cout << "a_size=" << sizeof(a) << endl;
  f1(a);    f2(a);    f3(a);
}
```

a\_size=40  
x1\_size=4  
x2\_size=4  
x3\_size=4  
(为什么是4  
第6章会解释)

```
void select_sort(int array[10], int n);
void select_sort(int array[], int n);
void select_sort(int array[5], int n);
```

形参数组 { 和实参数组大小相同  
不指定大小  
和实参数组大小不同 } 均认为是正确的

## § 5. 利用数组处理批量数据

### 5.4. 用数组名作函数参数

#### 5.4.1. 用数组元素做函数实参

#### 5.4.2. 用一维数组名做函数实参

#### ★ 形参为相应类型的一维数组

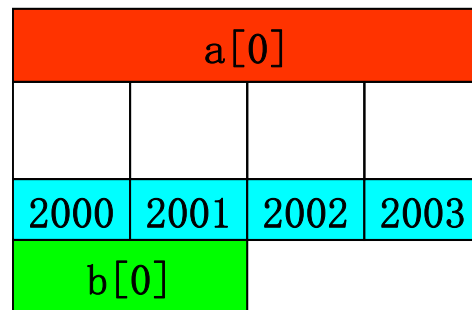
★ 实参传递时，将实参数组的首地址 (数组名表示数组的首地址) 传给形参，因此实、形参数组的内存地址重合 (实参占用空间，形参不占用空间)

★ 形参数组值的改变会影响到实参 (与简单参数不同)

★ 因为形参数组不分配空间，因此数组大小可不指定

★ 因为形参数组不分配空间，因此实形参的类型必须完全相同，否则编译错

实: int a[10]



形: short b[10]

```
#include <iostream>
using namespace std;
void f1(short b[])
{   return ;
}
void f2(long b[])
{   return ;
}
void f3(unsigned int b[])
{   return ;
}
int main()
{   int a[10];
    f1(a); //编译报错
    f2(a); //编译报错
    f3(a); //编译报错
}
```

## § 5. 利用数组处理批量数据

### 5.4. 用数组名作函数参数

#### 5.4.1. 用数组元素做函数实参

#### 5.4.2. 用一维数组名做函数实参

#### 5.4.3. 用多维数组名做函数实参

#### ★ 形参为相应类型的多维数组 P. 141 例5.8

```
int max_value(int array[][4])  
{  
}  
  
int main()  
{ int a[3][4];  
  max_value(a);  
}
```

实参: 二维数组名  
形参: 二维数组

```
#include <iostream>  
using namespace std;  
void f1(int x1[][4]) //形参数组不指定行大小  
{ cout << "x1_size=" << sizeof(x1) << endl;  
}  
void f2(int x2[3][4]) //形参数组行大小与实参相同  
{ cout << "x2_size=" << sizeof(x2) << endl;  
}  
void f3(int x3[123][4]) //形参数组行大小与实参不同  
{ cout << "x3_size=" << sizeof(x3) << endl;  
}  
  
int main()  
{ int a[3][4];  
  cout << "a_size=" << sizeof(a) << endl;  
  f1(a);  
  f2(a);  
  f3(a);  
}
```

a\_size=48  
x1\_size=4  
x2\_size=4  
x3\_size=4

#### ★ 实、形参数组的列必须相等，形参的行可以不指定，或为任意值

```
int main() { int a[3][4]; f(a); }  
  
int f(int x[3][4]);  
int f(int x[][4]);  
int f(int x[8][4]); //不推荐
```

三种方式  
都正确

问: 如何用一维数组的知识推导出本结论?

(如何做到知识的融会贯通?)

答: => 一维数组做参数要求实形参元素类型完全一致

=> 二维数组理解为元素是一维数组的一维数组

=> 元素类型完全一致

=> 做元素的一维数组完全一致

=> 实、形参数组的列必须相等

=> 形参是一维数组时，数组大小可以不指定

=> 形参的行可以不指定

## § 5. 利用数组处理批量数据

### 5. 4. 用数组名作函数参数

★ C/C++的函数返回类型不能是数组，但可以是指向数组的指针，具体见第6章

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

第2章中的概念:

- 1、char/unsigned char定义字符变量, 代表一个字符
- 2、一对单引号可表示字符常量, 可以字符/转义符形式('A' '\n' '\x41' '\101')
- 3、字符变量/常量在内存中占一个字节, 存储为该字符的ASCII码, 可当做1字节整数与整数通用
- 4、一对双引号可表示字符串常量, 字符串中字符的个数称为字符串的长度, 字符串的存储形式为每个字符的ASCII码+' \0' (尾零)
- 5、C++中无字符串变量, 可用一维字符数组来表示字符串变量
- 6、暂不讨论字符串中含' \0' 的情况("abc\0def")

#### 5.5.1. 含义

数据类型为字符型的数组

#### 5.5.2. 定义

char 数组名[正整型常量表达式]

一维数组

unsigned char 数组名[正整型常量表达式]

char 数组名[正整常量1][正整常量2]

二维数组

unsigned char 数组名[正整常量1][正整常量2]

★ 包括整型常量、整型符号常量和整型只读变量(部分编译器允许用变量定义数组, 不讨论)

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.3. 字符数组的初始化

##### 5.5.3.1. 全部初始化

`char a[5]={'c','h','i','n','a'};`      字符常量形式



`char a[5]={99, 104, 105, 110, 97};`      整数形式

`char a[5]='\143','\150','\151','\156','\141';`      8进制转义符形式

`char a[5]='\x63','\x68','\x69','\x6e','\x61';`      16进制转义符形式

四种表示方法等价，内存占5字节，表示如下：

0110 0011	0110 1000	0110 1001	0110 1110	0110 0001
-----------	-----------	-----------	-----------	-----------

`unsigned char a[4]={'c','h','i','n','a'};`

若个数多，则错误

`char a[]={'c','h','i','n','a'};`

缺省为5

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.3. 字符数组的初始化

##### 5.5.3.1. 全部初始化

```
char b[3][3]={'a','b','c','d','e','f','g','h','i'};
```

```
a  b  c
d  e  f
g  h  i
```

```
char b[][3]={'a','b','c','d','e','f','g','h','i'};
```

缺省为3

```
char b[3][3]={{'a','b','c'},{'d','e','f'},{'g','h','i'}};
```

```
char b[][3]= {{'a','b','c'},{'d','e','f'},{'g','h','i'}};
```

缺省为3

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.3. 字符数组的初始化

##### 5.5.3.1. 全部初始化

##### 5.5.3.2. 部分初始化

```
char a[10]= {'c','h','i','n','a'};
```



```
char a[10]={99, 104, 105, 110, 97};
```

对应a[0]-a[4]    a[5]-a[9]为'\0'

```
char a[3][3]= {'a','b','c'};
```

```
a   b   c
\0  \0  \0
\0  \0  \0
```

```
char a[3][3]= {'a'}, {'b'}, {'c'};
```

```
a   \0  \0
b   \0  \0
c   \0  \0
```

0	99
1	104
2	105
3	110
4	97
5	0
6	0
7	0
8	0
9	0

再次明确:		ASCII码
1字节整数	0	: 0000 0000
字符常量	'\0'	: 0000 0000
字符常量	'0'	: 0011 0000



## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.4. 字符串

##### 5.5.4.1. 含义

一串连续的字符

##### 5.5.4.2. 字符串的表示与存放

★ 一对双引号可表示字符串常量，字符串中字符的个数称为字符串的长度，字符串的存储形式为每个字符的ASCII码+' \0' (尾零)

问：常量"china"，想变为"China"，能否做到？

★ C++中无字符串变量，可用一维字符数组来表示字符串变量

=> 以一维字符数组方式表示，最后自动加一个' \0'

' \0' : 字符串结束标志(尾0)

##### 5.5.4.3. 字符数组与字符串的区别

字符串：最后一个字符必须为' \0'

字符数组：最后一个字符不必为' \0'

若无' \0'，不可与字符串相互替代

若有' \0'，可以与字符串相互表示

##### 5.5.4.4. 字符串的长度与字符数组的长度

字符串的长度：在' \0' 之前的实际长度

字符数组的长度：数组的大小

```
char a[10]={'c','h','i','n','a','\0','a','b','c','d'};
```

字符串长度：5

字符数组长度：10

字符数组a表示字符串"china"

令 a[0] = 'C';

则字符数组a所表示的字符串变为"China" => (变量)

```
char a[10]={'c','h','i','n','a','a','b','c','d','\0'};
```

字符串长度：9

字符数组长度：10

```
char a[10]={'c','h','i','n','a','a','b','c','d','e'};
```

字符串长度：不是字符串

字符数组长度：10

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.4. 字符串

##### 5.5.4.5. 用字符串常量的方式初始化字符数组

###### A. 全部初始化

`char a[6]="china";` 双引号

数组长度为字符串长度+1

`char a[6]="china";`

`char a[]="china";` 缺省为6

`char a[]={'c','h','i','n','a'};` 缺省为5

```
#include <iostream>
using namespace std;
int main()
{
    char a[]="china";
    char b[]={'c','h','i','n','a'};
    char c[5]="china"; //编译错
    cout << sizeof(a) << endl;
    cout << sizeof(b) << endl;
    return 0;
}
```

6

5

###### B. 部分初始化

`char a[10]="china";`

`a[5]-a[9]`为'\0'

###### C. 多个字符串初始化二维字符数组

`char a[3][6] = {"hello", "Hi", "Hello"};`

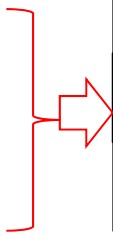
`char a[3][6] = {"hello", "Hi", "Hello"};`

`char a[][6] = {"hello", "Hi", "Hello"};`

`char a[][6] = {"hello", "Hi", "Hello"};`

`char a[3][5] = {"hello", "Hi", "Hello"};`

有错，为什么？



h	e	l	l	o	\0
H	i	\0	\0	\0	\0
H	e	l	l	o	\0

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.4. 字符串

##### 5.5.4.5. 用字符串常量的方式初始化字符数组

★ 字符数组定义后，无论是单字符方式，还是字符串方式，均不允许整体进行赋值操作，只能单个元素依次赋值

```
int a[10];  
a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; ×  
a[3] = 12; ✓  
char ch[10];  
ch = {'c', 'h', 'i', 'n', 'a'}; ×  
ch = "china"; ×  
ch[3] = 'A'; ✓
```

★ 字符串可用专用函数进行整体操作，具体见5.6

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.1. 输入

逐个输入: scanf("%c",&数组元素)      C方式

cin >> 数组元素      C++方式

//C方式输入单个字符

#define \_CRT\_SECURE\_NO\_WARNINGS //VS2017需要

#include <iostream>

using namespace std;

int main()

{ char a[10];

int i;

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

scanf("%c%c", &a[3], &a[7]); //假设输入AB

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

return 0;

}

数组下标表示前有  
取地址符号&  
因为scanf规定后面  
必须是变量的地址

先输出10行, 值随机

\*\*

..

\*\*

等待键盘输入, 输入AB后  
再输出10行, 其中2行  
有确定值

\*\*

\*\*

\*\*

65 //a[3]为 'A'

\*\*

\*\*

\*\*

66 //a[7]为 'B'

\*\*

\*\*

//C++方式输入单个字符

#include <iostream>

using namespace std;

int main()

{ char a[10];

int i;

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

cin >> a[3] >> a[7]; //假设输入AB

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

return 0;

}

数组下标表示前无  
取地址符号&

先输出10行, 值随机

\*\*

..

\*\*

等待键盘输入, 输入AB后  
再输出10行, 其中2行  
有确定值

\*\*

\*\*

\*\*

65 //a[3]为 'A'

\*\*

\*\*

\*\*

66 //a[7]为 'B'

\*\*

\*\*

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.1. 输入

逐个输入: scanf("%c",&数组元素)

C方式

cin >> 数组元素

C++方式

★ 多次逐个输入时，C方式会将回车当做下一个合法字符读取，而C++方式会忽略回车

//C方式多次逐个输入时回车的处理

#define \_CRT\_SECURE\_NO\_WARNINGS //VS2017需要

#include <iostream>

using namespace std;

int main()

{ char a[10];

int i;

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

scanf("%c%c", &a[3], &a[7]); //假设输入AB

scanf("%c", &a[0]);

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

return 0;

}

先输出10行，值随机

\*\*

..

\*\*

等待键盘输入，输入AB后再输出10行，其中3行有确定值

10 //a[0]为 '\n'

\*\*

\*\*

65 //a[3]为 'A'

\*\*

\*\*

\*\*

66 //a[7]为 'B'

\*\*

\*\*

//C++方式多次逐个输入时回车的处理

#include <iostream>

using namespace std;

int main()

{ char a[10];

int i;

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

cin >> a[3] >> a[7]; //假设输入AB

cin >> a[0];

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

return 0;

}

先输出10行，值随机

\*\*

..

\*\*

等待键盘输入，输入AB后会再次等待输入，假设输入C后再输出10行，其中3行有确定值

67 //a[0]为 'C'

\*\*

\*\*

65 //a[3]为 'A'

\*\*

\*\*

\*\*

66 //a[7]为 'B'

\*\*

\*\*

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.1. 输入

字符串形式: `scanf("%s", 数组名)`

`cin >> 数组名`

C方式

C++方式

- ★ 字符串形式跟数组名, 而不是数组元素
- ★ 输入的字符个数要小于定义的字符数组的长度
- ★ 输入时忽略掉最后的回车, 再自动加 '\0'

//C方式输入字符串

#define \_CRT\_SECURE\_NO\_WARNINGS //VS2017需要

#include <iostream>

using namespace std;

int main()

{ char a[10];

int i;

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

scanf("%s", a); //假设输入为Hello

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

return 0;

}

直接数组名, 无下标, 也不加&

因为C/C++规定, 数组名代表数组的起始地址

先输出10行, 值随机

\*\*

..

\*\*

等待键盘输入, 输入Hello  
后再输出10行, 其中6行有  
确定值

72 //a[0]为 'H'

101 //a[1]为 'e'

108 //a[2]为 'l'

108 //a[3]为 'l'

111 //a[4]为 'o'

0 //a[5]为 '\0'

\*\*

\*\*

\*\*

\*\*

回车未进入  
被\0替代

//C++方式输入字符串

#include <iostream>

using namespace std;

int main()

{ char a[10];

int i;

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

cin >> a; //假设输入为Hello

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

return 0;

}

直接数组名, 无下标, 也不加&

先输出10行, 值随机

\*\*

..

\*\*

等待键盘输入, 输入Hello  
后再输出10行, 其中6行有  
确定值

72 //a[0]为 'H'

101 //a[1]为 'e'

108 //a[2]为 'l'

108 //a[3]为 'l'

111 //a[4]为 'o'

0 //a[5]为 '\0'

\*\*

\*\*

\*\*

\*\*

回车未进入  
被\0替代

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.2. 输出

逐个: printf("%c", 数组元素)

cout << 数组元素

C方式

C++方式

//C/C++方式输出单个字符

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    char a[]="Student";    //长度缺省为8

    cout << sizeof(a) << endl;
    printf("%c*\n", a[5]);
    cout << a[3] << '*' << endl;

    return 0;
} //输出加*是为了确认只输出了一个字符
```

输出为:  
8  
n\*  
d\*

//C/C++方式以单个字符+循环形式输出整个数组

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    int i;
    char a[]="Student";
    for(i=0; i<7; i++)
        printf("%c", a[i]);
    cout << endl; //换行
    for(i=0; i<7; i++)
        cout << a[i];
    cout << endl; //换行
    return 0;
}
```

数组 a 缺省长度为8  
输出[0]-[6], 尾零不  
输出

输出为:  
Student  
Student

//C/C++方式以单个字符+循环形式输出整个数组

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    int i;
    char a[]="Student";
    for(i=0; i<7; i++)
        printf("%c, ", a[i]);
    cout << endl; //换行
    for(i=0; i<7; i++)
        cout << a[i] << '*';
    cout << endl; //换行
    return 0;
}
```

%c后面多一个,  
cout方式每个字符  
后面多一个\*

输出为:  
S, t, u, d, e, n, t,  
S\*t\*u\*d\*e\*n\*t\*

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.2. 输出

字符串形式: `printf("%s", 数组名)` C方式

`cout << 数组名` C++方式

//以字符串方式输出字符数组

```
#include <iostream>
using namespace std;
```

```
int main()
{
    char a[]="Student";
    printf("%s\n", a);
    cout << a << endl;

    return 0;
}
```

跟数组名  
不是数组元素名

输出为:  
Student  
Student

注: 尾零不会输出

//C/C++以字符串方式输出字符数组

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
```

```
    char a[]="Student\0china";
    cout << sizeof(a) << endl;
    printf("%s\n", a);
    cout << a << '*' << endl;
    cout << a[12] << endl;
    return 0;
}
```

1. 数组长度 14  
2. 字符串长度 7

输出为:

14  
Student\*  
Student\*  
a

注: 字符串形式输出字符  
数组仅到第一个'\0'为止



## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.2. 输出

字符串形式: `printf("%s", 数组名)` C方式

`cout << 数组名` C++方式

★ 字符串形式输出时跟数组名, 不是数组元素名

★ 从字符串的首字符开始输出, 到第1个' \0' 为止 (不含' \0' ), 字符数组的后续部分不再输出

推论: 若以字符串形式输出不含' \0' 的字符数组, 可能会得到不正确的结果

//以字符串方式输出不含' \0' 的字符数组

```
#include <iostream>
using namespace std;
```

```
int main()
{
```

```
    char a[5]={'C','h','i','n','a'};
    printf("%s\n", a);
    cout << a << endl;
    return 0;
}
```

```
int i;
for (i=0; i<5; i++)
    printf("%c", a[i]);
则不会看到后面的乱字符
```

//不能以字符串  
方式初始化

输出为:  
China加若干不确定字符  
甚至可能多行

//以字符串方式输出不含' \0' 的字符数组

```
#include <iostream>
using namespace std;
```

```
int main()
{
```

```
    char a[5]; //不初始化
    printf("%s\n", a);
    cout << a << endl;
    return 0;
}
```

输出为:  
若干不确定字符, 甚至  
可能多行

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.1. 输入

##### 5.5.5.2. 输出

##### 5.5.5.3. 从任一元素开始以字符串形式输入/输出

//从任一元素开始以字符串形式输出

```
#include <iostream>
using namespace std;
```

```
int main()
```

```
{
```

```
    char a[]="Student";
```

```
    printf("%s\n", &a[3]);
```

```
    cout << &a[3] << endl;
```

```
    return 0;
```

```
}
```

%s形式

&数组元素名形式

输出为:

dent

dent

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.1. 输入

##### 5.5.5.2. 输出

##### 5.5.5.3. 从任一元素开始以字符串形式输入/输出

//C方式从任一元素开始以字符串形式输入

#define \_CRT\_SECURE\_NO\_WARNINGS //VS2017需要

#include <iostream>

using namespace std;

int main()

{ int i;

char a[10];

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

scanf("%s", &a[3]); //假设输入Hello

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

return 0;

}

&数组元素名形式

先输出10行，值随机

\*\*

..

\*\*

等待键盘输入，输入Hello  
后再输出10行，其中6行有  
确定值

\*\*

\*\*

\*\*

72 //a[3]为 'H'  
101 //a[4]为 'e'  
108 //a[5]为 'l'  
108 //a[6]为 'l'  
111 //a[7]为 'o'  
0 //a[8]为 '\0'

\*\*

//C++方式从任一元素开始以字符串形式输入

#include <iostream>

using namespace std;

int main()

{ int i;

char a[10];

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

cin >> &a[3]; //假设输入Hello

for(i=0; i<10; i++)

cout << int(a[i]) << endl;

return 0;

}

&数组元素名形式

先输出10行，值随机

\*\*

..

\*\*

等待键盘输入，输入Hello  
后再输出10行，其中6行有  
确定值

\*\*

\*\*

\*\*

72 //a[3]为 'H'  
101 //a[4]为 'e'  
108 //a[5]为 'l'  
108 //a[6]为 'l'  
111 //a[7]为 'o'  
0 //a[8]为 '\0'

\*\*

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.1. 输入

##### 5.5.5.2. 输出

##### 5.5.5.3. 从任一元素开始以字符串形式输入/输出

★ C/C++方式输入输出都是 **&数组元素** 的形式

	C方式	C++方式
输入字符	scanf("%c", &元素名)	cin >> 元素名
输入字符串	scanf("%s", 数组名)	cin >> 数组名
输出字符	printf("%c", 元素名)	cout << 元素名
输出字符串	printf("%s", 数组名)	cout << 数组名
任一元素开始输入串	scanf("%s", &元素名)	cin >> &元素名
任一元素开始输出串	printf("%s", &元素名)	cout << &元素名

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.4. 多个字符串的输入

★ scanf/cin从键盘上输入的字符串不能含空格

//C方式多个字符串的输入

```
#define _CRT_SECURE_NO_WARNINGS //VS2017需要
#include <iostream>
#include <stdio>
using namespace std;

int main()
{
    char a[10], b[20];
    scanf("%s%s", a, b); //假设输入为abc空格def
    printf("%s-%s\n", a, b);
    return 0;
}
```

输出为:  
abc-def

//C++方式多个字符串的输入

```
#include <iostream>
using namespace std;

int main()
{
    char a[10], b[20];
    cin >> a >> b; //假设输入为abc空格def
    cout << a << '-' << b << endl;
    return 0;
}
```

输出为:  
abc-def

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.4. 多个字符串的输入

★ scanf/cin从键盘上输入的字符串不能含空格

★ 不同编译器从键盘输入含空格字符串的方法不同

- VS2017 : gets\_s, 无gets
- CodeBlocks: gets, 无gets\_s
- Dev C++ : gets, 无gets\_s
- Linux C++ : gets有warning, 无gets\_s
- 四个编译器均能使用fgets函数

fgets(字符数组名, 最大长度, stdin);  
但与gets/gets\_s的表现有不同, 请自行观察

★ scanf/cin通过某些高级设置方式还是可以输入含空格的字符串的, 本课程不再讨论

//VS2017下用gets\_s输入含空格的字符串

```
#include <iostream>
using namespace std;
int main()
{
    char a[10], b[20];
    gets_s(a); //最长输入9
    gets_s(b); //最长输入19
    cout << a << endl;
    cout << b << endl;
    return 0;
}
```

从键盘输入abc空格def  
会继续等待输入  
再输入xyz  
  
则输出为:  
abc def  
xyz

//CodeBlocks/Dev C++下用gets输入含空格的字符串

```
#include <iostream>
using namespace std;
int main()
{
    char a[10], b[20];
    gets(a); //最长输入9
    gets(b); //最长输入19
    cout << a << endl;
    cout << b << endl;
    return 0;
}
```

从键盘输入abc空格def  
会继续等待输入  
再输入xyz  
  
则输出为:  
abc def  
xyz

//四个编译器下均可用fgets输入含空格的字符串

```
#include <iostream>
using namespace std;
int main()
{
    char a[10], b[20];
    fgets(a, 10, stdin);
    fgets(b, 20, stdin);
    cout << a << endl;
    cout << b << endl;
    return 0;
}
```

从键盘输入abc空格def  
会继续等待输入  
再输入xyz  
  
则输出为:  
abc def  
xyz

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.5. 二维字符数组的输入/输出

★ 数组名加双下标表示元素，单下标表示一维数组

```
char a[3][30]={"***", "***", "***"};
```

// 单字符输出 (数组名+双下标)

```
printf("%c", a[1][15]);
```

```
cout << a[1][15];
```

// 字符串输出 (数组名+单下标)

```
printf("%s", a[1]);
```

```
cout << a[1];
```

//二维字符数组以单下标形式输出字符串

```
#include <iostream>
```

```
#include <cstdio>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    char a[3][30]={"ABCDEFGHJKLMNOPQRSTUVWXYZ",  
                  "abcdefghijklmnopqrstuvwxyz",  
                  "0123456789"};
```

```
    printf("a[0][2]=%c\n", a[0][2]);
```

```
    cout << "a[1][20]=" << a[1][20] << endl;
```

```
    printf("a[0]=%s\n", a[0]);
```

```
    cout << "a[2]=" << a[2] << endl;
```

```
}
```

```
a[0][2]=C
```

```
a[1][20]=u
```

```
a[0]=ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
a[2]=0123456789
```

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.5. 二维字符数组的输入/输出

★ 数组名加双下标表示元素，单下标表示一维数组

```
char a[3][30];  
// 单字符输入 (数组名+双下标)  
scanf("%c%c", &a[1][15], &a[2][29]);  
cin >> a[1][15] >> a[2][29];
```

```
char a[3][30];  
// 字符串输入 (数组名+单下标)  
scanf("%s", a[1]); // 不要超过29个字符  
cin >> a[1]; // 不要超过29个字符  
问：超出会怎样？为什么可以是59？
```

```
// 二维字符数组以单下标形式输入单个字符  
#define _CRT_SECURE_NO_WARNINGS //VS2017需要  
#include <iostream>  
#include <cstdio>  
using namespace std;  
int main()  
{  
    char a[3][30]={"ABCDEFGHIJKLMNPOQRSTUVWXYZ",  
                  "abcdefghijklmnopqrstuvwxyz",  
                  "0123456789"};  
    scanf("%c\n", &a[0][2]); // 输入一个符号#  
    cin >> a[1][20]; // 输入一个符号@  
    printf("a[0]=%s\n", a[0]);  
    cout << "a[1]=" << a[1] << endl;  
}
```

假设键盘输入#@，则输出为：  
a[0]=AB#DEFGHIJKLMNPOQRSTUVWXYZ  
a[1]=abcdefghijklmnopqrstuvwxyz@vwxyz

```
// 二维字符数组以单下标形式输入字符串  
#define _CRT_SECURE_NO_WARNINGS //VS2017需要  
#include <iostream>  
#include <cstdio>  
using namespace std;  
int main()  
{  
    char a[3][30];  
    printf("a[0]=%s\n", a[0]);  
    printf("a[1]=%s\n", a[1]);  
    printf("a[2]=%s\n", a[2]);  
    scanf("%s", a[1]); // a[1]是数组名, 无&, 或cin>>a[1];  
    cout << "a[0]=" << a[0] << endl;  
    cout << "a[1]=" << a[1] << endl;  
    cout << "a[2]=" << a[2] << endl;  
}
```

运行三次并观察：

- 1、输入≤29个字符
- 2、输入30-59个字符
- 3、输入60个字符

说明了什么？

二维数组越界理解，重要!!!!



## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.5. 字符数组的输入与输出

##### 5.5.5.5. 二维字符数组的输入/输出

★ 数组名加双下标表示元素，单下标表示一维数组

```
char a[3][30];  
//从任一位置开始输出字符串 (&+数组名+双下标)  
printf("%s", &a[1][3]);  
cout << &a[1][3];  
  
//从任一位置开始输入字符串 (&+数组名+双下标)  
scanf("%s", &a[1][3]); //不超过26/56个字符  
cin >> &a[1][3];      //不超过26/56个字符
```

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.6. 字符串处理函数

P. 146-148

- ① `strcat(char s[], const char t[])`
- ② `strcpy(char s[], const char t[])`
- ③ `strcmp(const char s[], const char t[])`
- ④ `strlen(const char s[])`

★ 使用前加 `#include <string.h> //C方式`  
`#include <cstring> //C++方式`

● VS2017下可以不加这个头文件

★ VS2017认为部分函数不够安全，使用前需要加 `#define _CRT_SECURE_NO_WARNINGS`

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.6. 字符串处理函数

① `strcat(char s[], const char t[])`

功 能：将字符串t连接到字符串s的尾部

输入参数：存放字符串s的字符数组s

存放字符串t的字符数组t (只读)

返 回 值：改变后的字符数组s

注意事项：字符数组s要有足够的空间 (两串总长+1)

//从P. 146例子而来的完整程序

`#define _CRT_SECURE_NO_WARNINGS`

`#include <iostream>`

`#include <cstring>`

`using namespace std;`

`int main()`

`{`

`char str1[30]="People's Republic of ";`

`char str2[]="China"; //缺省长度为6`

`cout << strcat(str1, str2) << endl;`

`return 0;`

`}`

图5.10 有错  
最后一个str1

cout输出为strcat函数的返回值，  
也证明了返回值是第1个字符数组

有空格

书上多), 错

输出为:

People's Republic of China

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.6. 字符串处理函数

##### ② strcpy(char s[], const char t[])

功 能：将字符串t复制到字符串s中, 覆盖原串s

输入参数：存放字符串s的字符数组s

存放字符串t的字符数组t (只读)

返 回 值：改变后的字符数组s

注意事项：字符数组s要有足够的空间 (串t长+1)

假设2:

char a[]="student", b[]="hellochina";

1、为什么错?

2、仅修改a的定义使正确, 如何做?

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    int i;
    char a[]="student", b[]="hello";
    strcpy(a, b);
    cout << a << endl;
    for(i=0; i<8; i++)
        cout << (int)a[i] << ' ';
    cout << endl;
    return 0;
}
```

a/b数组的缺省长度8/6

复制前a  
复制后a

s	t	u	d	e	n	t	\0
h	e	l	l	o	\0	t	\0

↑  
复制到\0为止  
a[6]以后保持原值

假设1:

char a[]="student", b[]="hello\0china";

则: a/b数组的缺省长度8/12

但结果与本例完全相同,  
复制的字符个数也相同

输出为:

hello

104 101 108 108 111 0 116 0

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.6. 字符串处理函数

② strcpy(char s[], const char t[])

★ 字符串复制时包含' \0'，到' \0' 为止

★ P.147错，应为strncpy(str1, str2, 2), 表示仅复制str2的前2个字符到str1中，并不再加\0

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    int i;
    char a[]="student", b[]="hello";
    strncpy(a, b, 2);
    cout << a << endl;
    for(i=0;i<8;i++)
        cout << (int)a[i] << ' ';
    cout << endl;
    return 0;
}
```

输出为:

heudent

104 101 117 100 101 110 116 0

证明了未加\0

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    int i;
    char a[]="student", b[]="hello";
    strncpy(a, _____, 2);
    cout << a << endl;
    for(i=0;i<8;i++)
        cout << (int)a[i] << ' ';
    cout << endl;
    return 0;
}
```

如果想从b[2]开始复制  
2个字符到a中，如何做？  
即期望输出: lludent

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.6. 字符串处理函数

② strcpy(char s[], const char t[])

★ 字符串复制时包含' \0'，到' \0' 为止

★ P.147错，应为strncpy(str1, str2, 2), 表示仅复制str2的前2个字符到str1中，并不再加\0

★ 不能用直接赋值的方法进行字符串的复制

char str1[10]="china"; 正确，定义时赋初值

char str2[10];

str2="hello"; 编译报错

str2=str1; 编译报错

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.6. 字符串处理函数

③ strcmp(const char s[], const char t[])

功 能：比较字符串s和字符串t的大小

输入参数：存放字符串s的字符数组s (只读)

存放字符串t的字符数组t (只读)

返 回 值：整型值(0:相等 >0:串1大 <0:串1小)

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char str1[] = "house", str2[] = "horse";
    char str3[] = "abcd", str4[] = "abcde";
    char str5[] = "abcd", str6[] = "abc";
    char str7[] = "abcd", str8[] = "abcd";
    char str9[] = "abcd", str10[] = "abcd\0efgh";
    cout << strcmp(str1, str2) << endl;
    cout << strcmp(str3, str4) << endl;
    cout << strcmp(str5, str6) << endl;
    cout << strcmp(str7, str8) << endl;
    cout << strcmp(str9, str10) << endl;
} //仅比较无赋值，因此不需加_CRT_SECURE_NO_WARNINGS
```

1	串1>串2
-1	<
1	>
0	==
0	==

```
#include <iostream>
#include <cstring>
using namespace std;
```

另一种输出形式：  
串1 < 串2

```
int main()
{
    char str1[]="abcd", str2[]="abcde";
    int k = strcmp(str1, str2);
    if (k==0)
        cout << "串1 = 串2" << endl;
    else if (k<0)
        cout << "串1 < 串2" << endl;
    else
        cout << "串1 > 串2" << endl;

    return 0;
}
```

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.6. 字符串处理函数

③ `strcmp(const char s[], const char t[])`

★ 两串相等的条件是长度相等且对应位置字符的ASCII码值相等

★ 字符串的比较流程如下：

两串首字符对应比较，若**不等则返回非0**，若相等则继续比较下一个字符，重复到两串对应字符均为'\0'则结束比较，**返回0(相等)**

`strcmp("house", "horse");` 到第3个字符结束

`strcmp("abcd", "abcde");` 到第5个字符结束

`strcmp("abcd", "abc");` 到第4个字符结束

`strcmp("abcd", "abcd");` 到第5个字符结束

★ 不相等返回时，有些系统返回-1/1，有些系统返回第一个不相等字符的ASCII差值，因此一般不比较具体值，而只是判断 `>0` / `<0` / `==0`

`strcmp("house", "horse");` VS2015不同编译器返回1  
某些编译器可能返回3



## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.6. 字符串处理函数

③ strcmp(const char s[], const char t[])

★ 不能直接用比较运算符比较字符串的大小

(但直接用比较运算符语法不错，只是含义不同)

```
#include <iostream>
using namespace std;

int main()
{   char str1[]="house", str2[]="horse";
    int k;
    k = str1 < str2;
    cout << k << endl;
    return 0;
}
```

输出为: 0

```
#include <iostream>
using namespace std;

int main()
{   char str1[]="horse", str2[]="house";
    int k;
    k = str1 < str2;
    cout << k << endl;
    return 0;
}
```

和左边比，已互换

输出为: 0

数组名代表数组的首地址，因此两串  
直接用比较运算符表示比较存放两串的  
内存地址的大小，而不是比较串的大小

★ 用strncmp(s, t, n)可比较两个字符串的前n个字符的大小

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.6. 字符串处理函数

##### ④ strlen(const char s[])

功 能：求字符串的长度

输入参数：存放字符串的字符数组

返 回 值：整型值表示的长度

注意事项：返回第一个'\0'前的字符数量, 不含'\0'

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{   char str[]="Hello";
    cout << sizeof(str) << endl;
    cout << strlen(str) << endl;
    return 0;
} //同strcmp, 不需要加
//_CRT_SECURE_NO_WARNINGS
```

输出为:

6  
5

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{   char str[]="china\0Hello\0\0";
    cout << sizeof(str) << endl;
    cout << strlen(str) << endl;
    return 0;
}
```

输出为:

14  
5

## § 5. 利用数组处理批量数据

### 5.5. 字符数组

#### 5.5.7. 字符数组应用

P.148 例5.10 输出字母排在最前的国家名

```
int main()
{
    void smallest_string(char str[][30], int i);
    int i;
    char country_name[3][30];
    for(i=0; i<3; i++)
        cin >> country_name[i];
    smallest_string(country_name, 3);
    return 0;
}

void smallest_string(char str[][30], int n)
{
    int i;
    char string[30];
    strcpy(string, str[0]); //先认为[0]是最小的
    for(i=0; i<n; i++)
        if (strcmp(str[i], string) < 0)
            strcpy(string, str[i]);
    cout << endl << "The smallest string is:" << string << endl;
    //输出先空一行
}
```

二维数组可理解为元素是一维数组的一维数组，因此可将country\_name理解为3个一维数组，每个一维数组最多存放30个字符(含'\0')

函数说明与函数实现中  
形参变量名不同，没关系

实参/形参

二维数组只写一个下标，  
可理解为一个一维数组的数组名，如前所述  
cin >> 一维数组名  
表示以字符串方式输入到一维字符数组中  
本例中表示循环读取3个串

初始已认为str[0]最小，  
因此比较从i=1开始即可

再依次比较，  
找到最小则赋值

## § 5. 利用数组处理批量数据

### 5.6. C++处理字符串的方法-字符串类与字符串变量

#### ★ 用一维字符数组来表示字符串变量的不足

- 字符串的长度受限于数组定义时的大小

=> 数组定义过大导致浪费

=> 数组定义过小导致不够

=> 数组定义的大小不能动态变化

- 赋值、复制、连接等必须用专用函数

#### ★ 字符串类(string类)的引入

string类是C++引入的字符串处理的新方法，通过定义类及运算符重载的方式实现

- 具体的实现原理及实现方法待第10章学习完成后再进一步了解
- 在本节中仅需要了解与一维字符数组表示字符串在表示及使用方法的差异即可

#### 5.6.1. 字符串变量的定义和引用

#### ★ 字符串变量的长度会自动调整，不必指定最大值

#### ★ 字符串变量内部存储时最后是否有'\0' 不能确定，因为长度可动态变化，故不关心其内部存储形式(通过第7章动态内存申请与释放的学习，以及第10章的作业可了解如何做到)

## § 5. 利用数组处理批量数据

### 5.6. C++处理字符串的方法-字符串类与字符串变量

#### 5.6.1. 字符串变量的定义和引用

项目	字符串变量	字符数组
适用	C++	C、C++
头文件	#include <string>	#include <string.h> #include <cstring>
定义	string 变量名 string s1;	char 变量名 char s1[10];
定义时赋初值	string s1="hello"; 长度无限制	char s1[10]="hello"; 长度不超过9
赋值	string s1; s1="hello"; 无限制 string s1="hello", s2; s2=s1;	char s1[10]; strcpy(s1, "hello"); 不超9 char s1[10]="hello", s2[10]; strcpy(s2, s1);
单字符操作	string s1="hello"; s1[2]='p';	char s1[10]="hello"; s1[2]='p';
输入	cin	cin、scanf
输出	cout	cout、printf
字符串复制	string s1, s2; ... s1=s2; 不必考虑溢出	char s1[10], s2[10]; ... strcpy(s1, s2); 防止溢出
字符串连接	string s1, s2; ... s1=s1+s2; 不必考虑溢出	char s1[20], s2[10]; ... strcat(s1, s2); 防止溢出
字符串比较	用比较运算符 s1==s2; s1>s2; s1>=s2;	用函数 if (!strcmp(s1, s2)) if (strcmp(s1, s2)>0) if (strcmp(s1, s2)>=0)

## § 5. 利用数组处理批量数据

### 5.6. C++处理字符串的方法-字符串类与字符串变量

#### 5.6.2. 字符串数组

形式:

```
string 数组名[正整型常量表达式];  
string name[5];
```

含义:

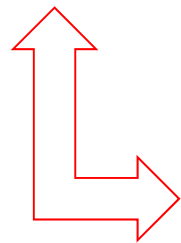
数组有若干元素, 每个元素是一个字符串变量

定义时赋初值:

```
string name[5]={"Zhang", "Li", "Wang", "Lin", "Zhao"};
```

与二维字符数组的内存表示比较: (定义时初始化)

```
char str[3][30]={"Zhang", "Li", "Wang"};  
string name[3]={"Zhang", "Li", "Wang"};
```



Z	h	a	n	g	\0	.....	\0
L	i	\0	\0	\0	\0	.....	\0
W	a	n	g	\0	\0	.....	\0

str占用连续的90个字节

Z	h	a	n	g
L	i			
W	a	n	g	

name[0]、name[1]、name[2]  
分别占用连续的5、2、4个字节, 但整体上不保证连续

## § 5. 利用数组处理批量数据

### 5.6. C++处理字符串的方法-字符串类与字符串变量

#### 5.6.2. 字符串数组

与二维字符数组的内存表示比较: (对键盘输入和用 `strcpy` / `=` 进行赋值的两种情况)

<pre>char str[3][30]; string name[3]; for (i=0; i&lt;3; i++) {     cin &gt;&gt; str[i];     cin &gt;&gt; name[i]; }</pre>	<pre>char str[3][30]; string name[3]; strcpy(str[0], "Zhang"); strcpy(str[1], "Li"); strcpy(str[2], "Wang"); name[0]="Zhang"; name[1]="Li"; name[2]="Wang";</pre>
通过键盘输入赋值 假设键盘输入: Zhang Zhang Li Li Wang Wang	通过语句赋值

Z	h	a	n	g	\0	?	.....	?
L	i	\0	?	?	?	?	.....	?
W	a	n	g	\0	?	?	.....	?

str占用连续的  
90个字节

Z	h	a	n	g
---	---	---	---	---

L	i
---	---

W	a	n	g
---	---	---	---

`name[0]`、`name[1]`、`name[2]`  
分别占用连续的5、2、4个字  
节, 但整体上不保证连续

P. 152 小字注释①, 看不懂可忽略  
RHEL7.1 VS2015

<code>sizeof(string)</code>	8	28
<code>sizeof(name)</code>	40	140

## § 5. 利用数组处理批量数据

### 5.6. C++处理字符串的方法-字符串类与字符串变量

#### 5.6.3. 字符串数组应用举例

P. 152-153 例5.11 字符串从小到大输出

例5.12 输入及查找学生

#### 例5.11

```
if (string2 > string3) { //保证串2<=串3
    temp = string2;
    string2 = string3;
    string3 = temp;
}

if (string1 <= string2)
    cout << "...1 2 3 ..."
else if (string1 <= string3) //已确定 1>2
    cout << "...2 1 3..."
else
    cout << "...2 3 1..."
```

string换成int后,  
处理过程完全一样  
=>string的好处



## § 5. 利用数组处理批量数据

### 5.6. C++处理字符串的方法-字符串类与字符串变量

#### 5.6.3. 字符串数组应用举例

P. 152-153 例5.11 字符串从小到大输出

例5.12 输入及查找学生

例5.12

```
void search(string find_name)
{   int i;
    bool flag=false;
    for(i=0; i<n; i++)
    {   if (name[i]==find_name) {
        cout << ... << endl;
        flag=true;
        break; //找到即退出
      }
    }
    if (flag==false)
        cout << "can't ..."
```

例5.12改进：不用bool型变量

```
void search(string find_name)
{   int i;
    bool flag=false;
    for(i=0; i<n; i++)
    {   if (name[i]==find_name) {
        cout << ... << endl;
        flag=true;
        break; //找到即退出
      }
    }
    if (flag==false (i>=n))
        cout << "can't ..."
```