# §5. 数组

## 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1：输入3个人的成绩，打印平均值

S中只保留了最后一次的输入

```
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;
}
```

例2：输入3个人的成绩，打印每个人的成绩及平均值

S中只保留了最后一次的输入

```
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 << endl;
       sum += s;
       }
    cout << "avg=" << sum/3.0 << endl;
    return 0;
}
```

例3：输入3个人的成绩，先打印平均值再打印成绩

输入/输出无法用循环,因为每次变量不同

```
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+s3+s3)/3.0 << endl;
    cout << "第1个:" << s1 << endl;
    cout << "第2个:" << s2 << endl;
    cout << "第3个:" << s3 << endl;
    return 0;
}
```

10000人 ？

人数不定 ？

# §5. 数组

## 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]; ×
```

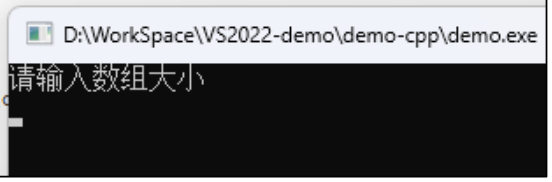
● 说明：早期C/C++标准中，数组大小必须是常量，新标准已允许为变量，为统一，此处仍要求为常量

```
#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "请输入数组大小" << endl;
    cin >> n;
    int a[n];
    return 0;
}
```

VS 编译：×
Dev编译：✓

```
已启动生成...
1>------ 已启动生成: 项目: demo-cpp, 配置: Debug Win32 ------
1>demo.cpp
1>D:\WorkSpace\VS2022-demo\demo-cpp\demo.cpp(8,11): error C2131: 表达式的计算结果不是常数
1>D:\WorkSpace\VS2022-demo\demo-cpp\demo.cpp(8,11): message : 因读取超过生命周期的变量而失败
1>D:\WorkSpace\VS2022-demo\demo-cpp\demo.cpp(8,11): message : 请参见“n”的用法
1>已完成生成项目“demo-cpp.vcxproj”的操作 - 失败。
========== 生成: 0 成功，1 失败，0 最新，0 已跳过 ==========
```

```
编译结果...
--------
- 错误: 0
- 警告: 0
- 输出文件名: D:\WorkSpace\VS2022-demo\d
- 输出大小: 2.33393955230713 MiB
- 编译时间: 0.63s
```



# §5. 数组

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

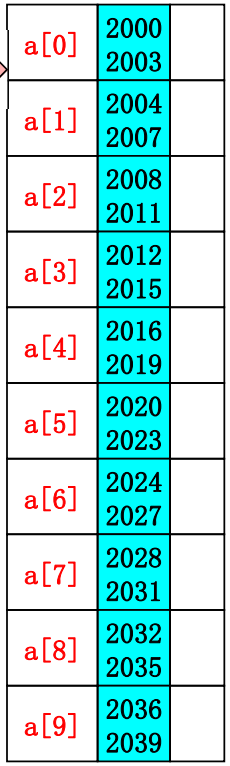
★ 若数组定义中整型常量表达式为n，表示有n个元素，则称数组长度应为n

★ 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：




★ 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Dev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. 数组

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++)<br>✓ 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;
}
```

思考：将int a[10]改为全局变量，前10行输出分别是什么？

10个不确定值
一个6/8位的16进制数
表示数组的首地址

Microsoft Visual Studio 调试控制台

```
-858993460
-858993460
-858993460
-858993460
-858993460
-858993460
-858993460
-858993460
-858993460
-858993460
0101F9F8
```

VS

D:\WorkSpace\VS2019-Demo\cpp-demo\cpp-demo.exe

```
-2
7929704
4199663
4242320
65535
0
0
47
8981760
7929704
0x78fe94
```

Dev

# §5. 数组

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）
```

VS下均有warning
观察执行后的错误表现
其余编译器自行观察

```
#include <iostream>
using namespace std;
int main()
{
    int k, a[10];
    k = 10 + a[10] * 2;
    cout << k << endl;
    return 0;
}//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;
    return 0;
} //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;
    return 0;
}//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;
    return0;
}//直接被操作系统终止
```

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. 数组

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） |

```
#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;
    return 0;
}
```

VS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Dev，观察运行结果

VS有IntelliSense，
结果为：
k=321
a[12]=123
k=123

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

int main()
{
    int k=321, a[10];
    cout << "k=" << k << endl;
    a[10] = 123;
    cout << "a[10]=" << a[10] << endl;
    cout << "k=" << k << endl;
    return 0;
}
```

用VS执行，观察运行结果

用Dev执行，观察运行结果

# §5. 数组

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};

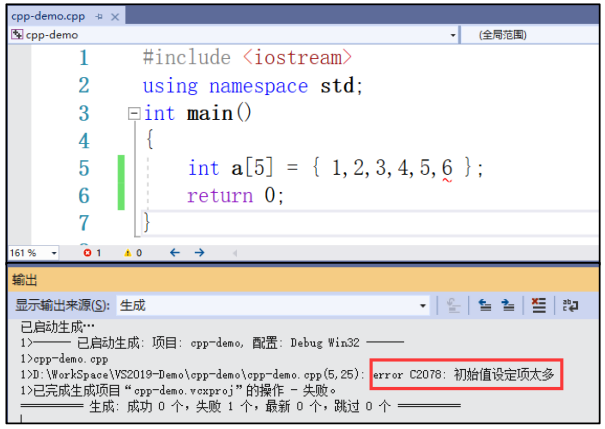
| | | |
|---|---|---|
| a[0] | 2000<br>2003 | 1 |
| a[1] | 2004<br>2007 | 2 |
| a[2] | 2008<br>2011 | 3 |
| a[3] | 2012<br>2015 | 4 |
| a[4] | 2016<br>2019 | 5 |
| a[5] | 2020<br>2023 | 6 |
| a[6] | 2024<br>2027 | 7 |
| a[7] | 2028<br>2031 | 8 |
| a[8] | 2032<br>2035 | 9 |
| a[9] | 2036<br>2039 | 10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

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

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

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

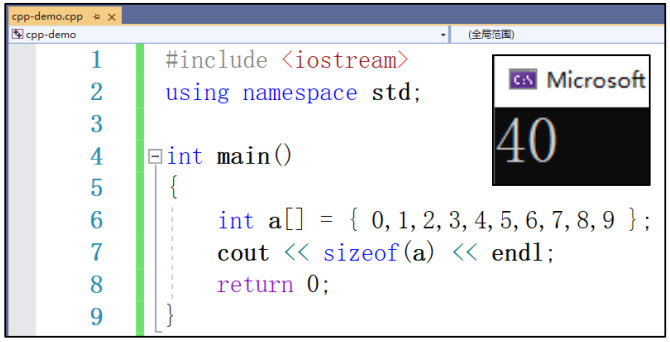
错误，因为初始化的个数超过了数组的大小




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）

如何验证？ => sizeof(a) 期望值40



# §5. 数组

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

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

```
int main()
{
    int i, a[10]={5,-2, 7 };
    for (i=0; i<10; i++)
        cout << a[i] << ' ';
    return 0;
}
```

Microsoft Visual Studio 调试控制台

5 -2 7 0 0 0 0 0 0 0

```
int a[1000]={0};    //全0
```

长度不可省略

```
#include <iostream>
using namespace std;
int main()
{
    int a[1000]={ 0 };
    cout << a[0] << endl;
    cout << a[1] << endl;
    cout << a[999] << endl;
    return 0;
}
```

Microsoft

0
0
0

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

注意：不是全1

```
#include <iostream>
using namespace std;
int main()
{
    int a[1000]={ 1 };
    cout << a[0] << endl;
    cout << a[1] << endl;
    cout << a[999] << endl;
    return 0;
}
```

Microsoft

1
0
0

```
int a[1000];    希望a[0] - a[999] 为 0 - 999
方法1：int a[1000] = {0,1,2,...,999};
方法2：int a[1000], i;
       for(i=0; i<1000; i++)
           a[i] = i;
```

语句太长，一般不用

不是定义时初始化，而是通过执行语句进行赋值操作

# §5. 数组

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
int main()
{   int b[100]; //不确定
}
```

```
#include <iostream>
using namespace std;
int a[1000];
int main()
{
    int b[100];
    cout << a[0] << ' ' << b[0] << endl;
    cout << a[9] << ' ' << b[9] << endl;
    return 0;
}
```

Microsoft Visual Studio 调试控制台

```
0 -858993460
0 -858993460
```

# §5. 数组

## 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

输入/输出无法用循环,因为每次变量不同

```
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+s3+s3)/3.0 << endl;
    cout << "第1个:" << s1 << endl;
    cout << "第2个:" << s2 << endl;
    cout << "第3个:" << s3 << endl;
    return 0;
}
```

例3：输入3个人的成绩，先打印平均值再打印成绩

循环方式

```
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;
}
```

例3：输入3个人的成绩，先打印平均值再打印成绩

10000人

```
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;
}
```

例3：输入3个人的成绩，先打印平均值再打印成绩

10000人,更合理方便修改及维护

```
#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;
}
```

# §5. 数组



5.2. 一维数组的定义和引用

5.2.4. 应用

例：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;
}
```

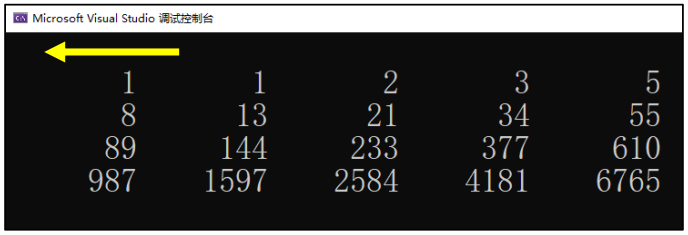
和前例相比

1、前例：边计算边输出
　　本例：先计算后输出

2、前例：输出后值不保留，无法回溯
　　本例：值全部保留，可回溯

空行

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




思考：
如果不希望空第一行，
如何实现？



例：求 Fibonacci 数列的前40项

本程序的不足之处：无法回溯
（当f1表示第7个月后，第5个月的值无法再现）

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int i, f1=1, f2=1;    //初值
    for(i=1; i<=20; i++) { //每次2数，20次=40个
              cout << setw(12) << f1 << setw(12) << f2;
       if (i%2==0)
          cout << endl; //每2次(4个)加换行
       f1 = f1 + f2;     //f1为第3/5/7/...个月
       f2 = f1 + f2;     //f2为第4/6/8/...个月
       }
    return 0;
}
```

# §5. 数组

5.2. 一维数组的定义和引用

5.2.4. 应用

例：冒泡法排序

```
#include <iostream>
using namespace std;
#define  N  10
int main()
{   int a[N] = { 3,7,29,23,12,82,1,72,8,5 };
    int i, j, t;

    for (i=0; i<N-1; i++)
        for (j=0; j<N-(i+1); j++)
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                }

    for (i=0; i<N; i++)
        cout << a[i] << ' ';
    cout << endl;
}
```

两数交换方法

内外循环的关系：

外循环i=0，内循环j=0~8，比较9次

外循环i=1，内循环j=0~7，比较8次

...

...

外循环i=8，内循环j=0~0，比较1次

# §5. 数组

5.2. 一维数组的定义和引用

5.2.4. 应用

例：冒泡法排序

```
#include <iostream>
using namespace std;
#define  N  10
int main()
{   int a[N] = { 3,7,29,23,12,82,1,72,8,5 };
    int i, j, t;

    for (i=0; i<N-1; i++)
        for (j=0; j<N-(i+1); j++)
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                }

    for (i=0; i<N; i++)
        cout << a[i] << ' ';
    cout << endl;
}
```

两数交换方法

设a[10]为{3,7,29,23,12,82,1,72,8,5}

第1次，外循环i=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. 数组

5.2. 一维数组的定义和引用

5.2.4. 应用

例：冒泡法排序

```
#include <iostream>
using namespace std;
#define  N  10
int main()
{   int a[N] = { 3,7,29,23,12,82,1,72,8,5 };
    int i, j, t;

    for (i=0; i<N-1; i++)
        for (j=0; j<N-(i+1); j++)
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                }

    for (i=0; i<N; i++)
        cout << a[i] << ' ';
    cout << endl;
}
```

两数交换方法

设a[10]为{3,7,29,23,12,82,1,72,8,5}

第1次，外循环i=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. 数组

5.2. 一维数组的定义和引用

5.2.4. 应用

例：冒泡法排序

```
#include <iostream>
using namespace std;
#define  N  10
int main()
{   int a[N] = { 3,7,29,23,12,82,1,72,8,5 };
    int i, j, t;

    for (i=0; i<N-1; i++)
        for (j=0; j<N-(i+1); j++)
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                }

    for (i=0; i<N; i++)
        cout << a[i] << ' ';
    cout << endl;
}
```

两数交换方法

设a[10]为{3,7,29,23,12,82,1,72,8,5}

第1次，外循环i=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. 数组

5.2. 一维数组的定义和引用

5.2.4. 应用

例：冒泡法排序

```
#include <iostream>
using namespace std;
#define  N  10
int main()
{   int a[N] = { 3,7,29,23,12,82,1,72,8,5 };
    int i, j, t;

    for (i=0; i<N-1; i++)
        for (j=0; j<N-(i+1); j++)
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                }

    for (i=0; i<N; i++)
        cout << a[i] << ' ';
    cout << endl;
}
```

两数交换方法

设a[10]为{3,7,29,23,12,82,1,72,8,5}

第1次，外循环i=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. 数组

5.2. 一维数组的定义和引用

5.2.4. 应用

例：冒泡法排序

```
#include <iostream>
using namespace std;
#define  N  10
int main()
{   int a[N] = { 3,7,29,23,12,82,1,72,8,5 };
    int i, j, t;

    for (i=0; i<N-1; i++)
        for (j=0; j<N-(i+1); j++)
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                }

    for (i=0; i<N; i++)
        cout << a[i] << ' ';
    cout << endl;
}
```

两数交换方法

设a[10]为{3,7,29,23,12,82,1,72,8,5}

第1次，外循环i=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. 数组

5.2. 一维数组的定义和引用

5.2.4. 应用

例：冒泡法排序

```
#include <iostream>
using namespace std;
#define  N  10
int main()
{   int a[N] = { 3,7,29,23,12,82,1,72,8,5 };
    int i, j, t;

    for (i=0; i<N-1; i++)
        for (j=0; j<N-(i+1); j++)
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                }

    for (i=0; i<N; i++)
        cout << a[i] << ' ';
    cout << endl;
}
```

两数交换方法

设a[10]为{3,7,29,23,12,82,1,72,8,5}

第1次，外循环i=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. 数组

5.2. 一维数组的定义和引用

5.2.4. 应用

例：冒泡法排序

```
#include <iostream>
using namespace std;
#define  N  10
int main()
{   int a[N] = { 3,7,29,23,12,82,1,72,8,5 };
    int i, j, t;

    for (i=0; i<N-1; i++)
        for (j=0; j<N-(i+1); j++)
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                }

    for (i=0; i<N; i++)
        cout << a[i] << ' ';
    cout << endl;
}
```

两数交换方法

设a[10]为{3,7,29,23,12,82,1,72,8,5}

第1次，外循环i=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. 数组

5.2. 一维数组的定义和引用

5.2.4. 应用

例：冒泡法排序

```
#include <iostream>
using namespace std;
#define  N  10
int main()
{   int a[N] = { 3,7,29,23,12,82,1,72,8,5 };
    int i, j, t;

    for (i=0; i<N-1; i++)
        for (j=0; j<N-(i+1); j++)
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                }

    for (i=0; i<N; i++)
        cout << a[i] << ' ';
    cout << endl;
}
```

两数交换方法

设a[10]为{3,7,29,23,12,82,1,72,8,5}

第1次，外循环i=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. 数组

5.2. 一维数组的定义和引用

5.2.4. 应用

例：冒泡法排序

```
#include <iostream>
using namespace std;
#define  N  10
int main()
{   int a[N] = { 3,7,29,23,12,82,1,72,8,5 };
    int i, j, t;

    for (i=0; i<N-1; i++)
        for (j=0; j<N-(i+1); j++)
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                }

    for (i=0; i<N; i++)
        cout << a[i] << ' ';
    cout << endl;
}
```

两数交换方法

设a[10]为{3,7,29,23,12,82,1,72,8,5}

第1次，外循环i=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 | | | | | | | | 82 |
| 9 | 5 | | | | | | | | |

# §5. 数组

5.2. 一维数组的定义和引用

5.2.4. 应用

例：冒泡法排序

```
#include <iostream>
using namespace std;
#define  N  10
int main()
{   int a[N] = { 3,7,29,23,12,82,1,72,8,5 };
    int i, j, t;

    for (i=0; i<N-1; i++)
        for (j=0; j<N-(i+1); j++)
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                }

    for (i=0; i<N; i++)
        cout << a[i] << ' ';
    cout << endl;
}
```

两数交换方法

设a[10]为{3,7,29,23,12,82,1,72,8,5}

第1次，外循环i=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 | | | | | | | | 82 | 5 |
| 9 | 5 | | | | | | | | | 82 |

# §5. 数组

5.2. 一维数组的定义和引用

5.2.4. 应用

例：冒泡法排序

```
#include <iostream>
using namespace std;
#define  N  10
int main()
{   int a[N] = { 3,7,29,23,12,82,1,72,8,5 };
    int i, j, t;

    for (i=0; i<N-1; i++)
        for (j=0; j<N-(i+1); j++)
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                }

    for (i=0; i<N; i++)
        cout << a[i] << ' ';
    cout << endl;
}
```

两数交换方法

设a[10]为{3,7,29,23,12,82,1,72,8,5}

第1次，外循环i=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 | 8 | | | | | | | | 82 | 5 | 5 |
| 9 | 5 | | | | | | | | | 82 | 82 |

第1次外循环后，最大数82在最后

# §5. 数组

5.2. 一维数组的定义和引用

5.2.4. 应用

例：冒泡法排序

```
#include <iostream>
using namespace std;
#define  N  10
int main()
{   int a[N] = { 3,7,29,23,12,82,1,72,8,5 };
    int i, j, t;

    for (i=0; i<N-1; i++)
        for (j=0; j<N-(i+1); j++)
            if (a[j] > a[j+1]) {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                }

    for (i=0; i<N; i++)
        cout << a[i] << ' ';
    cout << endl;
}
```

两数交换方法

设a[10]为{3,7,29,23,12,82,1,72,8,5}

第1次，外循环i=0

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

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. 数组

## 5.3. 二维数组的定义和引用

### 5.3.1. 定义

数据类型 数组名[正整型常量表达式1][正整型常量表达式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
#define N 10
int a[M][N] ✓
```

```
const int m=5,n=10;
int a[m][n]; ✓
```

```
int i=5,j=10;
int a[i][j]; ×
```

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

```
#include <iostream>
using namespace std;
int main()
{
    int m, n;
    cout << "请输入数组大小" << endl;
    cin >> m >> n;
    int a[m][n];
    return 0;
}
```

同一维数组

VS 编译：×
Dev编译：✓

```
已启动生成...
1>------ 已启动生成: 项目: demo-cpp, 配置: Debug Win32 ------
1>demo.cpp
1>D:\WorkSpace\VS2022-demo\demo-cpp\demo.cpp(8,11): error C2131: 表达式的计算结果不是常数
1>D:\WorkSpace\VS2022-demo\demo-cpp\demo.cpp(8,11): message : 因读取超过生命周期的变量而失败
1>D:\WorkSpace\VS2022-demo\demo-cpp\demo.cpp(8,11): message : 请参见“m”的用法
1>D:\WorkSpace\VS2022-demo\demo-cpp\demo.cpp(8,14): error C2131: 表达式的计算结果不是常数
1>D:\WorkSpace\VS2022-demo\demo-cpp\demo.cpp(8,14): message : 因读取超过生命周期的变量而失败
1>D:\WorkSpace\VS2022-demo\demo-cpp\demo.cpp(8,14): message : 请参见“n”的用法
1>已完成生成项目“demo-cpp.vcxproj”的操作 - 失败。
========== 生成: 0 成功，1 失败，0 最新，0 已跳过 ==========
```

```
编译结果...
--------
- 错误: 0
- 警告: 0
- 输出文件名: D:\WorkSpace\VS2022-demo\
- 输出大小: 2.33393955230713 MiB
- 编译时间: 0.61s
```

```
D:\WorkSpace\VS2022-demo\demo-cpp\demo.exe
请输入数组大小
```

# §5. 数组

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

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

```
48
4
```

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：

| | |
|---|---|
| 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 |

# §5. 数组

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

理解1：a [3][4]

理解2：a[3] [4]

```
#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;
    return 0;
}
```

```
48
16
4
```

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

# §5. 数组

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（数组的首地址）

★ 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. 数组

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

```
#include <iostream>
using namespace std;
int main()
{
    int i,j;
    int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
    for (i=0; i<3; i++) {
        for (j=0; j<4; j++)
            cout << a[i][j] << ' ';
        cout << endl;
        }
    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
1 2 3 4
5 6 7 8
9 10 11 12
```

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

cpp-demo.cpp

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a[3][4]= { 1,2,3,4,5,6,7,8,9,10,11,12,13 };
6      return 0;
7  }
```

注意：波浪号的位置是第13个

error C2078: 初始值设定项太多

# §5. 数组

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

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}};
```

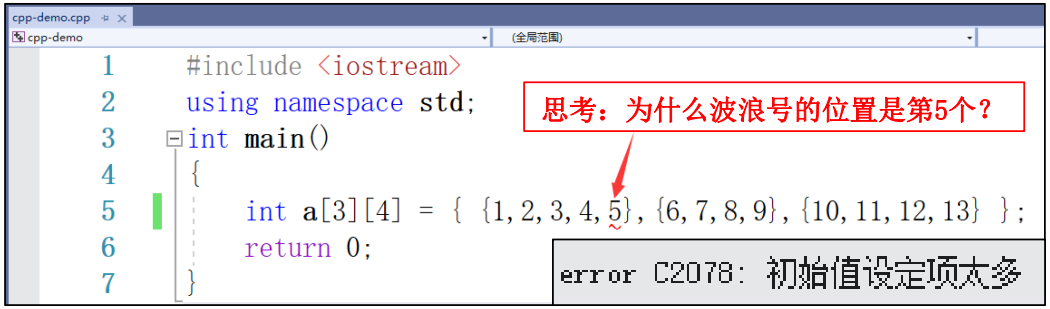
更直观

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

★ 如果采用双层括号方式，则内括号中元素的个数不超过列数，成对的内括号的总数不超过行数

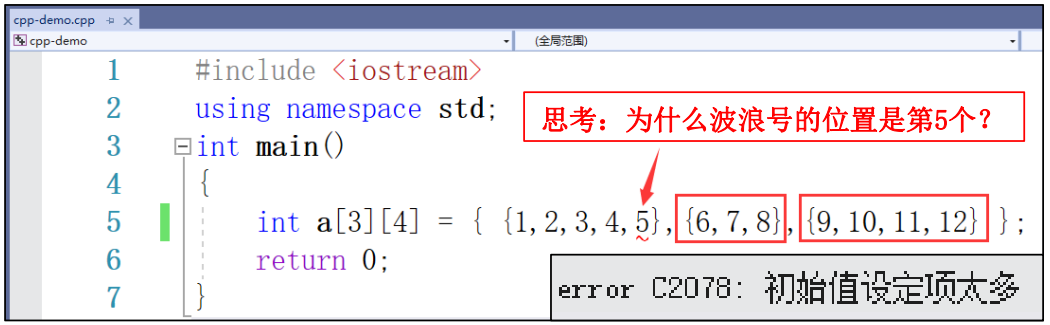
```
int a[3][4]={{1,2,3,4,5},{5,6,7,8},{9,10,11,12}};
```

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



# §5. 数组

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

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

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  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a[3][] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
6      return 0;
7  }
```

(5,16): error C2087: "a": 缺少下标
(5,26): error C2078: 初始值设定项太多

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a[3][] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
6      return 0;
7  }
```

(5,16): error C2087: "a": 缺少下标
(5,23): error C2078: 初始值设定项太多

问：为什么只能行缺省而不能列缺省？

答：二维表形式：{行缺省，行 = 总数/列
列缺省，列 = 总数/行

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. 数组

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. 数组

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};

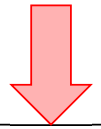
3(省略)

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
    return 0;
}
```

# §5. 数组

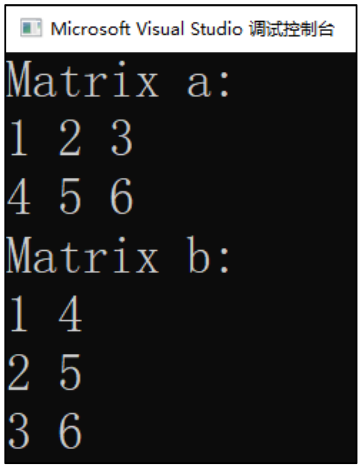
5.3. 二维数组的定义和引用

5.3.4. 应用

例：矩阵转置（2x3 转置为 3x2）

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

int main()
{
    int a[2][3] = { {1,2,3},{4,5,6} };
    int b[3][2], i, j;

    cout << "Matrix a:" << endl;
    for (i=0; i<2; i++) {
        for (j=0; j<3; j++)
            cout << a[i][j] << ' ';
        cout << endl;
    }
    for (i=0; i<2; i++)
        for (j=0; j<3; j++)
            b[j][i] = a[i][j];  //转置

    cout << "Matrix b:" << endl;
    for (i=0; i<3; i++) {
        for (j=0; j<2; j++)
            cout << b[i][j] << ' ';
        cout << endl;
    }

    return 0;
}
```

```
#include <iostream>
using namespace std;
const int M=2, N=3;
int main()
{
    int a[M][N] = { {1,2,3},{4,5,6} };
    int b[N][M], i, j;

    cout << "Matrix a:" << endl;
    for (i=0; i<M; i++) {
        for (j=0; j<N; j++)
            cout << a[i][j] << ' ';
        cout << endl;
    }
    for (i=0; i<M; i++)
        for (j=0; j<N; j++)
            b[j][i] = a[i][j];  //转置

    cout << "Matrix b:" << endl;
    for (i=0; i<N; i++) {
        for (j=0; j<M; j++)
            cout << b[i][j] << ' ';
        cout << endl;
    }

    return 0;
}
```




1. 用宏定义/常变量表示2/3更好
2. C/C++中的数组范围表示，一般遵循前闭后开的规则

```
for(i=0; i<=1; i++)
for(i=0; i<2; i++)   √
```

# §5. 数组

5.3. 二维数组的定义和引用

5.3.4. 应用

例：求最大值及所在行列(如有相同，按先行后列取第一个)

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

const int M=3, N=4;
int main()
{   int a[M][N] = { {7, -3, 12, 19},
                    {-23, 17, 92, 78} ,
                    {2, -1, 5, 92} };
    int i, j;
    int max=a[0][0], row=0, col=0;//初始认为最大值为a[0][0]

    for (i=0; i<M; i++)
        for (j=0; j<N; 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;
            }

    cout << "max=" << max << endl;
    cout << "row=" << row << endl;
    cout << "col=" << col << endl;

    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
max=92
row=1
col=2
```

例：求最大值及所在行列(如有相同，按先行后列取最后一个)

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

const int M=3, N=4;
int main()
{   int a[M][N] = { {7, -3, 12, 19},
                    {-23, 17, 92, 78} ,
                    {2, -1, 5, 92} };
    int i, j;
    int max=a[0][0], row=0, col=0;//初始认为最大值为a[0][0]

    for (i=0; i<M; i++)
        for (j=0; j<N; 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;
            }

    cout << "max=" << max << endl;
    cout << "row=" << row << endl;
    cout << "col=" << col << endl;

    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
max=92
row=2
col=3
```

# §5. 数组

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. 数组

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

例：数组元素做参数求最大值及所在行列

同前例，将3/4换为M/N更合理

例：求最大值（如果最大值有相同，按先行后列取第一个）

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

int main()
{
    int a[3][4] = { {7, -3, 12, 19},
                    {-23, 17, 92, 78},
                    {2, -1, 5, 92} };
    int i, j, max=a[0][0]; //初始最大值为a[0][0]

    for(i=0; i<3; i++)
       for(j=0; j<4; j++)
          max=max_value(a[i][j], max);

    cout << "max=" << max << endl;

    return 0;
}
```

实参:二维数组元素(int型)
形参:int型简单变量

例：求最大值及所在行列（如果最大值有相同，按先行后列取第一个）

```
int main()
{
    int a[3][4] = { {7, -3, 12, 19},
                    {-23, 17, 92, 78},
                    {2, -1, 5, 92} };
    int i, j, max=a[0][0], row=0, col=0; //初始最大值为a[0][0]
    for (i=0; i<3; i++)
        for (j=0; j<4; j++)
            if (a[i][j] > max) {
                max = a[i][j]; //替换新的max
                row = i;       //同时记录行/列值
                col = j;
            }
    cout << "max=" << max << endl;
    cout << "row=" << row << endl;
    cout << "col=" << col << endl;
    return 0;
}
```

例：求最大值（如果最大值有相同，按先行后列取第一个）

```
int main()
{
    int a[3][4] = { {7, -3, 12, 19},
                    {-23, 17, 92, 78},
                    {2, -1, 5, 92} };
    int i, j, max=a[0][0]; //初始最大值为a[0][0]
    for (i=0; i<3; i++)
        for (j=0; j<4; j++)
            if (a[i][j] > max) {
                max = a[i][j]; //替换新的max

            }
    cout << "max=" << max << endl;
    return 0;
}
```

# §5. 数组

5.4. 用数组名作函数参数

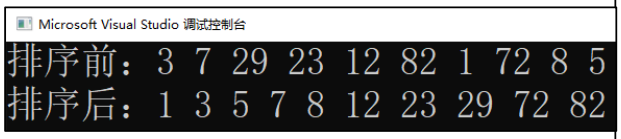
5.4.1. 用数组元素做函数实参

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

例：选择法排序

```
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;
       }
}

int main()
{
    const int N = 10;
    int a[N]={3,7,29,23,12,82,1,72,8,5};
    cout << "排序前：";
    for (i = 0; i < N; i++)
        cout << a[i] << ' ';
    cout << endl;

    select_sort(a, N);

    cout << "排序后：";
    for (i = 0; i < N; i++)
        cout << a[i] << ' ';
    cout << endl;
    return 0;
}
```



# §5. 数组

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

例：选择法排序

```
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;
       }
}

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

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. 数组

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

例：选择法排序

```
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;
       }
}

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

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 | | | | | | | | | | |
| 3 | | | | | | | | | | |
| 4 | | | | | | | | | | |
| 5 | | | | | | | | | | |
| 6 | | | | | | | | | | |
| 7 | | | | | | | | | | |
| 8 | | | | | | | | | | |
| 9 | | | | | | | | | | |

# §5. 数组

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

例：选择法排序

```
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;
       }
}

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

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. 数组

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

例：选择法排序

```
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;
       }
}

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

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. 数组

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

例：选择法排序

```
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;
       }
}

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

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 | | | | | | | | | | |
| 6 | | | | | | | | | | |
| 7 | | | | | | | | | | |
| 8 | | | | | | | | | | |
| 9 | | | | | | | | | | |

# §5. 数组

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

例：选择法排序

```
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;
       }
}

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

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. 数组

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

例：选择法排序

```
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;
       }
}

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

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 | | | | | | | | | | |
| 8 | | | | | | | | | | |
| 9 | | | | | | | | | | |

# §5. 数组

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

例：选择法排序

```
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;
       }
}

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

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. 数组

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

例：选择法排序

```
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;
       }
}

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

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. 数组

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

例：选择法排序

```
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;
       }
}

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

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. 数组

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

例：选择法排序

```
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;
       }
}

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

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. 数组

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

例：选择法排序

```
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;
       }
}

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

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. 数组

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

这些内容与第04模块中简单变量做实形参的概念不同，第06模块指针中再详细解释

★ 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，因此实形参类型必须完全相同，否则编译错

# §5. 数组

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

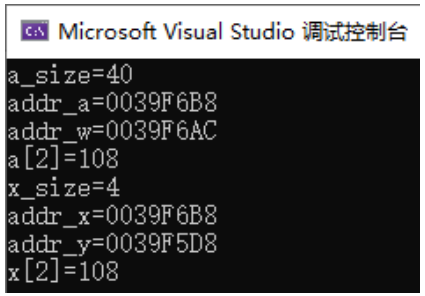
★ 形参为相应类型的一维数组

★ 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)

```
//数组名/普通变量做函数参数，验证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. 数组

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;
}
```

```
Microsoft Visual Studio 调试控制台
18 19
37 19
```

# §5. 数组

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

```
//形参数组不指定大小/指定大小与实参数组相同/不同，验证sizeof的值是否相同
#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);
}
```

```
Microsoft Visual Studio 调试控制台
a_size=40
x1_size=4
x2_size=4
x3_size=4
```

(为什么是4后续会解释)

```
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. 数组

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

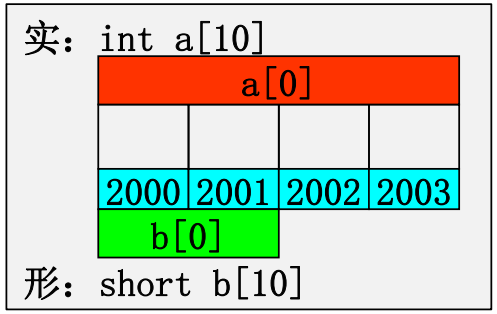
★ 实参传递时，将实参数组的首地址(数组名表示数组的首地址)传给形参，因此实、形参数组的内存地址重合
(实参占用空间，形参不占用空间)

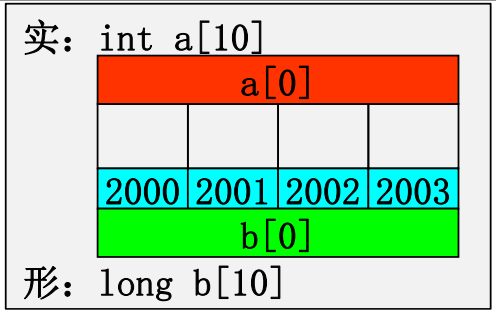
★ 形参数组值的改变会影响到实参(与简单参数不同)

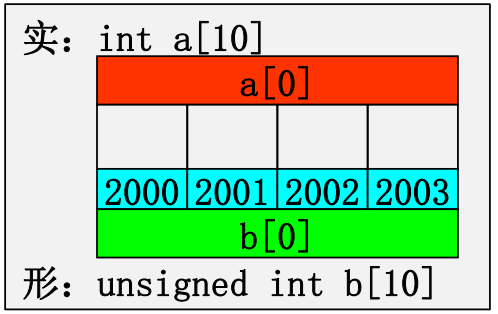
★ 因为形参数组不分配空间，因此数组大小可不指定

★ 因为形参数组不分配空间，因此实形参类型必须完全相同，否则编译错

```
#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
}
```




```
error C2664: “void f1(short [])”: 无法将参数 1 从“int [10]”转换为“short []”
message : 与指向的类型无关；强制转换要求 reinterpret_cast、C 样式强制转换或函数样式强制转换
message : 参见“f1”的声明
error C2664: “void f2(long [])”: 无法将参数 1 从“int [10]”转换为“long []”
message : 与指向的类型无关；强制转换要求 reinterpret_cast、C 样式强制转换或函数样式强制转换
message : 参见“f2”的声明
error C2664: “void f3(unsigned int [])”: 无法将参数 1 从“int [10]”转换为“unsigned int []”
message : 与指向的类型无关；强制转换要求 reinterpret_cast、C 样式强制转换或函数样式强制转换
message : 参见“f3”的声明
```

# §5. 数组

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

例：求最大值（如果最大值有相同，按先行后列取第一个）

```
int max_value(int array[][4])
{
    int i, j, max=array[0][0]; //初始认为最大值为a[0][0]
    for(i=0; i<3; i++)
       for(j=0; j<4; j++)
           if (array[i][j] > max)
               max = array[i][j];
    return max;
}
int main()
{   int a[3][4] = { {7, -3, 12, 19}, {-23, 17, 92, 78} ,{2, -1, 5, 92} };
    int max = max_value(a);
    cout << "max=" << max <<endl;
}
```

实参:二维数组名
形参:二维数组

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

三种方式都正确

```
//形参数组不指定大小/指定大小与实参数组相同/不同，
//验证sizeof的值是否相同
#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);
    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
a_size=48
x1_size=4
x2_size=4
x3_size=4
```

（为什么是4后续会解释）

# §5. 数组

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

例：求最大值（如果最大值有相同，按先行后列取第一个）

```
int max_value(int array[][4])
{
    int i, j, max=array[0][0]; //初始认为最大值为a[0][0]
    for(i=0; i<3; i++)
       for(j=0; j<4; j++)
           if (array[i][j] > max)
               max = array[i][j];
    return max;
}
int main()
{   int a[3][4] = { {7, -3, 12, 19}, {-23, 17, 92, 78} ,{2, -1, 5, 92} };
    int max = max_value(a);
    cout << "max=" << max <<endl;
}
```

实参:二维数组名
形参:二维数组

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

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.数组

## 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，具体见后续模块

# §5.数组

## 5.5.字符数组

第02模块中的概念：
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[正整型常量表达式1][正整型常量表达式2]　　二维数组
unsigned char 数组名[正整型常量表达式1][正整型常量表达式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. 数组

## 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(十进制)

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

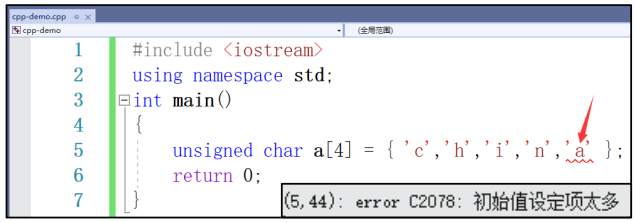
```
char a[5]={'\x63','\x68','\x69','\x6e','\x61'};
```
16进制转义符形式

```
char a[5]={0x63, 104, 105, 110, 0141};
```
整数形式(多进制)

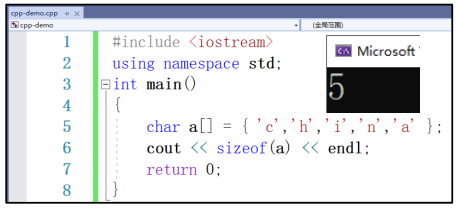
多种表示方法等价，内存占5字节，表示如下：

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

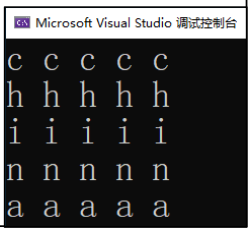
```
unsigned char a[4]={'c','h','i','n','a'};
```
若个数多，则错误




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




```
int main()
{
    char a1[5] = { 'c','h','i','n','a' };
    char a2[5] = { 99, 104, 105, 110, 97 };
    char a3[5] = { '\143','\150','\151','\156','\141' };
    char a4[5] = { '\x63','\x68','\x69','\x6e','\x61' };
    char a5[5] = { 0x63, 104, 105, 110, 0141 };

    for (int i = 0; i < 5; i++) {
        cout << a1[i] << ' ';
        cout << a2[i] << ' ';
        cout << a3[i] << ' ';
        cout << a4[i] << ' ';
        cout << a5[i] << endl;
        }
    return 0;
}
```




```
int main()
{
    char a1[5] = { 'c','h','i','n','a' };
    char a2[5] = { 99, 104, 105, 110, 97 };
    char a3[5] = { '\143','\150','\151','\156','\141' };
    char a4[5] = { '\x63','\x68','\x69','\x6e','\x61' };
    char a5[5] = { 0x63, 104, 105, 110, 0141 };
    for (int i = 0; i < 5; i++) {
        cout << int(a1[i]) << ' ';
        cout << int(a2[i]) << ' ';
        cout << int(a3[i]) << ' ';
        cout << int(a4[i]) << ' ';
        cout << int(a5[i]) << endl;
        }
    return 0;
}
```

# §5.数组

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. 数组

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'

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

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

```
#include <iostream>
using namespace std;
int main()
{
    int i, j;
    char a[3][3] = { 'a','b','c' };
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++)
            cout << int(a[i][j]) << ' ';
        cout << endl;
    }
}
```

Microsoft Visual Studio 调试控制台

```
97 98 99
0 0 0
0 0 0
```

```
#include <iostream>
using namespace std;
int main()
{
    int i, j;
    char a[3][3] = {{'a'},{'b'},{'c'}};
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++)
            cout << int(a[i][j]) << ' ';
        cout << endl;
    }
}
```

Microsoft Visual Studio 调试控制台

```
97 0 0
98 0 0
99 0 0
```

# §5.数组

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" =>(体现出变量特性)

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. 数组

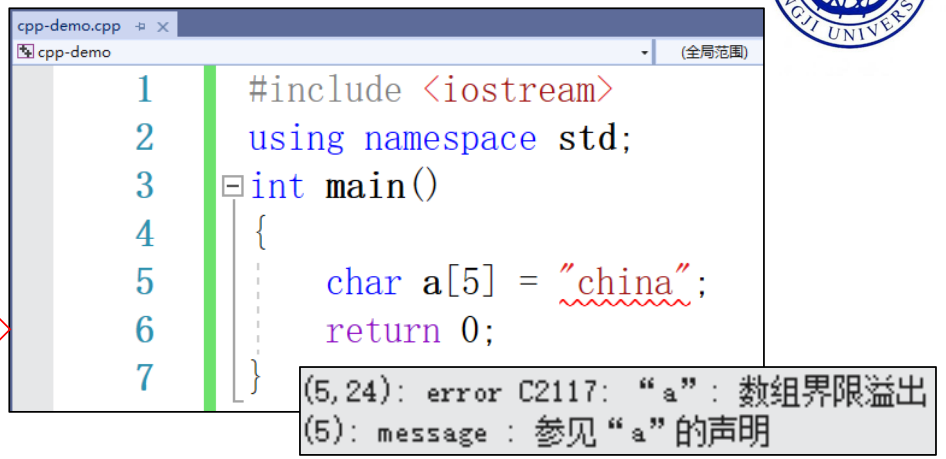
5.5. 字符数组

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"}; //双引号方式，数组长度为字符串长度+1
char a[6]="china";   //大括号可省略
char a[5]="china";   //数组大小未计算尾零位置则编译报错
```




若初始化时不给出数组大小，则字符串方式要算\0位置(字符方式不需要)

```
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'};
    cout << sizeof(a) << endl;
    cout << sizeof(b) << endl;
    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
6
5
```

# §5. 数组

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

B. 部分初始化

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

a[5]-a[9]为'\0'

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

Microsoft Visual Studio 调试控制台

```
99 104 105 110 97 0 0 0 0 0 #
```

C. 多个字符串初始化二维字符数组

```
char a[3][6] = {"hello", "tong", "ji"};
char a[3][6] = {{"hello"}, {"tong"}, {"ji"}};
char a[][6] = {"hello", "tong", "ji"};
char a[][6] = {{"hello"}, {"tong"}, {"ji"}};
```

| | | | | | |
|---|---|---|---|---|---|
| h | e | l | l | o | \0 |
| t | o | n | g | \0 | \0 |
| j | i | \0 | \0 | \0 | \0 |

★ 串的个数不超过行大小，串长+1不超过列大小

★ 若初始化时不给出行，则缺省为串的个数

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

有错，为什么？

# §5.数组

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.数组

5.5.字符数组

5.5.5.字符数组的输入与输出

5.5.5.1.字符的输出

★ 第2章补充视频 - 字符与字符串的输出（自行回顾）

§2.基础知识

补充2：字符的输出

★ (前序)将默认输出窗口从Windows终端改为Windows控制台主机

★ 有效的输出

- 纯ASCII图形字符的输出(基本ASCII码 33-126之间)
- 汉字的输出(成对的扩展ASCII码)
- 汉字+纯ASCII图形字符的混合输出

★ 无效的输出

- 单独的扩展ASCII码的输出(中文系统下非法)
- 非法的情况(不成对的扩展ASCII码、含控制字符)
- 同一程序设置不同显示字体的情况下

★ 尾零的输出

- 不同控制台形式、不同字体下显示不同，最终结论：\0不是合法的图形字符，看到的内容不可信!!!

★ 总结

- 含汉字/图形字符/空格的字符串可以正常显示，否则输出不可信
- 想研究长度/占内存空间的做法：strlen、sizeof
- 想得到某个字节确定值的做法：转int打印

# §5.数组

5.5.字符数组

5.5.5.字符数组的输入与输出

5.5.5.1.字符的输出

★ 第2章补充视频 - 字符与字符串的输出（自行回顾）

5.5.5.2.字符数组的输出

★ 字符数组中每个元素的单独输出

★ 以字符串方式输出含\0的一维字符数组

★ 以字符串方式输出不含\0的一维字符数组

# §5. 数组

5.5. 字符数组

5.5.5. 字符数组的输入与输出

5.5.5.1. 字符的输出

5.5.5.2. 字符数组的输出

★ 字符数组中每个元素的单独输出（注意事项同字符的输出）

- 不要以字符形式输出\0及其它非图形字符(看到的内容不可信)
- 如想准确得知字符的值，转int输出即可

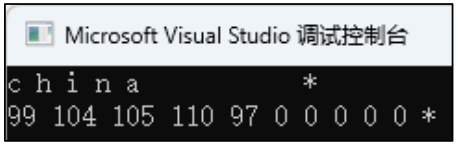
```
#include <iostream>
using namespace std;
int main()
{
    int i;
    char a[10] = {'c','h','i','n','a'};
    for (i=0; i<10; i++)
        cout << a[i] << ' ';//原始
    cout << '*' << endl;

    for (i=0; i<10; i++)
        cout << int(a[i]) << ' ';//整型
    cout << '*' << endl;
    return 0;
}
```

例：初始化数组的前5个元素，再循环输出全部元素

情况1：
新版控制台/新宋体28

情况2：
旧版控制台/新宋体28

情况3：
旧版控制台/新宋体16



# §5. 数组

5.5. 字符数组

5.5.5. 字符数组的输入与输出

5.5.5.1. 字符的输出

5.5.5.2. 字符数组的输出

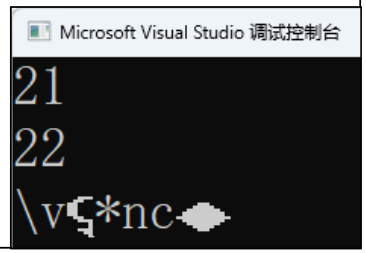
★ 以字符串方式输出含\0的一维字符数组

(仅可显示字符和\0)

```
#include <iostream>
using namespace std;
int main()
{
    char a[10] = {'c','h','i','n','a'}; //5个\0
    char b[]   = "hello";               //大小为6
    char c[10] = "同济\0大学";          //含显式\0
    int  d[10] = {'c','h','i','n','a'}; //int数组
    cout << a << '*' << endl;
    cout << b << '*' << endl;
    cout << c << '*' << endl;
    cout << d << '*' << endl;
}
```

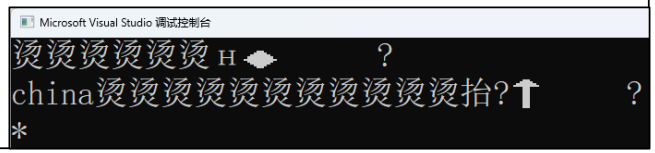
第2章 PPT作业
(含其它不可显示字符)

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

int main()
{
    char s[] = "\bvt\\tnc\4921\x3fr\2a\'\r\\v\a\f";

    cout << strlen(s) << endl;
    cout << sizeof(s) << endl;
    cout << s << '*' << endl;

    return 0;
}
```




★ 以字符串方式输出不含\0的一维字符数组

```
#include <iostream>
using namespace std;
char c[5]; //全部变量不初始化，初值为0（\0）
int main()
{
    char a[5]; //自动变量不初始化，初值随机
    char b[5] = {'c','h','i','n','a'}; //无\0

    cout << a << '*' << endl;
    cout << b << '*' << endl;
    cout << c << '*' << endl;
    return 0;
}
```




结论：

- 输出跟数组名，其他类型(int/double等)是输出数组的起始地址，而字符数组(char/unsigned char)是表示从数组的起始地址开始依次输出各字符的值，到\0为止（简称：字符串方式输出）
- 字符串方式输出下\0表示结束，\0自身不输出
- 字符串方式输出不含\0的数组，则会持续越界输出直到碰见\0为止（错误的具体表现为正常/乱码等多种形式）
- 如果字符数组中包含多个显式/隐式\0，则输出到第一个\0即停止
- 对于含不可显示字符的字符串，直接输出方式不可信，仍建议用strlen/sizeof等方式进行研究

# §5.数组

5.5.字符数组

5.5.5.字符数组的输入与输出

5.5.5.1.字符的输出

★ 第2章补充视频 - 字符与字符串的输出（自行回顾）

5.5.5.2.字符数组的输出

★ 字符数组中每个元素的单独输出

★ 以字符串方式输出含\0的一维字符数组

★ 以字符串方式输出不含\0的一维字符数组

5.5.5.3.字符数组的输入

★ 以单个字符方式输入到字符数组的某个元素中

★ 以字符串方式输入到整个一维字符数组中

通过完成相应的PPT作业来理解字符数组的输入与输出并体会其中的细节差异

# §5.数组

5.6.常用的字符串处理函数

5.6.1.C方式的字符串处理函数

★ 操作对象

一维字符数组表示的字符串

★ 对应头文件

- 使用前加 #include <string.h> //C方式
  #include <cstring> //C++方式
  - ◆ VS下可以不加这个头文件
- VS认为部分函数不安全，使用前需要加 #define _CRT_SECURE_NO_WARNINGS

★ 常用字符串处理函数

① strlen (const char s[]);
② strcat (char dst[], const char src[]);
③ strncat(char dst[], const char src[], const unsigned int len);
④ strcpy (char dst[], const char src[]);
⑤ strncpy(char dst[], const char src[], const unsigned int len);
⑥ strcmp (const char s1[], const char s2[]);
⑦ strncmp(const char s1[], const char s2[], const unsigned int len);

课上只讲了其中4个，
通过作业进一步理解

- 更多的字符串处理函数通过作业完成并理解
- 教材/参考资料/其它老师的课件中，很多形式是 const char *s，暂时忽略，待学习指针后再进一步理解
- 先不要考虑这些函数的返回值，待学习指针后再进一步理解

# §5.数组

5.6.常用的字符串处理函数

5.6.1.C方式的字符串处理函数

★ 常用字符串处理函数

① 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'

```
//例：字符数组与字符串长度
//读操作，不需要加_CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    char str1[]="Hello";
    cout << sizeof(str1) << endl;
    cout << strlen(str1) << endl;

    char str2[]="china\0Hello\0\0";
    cout << sizeof(str2) << endl;
    cout << strlen(str2) << endl;

    return 0;
}
```

```
6   数组长度
5   字符串长度


14  数组长度
5   字符串长度
```

# §5.数组

5.6.常用的字符串处理函数

5.6.1.C方式的字符串处理函数

★ 常用字符串处理函数

② strcat(char dst[], const char src[])

功　　能：将字符串src连接到字符串dst的尾部

输入参数：存放字符串dst的字符数组dst

　　存放字符串src的字符数组src(只读)

返 回 值：改变后的字符数组dst

注意事项：字符数组dst要有足够的空间(两串总长+1)

```
//例：字符串连接
#define _CRT_SECURE_NO_WARNINGS  //VS需要
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    char str1[30]="Tongji ";  //不能缺省，至少18字节!!!
    char str2[]="University"; //缺省长度为11
    cout << strcat(str1, str2) << endl;
    return 0;
}
```

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

输出为：
Tongji University

```
//例：字符串连接
#define _CRT_SECURE_NO_WARNINGS  //VS需要
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char str1[]="Tongji ";    //缺省长度为8
    char str2[]="University"; //缺省长度为11
    cout << strcat(str1, str2) << endl;

    return 0;
}
```

观察输出是否有错

# §5.数组

5.6.常用的字符串处理函数

5.6.1.C方式的字符串处理函数

★ 常用字符串处理函数

③ strncat(char dst[], const char src[], const unsigned int n)

功　　能：将字符串src的前n个字符连接到字符串dst的尾部

输入参数：存放字符串dst的字符数组dst

存放字符串src的字符数组src(只读)

要复制的长度n(只读，如果n超过src长度，则只连接src个)

返 回 值：改变后的字符数组dst

注意事项：字符数组dst要有足够的空间(原dst长度+n+1)

```
//例：字符串连接前n个字符
#define _CRT_SECURE_NO_WARNINGS  //VS需要
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
   char str1[30]="Tongji ";
   char str2[30]="Tongji ";
   char str3[]="University"; //缺省长度为11

   cout << strncat(str1, str3, 3) << '*' << endl;
   cout << strncat(str2, str3, 300) << '*' << endl;
   return 0;
}
```

输出为：

```
Tongji Uni*
Tongji University*
```

# §5. 数组

5.6. 常用的字符串处理函数

5.6.1. C方式的字符串处理函数

★ 常用字符串处理函数

④ strcpy(char dst[], const char src[])

功　　能：将字符串src复制到字符串dst中,覆盖原dst串

输入参数：存放字符串dst的字符数组dst

　　存放字符串src的字符数组src(只读)

返 回 值：改变后的字符数组dst

注意事项：字符数组dst要有足够的空间(串src长+1)

● 字符串复制时包含'\0'，到'\0'为止

```
//例：字符串拷贝
#define _CRT_SECURE_NO_WARNINGS //VS需要
#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;
}
```

| 复制前a | s | t | u | d | e | n | t | \0 |
|---|---|---|---|---|---|---|---|---|
| 复制后a | h | e | l | l | o | \0 | t | \0 |

a/b数组的缺省长度8/6

复制到\0为止
a[6]以后保持原值

输出为：
hello
104 101 108 108 111 0 116 0

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？

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

# §5. 数组

5.6. 常用的字符串处理函数

5.6.1. C方式的字符串处理函数

★ 常用字符串处理函数

⑤ strncpy(char dst[], const char src[], unsigned int n)

功　　能：将字符串src的前n个复制到字符串dst中,覆盖原dst串

输入参数：存放字符串dst的字符数组dst

存放字符串src的字符数组src(只读)　　继续复制导致出错

要复制的长度n(只读，如果n超过src长度，则只复制src个)

返 回 值：改变后的字符数组dst　　n+1

注意事项：字符数组dst要有足够的空间(min(串src长,n)+1)

● strncpy复制时不包含'\0'，且长度超过时出错，要人为保证n正确

```
//例：字符串拷贝前n个字符
#define _CRT_SECURE_NO_WARNINGS  //VS需要
#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;
}
```

证明了未加\0

输出为：

heudent

104 101 117 100 101 110 116 0

```
//例：字符串拷贝前n个字符
#define _CRT_SECURE_NO_WARNINGS  //VS需要
#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;
}
```

提示：数组名代表数组的首地址
即　：b ⇔ &b[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. 数组

5.6. 常用的字符串处理函数

5.6.1. C方式的字符串处理函数

★ 常用字符串处理函数

⑤ strncpy(char dst[], const char src[], unsigned int n)

功　　能：将字符串src的前n个复制到字符串dst中,覆盖原dst串

输入参数：存放字符串dst的字符数组dst

存放字符串src的字符数组src(只读)　　继续复制导致出错

要复制的长度n(只读，如果n超过src长度，则~~只复制src个~~)

返 回 值：改变后的字符数组dst　　n+1

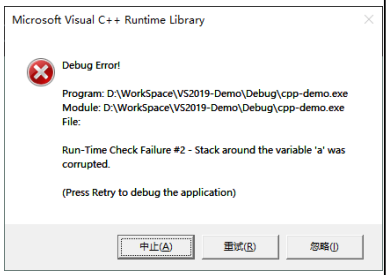
注意事项：字符数组dst要有足够的空间(~~min(串src长,n)+1~~)

深度讨论

● strncpy复制时不包含'\0'，且长度超过时出错，要人为保证n正确

```
#define _CRT_SECURE_NO_WARNINGS  //VS需要
#include <iostream>
#include <cstring>
using namespace std;
int main()
{   int i;
    char a[] = "student", b[] = "hello";
    for (i = 0; i < 12; i++) //12已越界，目的?
        cout << (int)a[i] << ' ';
    cout << endl;
    strncpy(a, b, 200);
    cout << a << endl;
    for (i = 0; i < 12; i++) //12已越界，目的?
        cout << (int)a[i] << ' ';
    cout << endl;
    return 0;
}
```

```
D:\WorkSpace\VS2019-Demo\Debug\cpp-demo.exe
115 116 117 100 101 110 116 0 -52 -52 -52 -52
hello
104 101 108 108 111 0 0 0 0 0 0 0
```




VS输出为：上图+弹窗

```
#define _CRT_SECURE_NO_WARNINGS  //VS需要
#include <iostream>
#include <cstring>
using namespace std;
int main()
{   int i;
    char a[] = "student", b[] = "hello";
    for (i = 0; i < 12; i++) //12已越界
        cout << (int)a[i] << ' ';
    cout << endl;
    strncpy(a, b, 200);
    cout << a << endl;
    for (i = 0; i < 12; i++) //12已越界，目的?
        cout << (int)a[i] << ' ';
    cout << endl;
    return 0;
}
```

```
D:\WorkSpace\VS2019-Demo\cpp-demo\cpp-demo.exe
115 116 117 100 101 110 116 0 8 0 0 0
hello
104 101 108 108 111 0 0 0 8 0 0 0

Process exited after 0.3279 seconds with return value 3221225477
请按任意键继续. . .
```

```
D:\WorkSpace\VS2019-Demo\cpp-demo\cpp-demo.exe
115 116 117 100 101 110 116 0 8 0 0 0 32 -68 64 0 -32 -2 120 0
hello
104 101 108 108 111 0 0 0 8 0 0 0 0 0 0 0 0 0 0 0

Process exited after 0.3316 seconds with return value 3221225477
请按任意键继续. . .
```

Dev输出为：上图

正确吗？怎么体现错？

12换20或更大数试试

# §5. 数组

5.6. 常用的字符串处理函数

5.6.1. C方式的字符串处理函数

★ 常用字符串处理函数

⑥ strcmp(const char s1[], const char s2[])

功　　能：比较字符串s1和字符串s2的大小

输入参数：存放字符串s1的字符数组s1(只读)

　　存放字符串s2的字符数组s2(只读)

返 回 值：整型值(0:相等　>0:串1大　<0:串1小)

另一种输出形式：
串1 < 串2

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

```
#include <iostream>
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;

    return 0;
} //仅比较无赋值，因此不需加_CRT_SECURE_NO_WARNINGS
```

| | |
|---|---|
| 1 | 串1>串2 |
| -1 | < |
| 1 | > |
| 0 | == |
| 0 | == |

# §5.数组

5.6.常用的字符串处理函数

5.6.1.C方式的字符串处理函数

★ 常用字符串处理函数

⑥ strcmp(const char s1[], const char s2[])

- 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");   VS/Dev编译器返回1
                            某些编译器可能返回3
```

# §5. 数组

5.6. 常用的字符串处理函数

5.6.1. C方式的字符串处理函数

★ 常用字符串处理函数

⑥ strcmp(const char s1[], const char s2[])

● 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)

输出：？

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

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

输出：？

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

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

输出：？

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

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

输出：？

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

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

问题：

1、四个例子输出分别是什么？为什么语法不错？< 实际比较的是什么？（基本知识，需要弄懂）

2、局部和全局的定义，结果相反，说明局部变量和全局变量的分配分别是从高地址开始还是低地址开始？（深度讨论，不懂就算了）

# §5.数组

5.6.常用的字符串处理函数

5.6.1.C方式的字符串处理函数

★ 常用字符串处理函数

⑦ strncmp(const char s1[], const char s2[], const unsigned int n)

功　　能：比较字符串s1和字符串s2的前n个字符的大小

输入参数：存放字符串s1的字符数组s1(只读)

存放字符串s2的字符数组s2(只读)

要比较的长度n(只读，n/s1长/s2长三者关系有影响吗?)

返 回 值：整型值(0:相等　>0:串1大　<0:串1小)

● n大于短串长度(短串长度+1)时，一定会结束

```
//例：字符串比较前n个字符，不需加_CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char str1[] = "abcd",  str2[]  = "abcde";
    cout << strncmp(str1, str2, 3)   << endl; //1~4均可
    cout << strncmp(str1, str2, 5)   << endl;
    cout << strncmp(str1, str2, 100) << endl;
    return 0;
}
```

```
0
-1
-1
```

# §5.数组

5.6.常用的字符串处理函数

5.6.2.C方式的字符处理函数

① isdigit(char ch) //判断是否数字(0-9)
② isalpha(char ch) //判断是否字母(A~Z，a-z)
③ isalnum(char ch) //判断是否字母或数字(A~Z，a-z，0-9)
④ isxdigit(char ch) //判断是否16进制数字(0-9，A-F，a-f)
⑤ isspace(char ch) //判断是否空格(含回车/换行/换页/tab/竖向tab)
⑥ islower(char ch) //判断是否小写字母(a-z)
⑦ isupper(char ch) //判断是否大写字母(A-Z)
⑧ tolower(char ch) //大写转小写，其余原值返回
⑨ toupper(char ch) //小写转大写，其余原值返回

这些判断函数满足条件返回非零，不满足返回0
具体的返回值涉及到位运算概念，暂时忽略(荣誉)

★ 需要包含头文件<ctype.h>或<cctype>

```
//实现也相对比较简单，不再深入讨论
int isdigit(int ch)
{
    return (ch>='0' && ch<='9') ? 1 : 0;
}

int tolower(int ch)
{
    return (ch>='A'&&ch<='Z') ? ch+32 : ch;
}
```

# §5.数组

5.6.常用的字符串处理函数

5.6.3.C方式的字符串与格式化输入输出

5.6.3.1.将格式化数据输出到字符串中

★ 标准输出函数

int printf("格式串", 输出表列);

★ 向字符串输出函数

int sprintf(字符数组, "格式串", 输出表列);

- 返回值是输出字符的个数(同printf)
- 字符数组要有足够空间容纳输出的数据(否则越界错)
- 格式串同printf
- VS下需加 #define _CRT_SECURE_NO_WARNINGS

```
//例：sprintf的基本使用
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;

int main()
{
    char str[80];
    int k=123, ret;
    double pi=3.1415925;

    ret = sprintf(str, "k=%-4d*pi=%.2f#", k, pi);
    printf("ret : %d\n", ret);
    printf("str : %s\n", str);

    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
ret : 15
str : k=123 *pi=3.14#
```

# §5. 数组

5.6. 常用的字符串处理函数

5.6.3. C方式的字符串与格式化输入输出

5.6.3.2. 从字符串中输入格式化数据

★ 标准输入函数

int scanf("格式串", 输入地址表列);

★ 从字符串中输入函数

int sscanf(字符数组，"格式串", 输入地址表列);

- 返回值是正确读入的输入数据的个数(同scanf)
- 格式串同scanf
- VS下需加 #define _CRT_SECURE_NO_WARNINGS

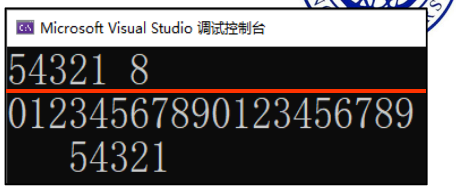
```
//例：sscanf的基本使用
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;

int main()
{
    char str[80] = "Hello 123 11.2", s[10];
    int  i, ret;
    double d;

    ret = sscanf(str, "%s %d %lf", s, &i, &d);
    printf("ret : %d\n", ret);
    printf("s=%s i=%d d=%f\n", s, i, d);

    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
ret : 3
s=Hello i=123 d=11.200000
```

# §5. 数组

5.6. 常用的字符串处理函数

5.6.3. C方式的字符串与格式化输入输出




例：输入整型x∈[1..99999]及宽度w∈[6..10]，以右对齐方式输出x

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

int main()
{
    int x, w;
    cin >> x >> w; //不考虑输入错误
    cout << "01234567890123456789" << endl; //标尺
    cout <<setw(w) << x << endl;

    return 0;
}
```

C++方式

```cpp
//次差方法：%后的宽度要求常量，因此用case做变量转常量
switch(w) {
    case 6:
        printf("%6d", x);
        break;
    case 7:
        printf("%7d", x);
        break;
    ...
}
```

C方式方法2：次差

```cpp
//最差方法：不讲任何技巧的无脑分类讨论
if (w==6) {
    if (x < 10) //1位数
        printf("     %d", x); //5空格
    else if (x < 100) //2位数
        printf("    %d", x);  //4空格
    else if //3位数
        ...
    }
else if (w==7) {
   if (x < 10) //1位数
        printf("      %d", x); //6空格
    else if (x < 100) //2位数
        printf("     %d", x);  //5空格
    else if //3位数
        ...
    }
... //8、9、10
```

C方式方法1：最差

```cpp
//最佳方法：借助sprintf做变量转常量串
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main()
{
    int x, w;
    scanf("%d %d", &x, &w); //不考虑输入错误
    printf("01234567890123456789\n"); //标尺

    char fmt[16];
    sprintf(fmt, "%%%dd\n", w);//变量w放入fmt
    printf(fmt, x);//fmt中的宽度为常量
    return 0;
}
```

C方式方法3

"%%%dd\n", w

%% => 字符%
%d => 8/10 (w)
d => 字符d
=> "%8d"
"%10d"

# §5.数组

5.6.常用的字符串处理函数

5.6.4.用C++方式的cin/cout成员函数处理字符串

★ cin.get(); //多种形式

★ cin.getline();

★ cin.eof()

★ cin.peek()

★ cin.putback()

★ cin.ignore()

★ cout.put();

更多的细节通过完成相应的PPT作业来理解并体会其中的差异

# §5.数组

5.7.C++处理字符串的方法 - 字符串类与字符串变量

5.7.1.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(strcpy/strcat/...)

5.7.2.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

# §5. 数组

## 5.7. C++处理字符串的方法 - 字符串类与字符串变量

### 5.7.3. string类简单变量的定义和使用（与C方式对比）

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

# §5. 数组

## 5.7. C++处理字符串的方法 - 字符串类与字符串变量

### 5.7.3. string类简单变量的定义和使用（与C方式对比）

```
//观察string的所占空间及存放的字符串长度的关系
#include <iostream>
using namespace std;
int main()
{   string s1 = "abc";
    string s2 = "abcdefghijklmnopqrstuvwxyz0123456789";
    cout << sizeof(s1) << ' ' << s1.length() << endl;
    cout << sizeof(s2) << ' ' << s2.length() << endl;
    for (int i = 0; i < 10000; i++)
        s1 += s2;
    cout << sizeof(s1) << ' ' << s1.length() << endl;
    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
28 3
28 36
28 360003
```

现象：大小28字节的变量存储了超过28字节的内容

★ string变量不是原生的基本数据类型，是C++封装的一种复杂数据类型

★ string变量存放的字符串的长度会自动调整，不是固定大小

★ string变量的sizeof大小与存放串的长度无关，涉及到一种新的内存存储和分配方式(动态内存的申请与释放-后续课程)

★ string变量能用+、=、>等运算符来实现连接、赋值、比较等运算，具体的实现原理及实现方法待后续内容(运算符的重载-后续课程)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

★ 除特定题目外，string类本学期都不准使用

# §5.数组

5.7.C++处理字符串的方法 - 字符串类与字符串变量

5.7.4.string类数组的定义和使用（与C方式对比）

★ 形式

```
string 数组名[正整型常量表达式]; //一维
string 数组名[正整型常量表达式1][正整型常量表达式2]; //二维
string name[5];
string course[3][4];
```

★ 含义

数组有若干元素，每个元素是一个任意长度的string类变量

★ 定义时赋初值

```
string name[5]={"Zhang", "Li", "Wang", "Lin", "Zhao"};
string course[3][4]={ {"C语言", "C++", "Pyhton", "Java"},
                      {"高数", "数分, "高代", "线代"},
                      {"足球", "篮球", "排球", "游泳"}
                    };
```

- 对一维数组，字符串的总数不超过数组大小
- 对二维数组，内括号内字符串个数不超过列，内括号总数不超过行
- 字符串的长度不受限制

# §5. 数组

5.7. C++处理字符串的方法 - 字符串类与字符串变量

5.7.4. string类数组的定义和使用（与C方式对比）

★ 与二维字符数组的内存表示比较(定义时赋初值)

```
char str[3][30]={"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

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

| | | | | |
|---|---|---|---|---|
| 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

★ string类一维数组与二维字符数组的比较(输入/输出)

```
char str[3][30];
string name[3];
int i;
for (i=0; i<3; i++) {
    cin >> str[i]; //二维数组带单下标，输入长度不超过29
    cin >> name[i];//一维数组带下标，输入长度不限
    cout << str[i] << '-' << name[i] << endl;
    }
```

```
Microsoft Visual Studio 调试控制台
Zhang zhang Li li Wang wang
Zhang-zhang
Li-li
Wang-wang
```

★ string类一维数组与二维字符数组的比较(通过strcpy/=进行赋值)

```
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";
```

专用函数strcpy 且串长不超过29

赋值运算符 长度不限