



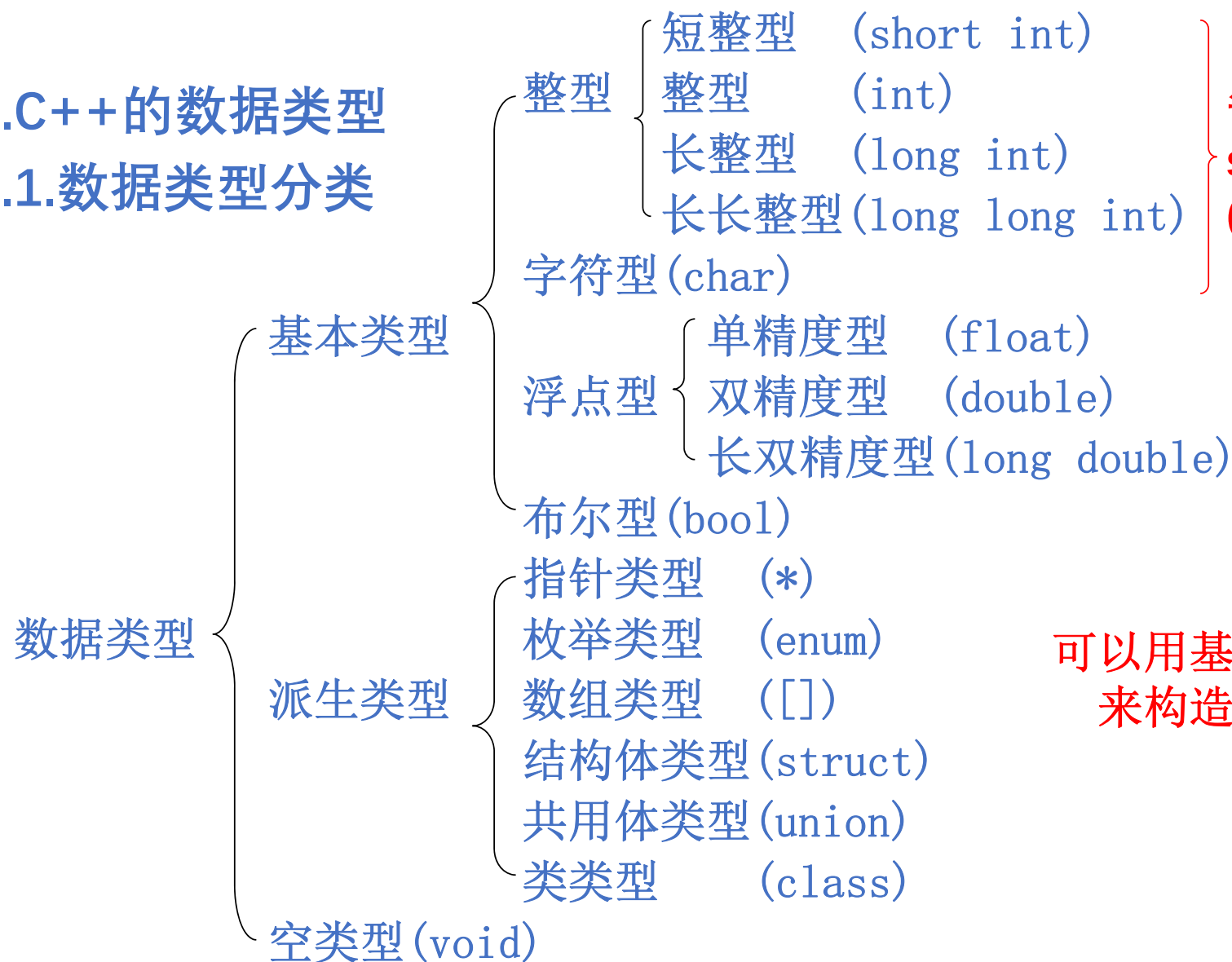
# 5.1 一维数组的定义及使用

李洁

nijanice@163.com

## 2.5.C++的数据类型

### 2.5.1.数据类型分类



每种int都有  
signed与unsigned  
(包括char型)

可以用基本数据类型  
来构造派生类型





# 目录

基本概念

一维数组的声明

一维数组的初始化

一维数组的基本使用

基于一维数组的排序算法



## 5.1 一维数组的定义及使用

### 5.1.1 基本概念

什么是数组？

数组：一种数据格式，存储以**组**的形式出现的多个同**一种数据类型的值**。

学生名单(**数组**)：张三(**元素1**)，李四(**元素2**)，...

每月销售额 (**数组**)：19万元(**元素1**)，23万元(**元素2**)，...

学生成绩(**数组**)：90分(**元素1**)，40分(**元素2**)，...



## 5.1 一维数组的定义及使用

### 5.1.1 基本概念

例：读入100个学生的成绩，求平均成绩并统计高于平均分的人数。

使用之前所学知识实现上例



```
int main()    //用以前所学知识实现:
{
...
...
...
    double s1, s2, s3, ... //需要定义100个变量?

...
...
...

return 0;
}
```



```
int main()      //用以前所学知识实现:
{
...
...
...
    double s1, s2, s3, ... //需要定义100个变量?

...
...
...

return 0;
}
```

辨析:

★使用非数组的普通变量来存储输入的学生成绩时, 需要使用多个变量来存储输入数据, 此时无法使用循环进行统一的输入、输出和计算

```
int main()    //用以前所学知识实现:
{
    int k = 0, i;
    double s, ave, sum = 0;
    for (i = 0; i < 100; i++)
    {
        cin >> s;
        sum = sum + s;
    }
    ave = sum / 100;
    for (i = 0; i < 100; i++)
    {
        cin >> s;    //需要输入两遍元素
        if (s > ave) k++;
    }
    cout << k;
    return 0;
}
```







```
int main()    //用以前所学知识实现:
{
    int k = 0, i;
    double s, ave, sum = 0;
    for (i = 0; i < 100; i++)
    {
        cin >> s;
        sum = sum + s;
    }
    ave = sum / 100;
    for (i = 0; i < 100; i++)
    {
        cin >> s;    //需要输入
        if (s > ave) k++;
    }
    cout << k;
    return 0;
}
```

辨析:

★使用一个s来存储当前输入的学生成绩时，成绩无法保存，用过就被下一个覆盖，再次使用时需要重复读入数据，程序比较混乱



```
int main()//用数组实现:
{
    int k = 0, i;
    double s[100], ave, sum = 0;
    for (i = 0;i < 100;i++)
    {
        cin >> s[i];    //只需要输入一遍元素
        sum = sum + s[i];
    }
    ave = sum / 100;
    for (i = 0;i < 100;i++)
        if (s[i] > ave)
            k++;
    cout << k;
}
```



```
int main()//用数组实现:
{
    int k = 0, i;
    double s[100], ave, sum = 0;
    for (i = 0;i < 100;i++)
    {
        cin >> s[i];    //只需要输入一遍元素
        sum = sum + s[i];
    }
    ave = sum / 100;
    for (i = 0;i < 100;i++)
        if (s[i] > ave)
            k++;
    cout << k;
}
```

辨析:

★使用数组来存储输入的学生成绩时，程序清晰易读，且使用到学生成绩信息时，**直接调用数组**即可



## 5.1 一维数组的定义及使用

### 5.1.1 基本概念

为什么要使用数组：

对于拥有**同样特质**的变量，将他们看作**一个组**并统一存储和操作，实现了将一组元素（一片连续的存储空间）抽象为**一个变量**（数组变量），使得程序可以对数组内的变量统一操作，提高了程序的**效率**。



## 5.1 一维数组的定义及使用

### 5.1.2 一维数组声明

声明/定义格式

typeName          arrayName[arraySize]    ;

数组元素类型          数组名    数组长度



## 5.1 一维数组的定义及使用

### 5.1.2 一维数组声明

#### 声明/定义格式

typeName      arrayName[arraySize]    ;

数组元素类型      数组名    数组长度

- ★ 一个数组可以含有若干个元素，这些元素的**数据类型必须相同**。
- ★ 数组由一组相同类型的元素(量)构成，是一种新的数据类型，称为**复合数据类型**，其中复合指该数据类型是由其他数据类型来创建的。
- ★ 一个数组能存储的最大元素个数称为数组的**长度** arraySize，其值为**大于等于1的整数**，其取值上限取决于编程环境的内存限制。



## 5.1 一维数组的定义及使用

### 5.1.2 一维数组声明

例:

```
int studentNumber[20] ; // 大小为20的int型数组，存储学生的学号  
double grade[50] ; // 大小为50的double型数组，存储学生的成绩  
char character[26] ; //大小为26的char型数组，存储26个英文字符
```

在数组声明时，字段arraySize应是取值固定的常数，且数组的长度在固定后不可改变（使用有关程序内存的new运算符可以避开这个限制，详见章节6：指针）。

//一维数组声明示例:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()//用数组实现:
```

```
{
```

```
    int s = 10;
```

```
    int a[s];
```

```
}
```







//一维数组声明示例:

```
#include <iostream>
```

```
using namespace std;
```

```
int main() //用数组实现:
```

```
{
```

```
    int s = 10;
```

```
    int a[s]; // 数组的长度应该是整型常量表达式，而不是变量
```

```
}
```

辨析:

★在VS中，不允许使用变量定义数组的大小;

	代码	说明
▷ abc	E0028	表达式必须含有常量值
✖	C2131	表达式的计算结果不是常数



//一维数组声明示例:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()//用数组实现:
```

```
{
```

```
    int s = 10;
```

```
    int a[s]; // 数组的长度应该是整型常量表达式，而不是变量
```

```
}
```

辨析:

★在DevC++中，允许使用变量定义数组的大小

//一维数组声明示例:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()//用数组实现:
```

```
{
```

```
    int s = 10;
```

```
    int a[s] = {0};
```

```
    cout << sizeof(a)/4 <<endl;
```

```
    s = 15;
```

```
    cout << sizeof(a)/4 <<endl;
```

```
}
```





//一维数组声明示例:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()//用数组实现:
```

```
{
```

```
    int s = 10;
```

```
    int a[s] = {0}; // 数组的长度应该是整型常量，而不是变量
```

```
    cout << sizeof(a)/4 <<endl;
```

```
    s = 15;
```

```
    cout << sizeof(a)/4 <<endl; //数组a的长度仍为10，没有发生改变
```

```
}
```

```
10
10
```

辨析:

★在DevC++中，允许使用变量定义数组的大小，但一旦数组定义后，再改变变量大小，数组大小是不变的

//一维数组声明错误示例:

```
#include <iostream>
using namespace std;
int main()//用数组实现:
{
    float  b[3.4];
}
```





//一维数组声明错误示例:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()//用数组实现:
```

```
{
```



```
    float  b[3.4]; // 错误, 数组的长度应该是整型常量, 而不是浮点型
```

```
}
```

辨析:

★数组的长度应该是整型常量, 而不是浮点型

;

	代码	说明
	E2373	此常量表达式的类型为 "double", 而所需类型为 "unsigned int"
	C2058	常量表达式不是整型

//一维数组声明正确示例:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()//用数组实现:
```

```
{
```

```
    const int s = 10;
```

```
    int a[s];
```

```
}
```



//一维数组声明正确示例:

```
#include <iostream>
```

```
using namespace std;
```

```
int main()//用数组实现:
```

```
{
```

```
    const int s = 10; // 正确, 数组的长度使用的是整型常量表达式
```

```
    int a[s];
```

```
}
```







## 5.1 一维数组的定义及使用

### 5.1.3 一维数组初始化

一个数组在被声明后，通常需要先进行初始化的过程来**赋予初值**，以防止因数组中存储着含有我们不知道的值的元素而导致的BUG

```
#include <iostream>
using namespace std;
int main()
{
    int grade[5];
    cout << grade[2];
    return 0;
}
```



## 5.1 一维数组的定义及使用

### 5.1.3 一维数组初始化

一个数组在被声明后，通常需要先进行初始化的过程来**赋予初值**，以防止因数组中存储着含有我们不知道的值的元素而导致的BUG

```
#include <iostream>
using namespace std;
int main()
{
    int grade[5];
    cout << grade[2];
    return 0;
}
```

Microsoft Visual Studio 调试控制台  
-858993460

辨析：

★此时数组在声明后没有进行初始化，数组对应的存储空间存放着程序员不知道的未知值，正确的做法是进行下页所示的数组初始化后，再使用数据元素



## 5.1 一维数组的定义及使用

### 5.1.3 一维数组初始化

一个数组在被声明后，该数组在程序的**存储空间中的起始地址**将记录在**数组名**中，即此时的数据名为一个指针（具体内容见章节六），下段程序的输出为一个**16进制数**，表示数组的起始地址

```
#include <iostream>
using namespace std;
int main()
{
    int grade[5];
    cout << grade;
    return 0;}
```

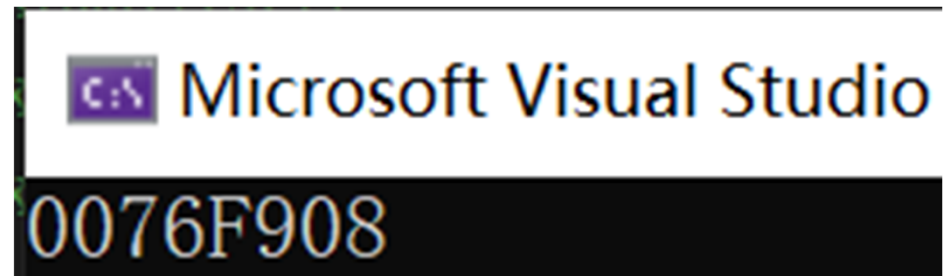


## 5.1 一维数组的定义及使用

### 5.1.3 一维数组初始化

一个数组在被声明后，该数组在程序的**存储空间中的起始地址**将记录在**数组名**中，即此时的数据名为一个指针（具体内容见章节六），下段程序的输出为一个**16进制数**，表示数组的起始地址

```
#include <iostream>
using namespace std;
int main()
{
    int grade[5];
    cout << grade;
    return 0;}
```





## 5.1 一维数组的定义及使用

### 5.1.3 一维数组初始化

初始化方法（全部初始化）：

- `int grade[5] = {1, 2, 3, 4, 5};` //所有元素依次初始化
- `int grade[5] {1, 2, 3, 4, 5};` //数组初始化可以省略 = 号
- `int grade[] = {1, 2, 3, 4, 5};` // 编译器自动设定数组的元素个数为5



## 5.1 一维数组的定义及使用

### 5.1.3 一维数组初始化

初始化方法（部分初始化）：

- `int grade[5] = {};`      //所有元素初始化为0
- `int grade[5] = {0};`      //所有元素初始化为0
- `int grade[5] = {1};`  
    //数组第一个元素初始化为1，剩下元素自动初始化为0
- `int grade[5] = {1, 2};`  
    //数组前两个元素初始化为1和2，剩下元素自动初始化为0



## 5.1 一维数组的定义及使用

### 5.1.3 一维数组初始化

错误示例（请指出错误）：

```
int grade[5] = {1, 2, 3, 4, 5};
```

```
int grade1[5];
```

```
grade1 = grade;
```

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

```
int grade[5] = {0, 1.1, 2, 3, '4'};
```

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

```
a={1, 3, 5, 7, 9};
```



## 5.1 一维数组的定义及使用

### 5.1.3 一维数组初始化

错误示例（请指出错误）：

```
int grade[5] = {1, 2, 3, 4, 5};
```

```
int grade1[5];
```

```
grade1 = grade; // 不能用该方式将一个数组赋值给另一个数组
```

```
int grade[5] = {1, 2, 3, 4, 5, 6}; //数组长度超出定义
```

```
int grade[5] = {0, 1.1, 2, 3, '4'}; //数组初始化禁止使用缩窄转换
```

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

```
a={1, 3, 5, 7, 9}; //数组名是个地址常量，不能被赋值
```





## 5.1 一维数组的定义及使用

### 5.1.3 一维数组初始化

错误示例（请指出错误）：

```
int grade[5];  
grade[5] = {1, 2, 3, 4, 5};
```

```
char slifs[4] = { 'h', 'i', 1122011, '\0' };
```

```
long plifs[3] = { 25, 92, 3.0 };
```



## 5.1 一维数组的定义及使用

### 5.1.3 一维数组初始化

错误示例（请指出错误）：

```
int grade[5];
```

`grade[5] = {1, 2, 3, 4, 5};` //一维数组初始化只能在数组声明定义时使用，不能在程序的其他位置使用；不能用花括号为一个元素赋多个值

`char slifs[4] = { 'h' , 'i' , 1122011, '\0' };` //数组初始化禁止使用缩窄转换

`long plifs[3] = { 25, 92, 3.0 };` //数组初始化禁止使用缩窄转换



## 5.1 一维数组的定义及使用

### 5.1.3 一维数组初始化

★一维数组的声明和初始化过程，可以分开成两条语句（这种情况称为数组**初始值赋值**而不是初始化），也可以合在一起写成一条语句。

★一维数组的初始化语句**并不是必须要写**的程序语句，但使用未初始化的数组可能会导致程序出现错误，合理的初始化一维数组是**编程的好习惯**之一。



## 5.1 一维数组的定义及使用

### 5.1.3 一维数组初始化

★一维数组的声明和初始化使用一条语句(推荐):

```
int grade[5] = {1, 2, 3, 4, 5};
```

★一维数组的声明和初始值赋值使用分开的两条语句:

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

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

```
    grade[i] = i + 1;
```



## 5.1 一维数组的定义及使用

### 5.1.4 一维数组的基本使用

#### ★ 数组的存储

数组在程序运行时的存储空间中，存放在一组**连续的地址单元**内。  
数组名是常量，表示数组在内存中的首地址。

如 `int s[5];`



S是数组名，也表示数组在内存中的首地址



## 5.1 一维数组的定义及使用

### 5.1.4 一维数组的基本使用

如何单独访问数组元素？

#### ★数组的下标(索引)

数组对于其所含元素进行编号，便于程序对数组中的元素进行访问。在C/C++中，数组的元素编号从0开始，并依次递增，数组中元素的编号称为下标或索引。

如 `int s[5];`

<code>s[0]</code>	<code>s[1]</code>	<code>s[2]</code>	<code>s[3]</code>	<code>s[4]</code>
-------------------	-------------------	-------------------	-------------------	-------------------

注意：

★数组的大小为`arraySize`时，其合理的下标范围为0到`arraySize-1`；

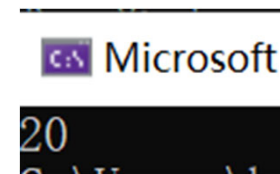


## 5.1 一维数组的定义及使用

### 5.1.4 一维数组的基本使用

★ 数组长度的计算使用运算符 `sizeof`，`sizeof` 在输入数组名做为参数时，会返回整个数组的字节数

```
int grade[5] = { 1, 2, 3, 4, 5 };  
cout << sizeof(grade);
```



数组的长度即为：

整个数组的字节数 / 数组中元素的数据类型所占的字节数



## 5.1 一维数组的定义及使用

### 5.1.4 一维数组的基本使用

★对于一维数组中的元素，使用**数组下标**引用它们，相当于使用一个**变量**。

★对于一维数组中的元素，还可以使用指针引用它们，具体内容见章节六。

★**数组的元素做为变量被函数使用**，此时参数调用的方式为**按值传递**



```
int main() //一维数组声明、初始化和赋值的综合示例(P71, 例4.1)
{
    int yams[3] = { 7, 8, 6 }; //每袋马铃薯的个数
    int yamcost[3] = { 20, 30, 5 }; //每袋马铃薯的价格
    cout << "Total yams = ";
    cout << yams[0] + yams[1] + yams[2] << endl; //三袋马铃薯总数
    cout << "The package with " << yams[1] << " yams costs ";
    cout << yamcost[1] << " cents per yam\n"; //第二袋马铃薯的单价
    int total = yams[0] * yamcost[0] + yams[1] * yamcost[1]
+ yams[2] * yamcost[2]; //三袋马铃薯的总价
    cout << "The total yam expense is " << total << " cents\n";
    //马铃薯个数数组的内存空间大小
    cout << "\nSize of yams array = " << sizeof(yams) << " bytes.\n";
    //马铃薯个数数组中一个元素的内存空间大小
    cout << "Size of one element = " << sizeof(yams[0]) << " bytes.\n";
    return 0;
}
```

int main() //一维数组声明、初始化和赋值的综合示例(P71, 例4.1)

```
{  
    int yams[3] = { 7, 8, 6 }; //每袋马铃薯的个数  
    int yamcost[3] = { 20, 30, 5 }; //每袋马铃薯的价格  
    cout << "Total yams = ";  
    cout << yams[0] + yams[1] + yams[2] << endl; //三袋马铃薯总数  
    cout << "The package with " << yams[1] << " yams costs ";  
    cout << yamcost[1] << " cents per yam\n"; //第二袋马铃薯的单价  
    int total = yams[0] * yamcost[0] + yams[1] * yamcost[1]  
+ yams[2] * yamcost[2];  
    cout << "Total yams = 21\n";  
    //马铃薯个数  
    cout << "The package with 8 yams costs 30 cents per yam\n";  
    //马铃薯单价  
    cout << "The total yam expense is 410 cents\n";  
    //马铃薯总数  
    cout << "Size of yams array = 12 bytes.\n";  
    cout << "Size of one element = 4 bytes.\n";  
    return 0;  
}
```

```
#include <iostream>
using namespace std;
int add(int a, int b)
{
    int c; c = a + b;
    return c;
}
int main()
{
    int grade[5] = { 1, 2, 3, 4, 5 };
    cout << grade[2] << "\n";
    int count1, count2;
    count1 = grade[1] + grade[2];
    count2 = add(grade[1], grade[2]);
    cout << count1 << "\n" << count2;
    return 0;
}
```

```
#include <iostream>
using namespace std;
int add(int a, int b)
{
    int c; c = a + b;
    return c;
}
```

```
int main()
{
```

```
    int grade[5] = { 1, 2, 3, 4, 5 };
```

```
    cout << grade[2] << "\n";    //数组的元素做为变量被输出
```

```
    int count1, count2;
```

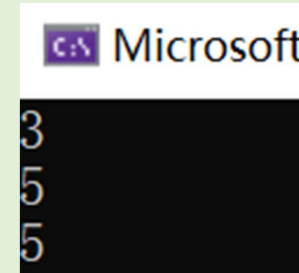
```
    count1 = grade[1] + grade[2];    //数组的元素做为变量被表达式使用
```

```
    count2 = add(grade[1], grade[2]);    //数组的元素做为变量被函数使用
```

```
    cout << count1 << "\n" << count2;
```

```
    return 0;
```

```
}
```



```

#include <iostream>
using namespace std;
int add(int a, int b)
{
    int c; c = a + b;
    return c;
}
int main()
{
    int grade[5] = { 1, 2, 3,
    cout << grade[2] << "\n";
    int count1, count2;
    count1 = grade[1] + grade[
    count2 = add(grade[1], gra
    cout << count1 << "\n" <<
    return 0;
}

```

辨析：

★对于一维数组中的元素，使用数组下标引用它们，相当于使用一个变量。

★数组的元素做为变量被函数使用，此时参数调用的方式为按值传递

用



## 5.1 一维数组的定义及使用

### 5.1.4 一维数组的基本使用

★ **数组遍历**：指对数组中连续的几个元素或是全部元素进行某项操作，是对于数组的常见操作之一



```
int main()//用数组实现:
{
    int k = 0, i;
    double s[100], ave, sum = 0;
    for (i = 0;i < 100;i++)
    {
        cin >> s[i];    // 数组元素的输入
        sum = sum + s[i]; // 数组元素的求和
    }
    ave = sum / 100;
    for (i = 0;i < 100;i++)
        if (s[i] > ave)
            k++;
    cout << k;
}
```



```
int main() //用数组实现:
{
    int k = 0, i;
    double s[100], ave, sum = 0;
    for (i = 0; i < 100; i++)
    {
        cin >> s[i];    // 数组元素的输入
    }
    int max = s[0], imax = 0;    //假设第一个元素值最大
    for (i = 1; i < 100; i++)
        if (s[i] > max)
        {
            max = s[i];    //求最大元素
            imax = i;    //求最大元素下标
        }
    cout << max << "\n" << imax;
}
```





## 5.1 一维数组的定义及使用

### 5.1.4 一维数组的基本使用

★ 对于一维数组这个整体，在使用**数组名**引用数组时，数组名实际表示的是数组在程序的存储空间中**存储位置的起始地址**。因此，对于数组名的引用，**不可以用于赋值或是在表达式中进行计算**。

★ **数组的复制，拆分和合并**：在需要完成数组的“复制”或是“拆分”的逻辑时，而是应该将数组的元素依次复制给另一个数组。



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

int main()
{
    int grade1[5] = {1, 2, 3, 4, 5};
    int grade2[5];
    int i;
    grade2 = grade1;
    for (i = 0; i < 5; i++)
        cout << grade2[i] << "\n";
    return 0;
}
```



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

int main()
{
    int grade1[5] = {1, 2, 3, 4, 5};
    int grade2[5];
    int i;
    grade2 = grade1;
    for (i = 0; i < 5; i++)
        cout << grade2[i] << "\n";
    return 0;
}
```

#### 错误列表

整个解决方案

❌ 错误 2

⚠️ 警告 0

	代码	说明
abc	E0137	表达式必须是可修改的左值
❌	C3863	不可指定数组类型“int [5]”



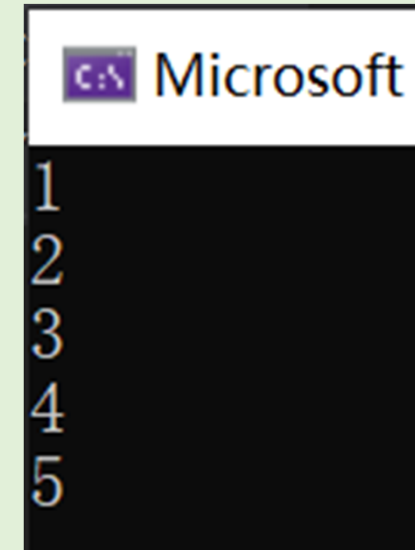
```
#include <iostream>
using namespace std;

int main()
{
    int grade1[5] = {1, 2, 3, 4, 5};
    int grade2[5];
    int i;
    for (i = 0; i < 5; i++)
        grade2[i] = grade1[i];
    for (i = 0; i < 5; i++)
        cout << grade2[i] << "\n";
    return 0;
}
```



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

int main()
{
    int grade1[5] = {1, 2, 3, 4, 5};
    int grade2[5];
    int i;
    for (i = 0; i < 5; i++)
        grade2[i] = grade1[i]; //数组复制
    for (i = 0; i < 5; i++)
        cout << grade2[i] << "\n";
    return 0;
}
```



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

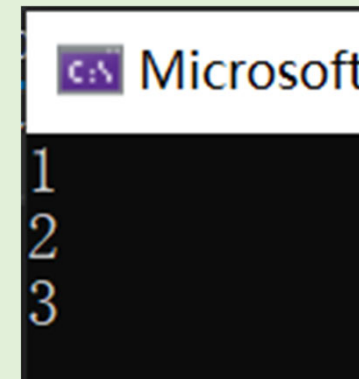
int main()
{
    int grade[5] = { 1, 2, 3, 4, 5 };
    int part_grade[3] = {};
    int i;
    for (i = 0; i < 5; i++)
    {
        if (i < 3)
            part_grade[i] = grade[i];
    }
    for (i = 0; i < 3; i++)
        cout << part_grade[i] << "\n";
    return 0;
}
```





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

int main()
{
    int grade[5] = { 1, 2, 3, 4, 5 };
    int part_grade[3] = {};
    int i;
    for (i = 0; i < 5; i++)
    {
        if (i < 3)
            part_grade[i] = grade[i]; //数组拆分
    }
    for (i = 0; i < 3; i++)
        cout << part_grade[i] << "\n";
    return 0;
}
```



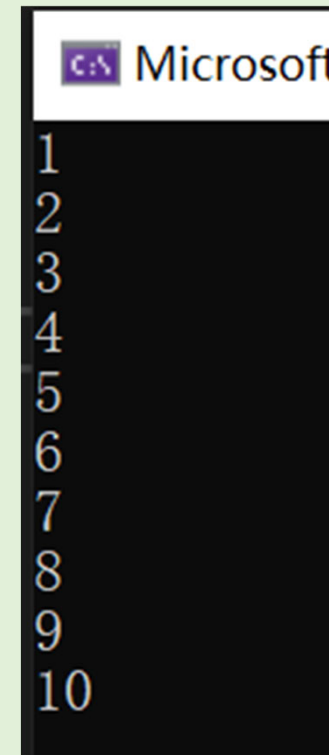
```
#include <iostream>
using namespace std;
int main()
{
    int grade1[5] = {1, 2, 3, 4, 5};
    int grade2[5] = {6, 7, 8, 9, 10};
    int grade[10] = {};
    int i ;
    for(i=0; i<10 ; i++)
    {
        if (i < 5)
            grade[i] = grade1[i];
        else
            grade[i] = grade2[i-5];
    }
    for (i = 0; i < 10; i++)
        cout << grade [i] << "\n";
    return 0;
}
```







```
#include <iostream>
using namespace std;
int main()
{
    int grade1[5] = {1, 2, 3, 4, 5};
    int grade2[5] = {6, 7, 8, 9, 10};
    int grade[10] = {};
    int i ;
    for(i=0; i<10 ; i++) //数组合并
    {
        if (i < 5)
            grade[i] = grade1[i];
        else
            grade[i] = grade2[i-5];
    }
    for (i = 0; i < 10; i++)
        cout << grade [i] << "\n";
    return 0;
}
```





## 5.1 一维数组的定义及使用

### 5.1.4 一维数组的基本使用

★ **数组名作为函数参数**：在使用**数组名**引用数组时，数组名实际表示的是数组在程序的存储空间中**存储位置的起始地址**。对于数组名的引用，常用于函数的参数传递，此时为**按地址传递**

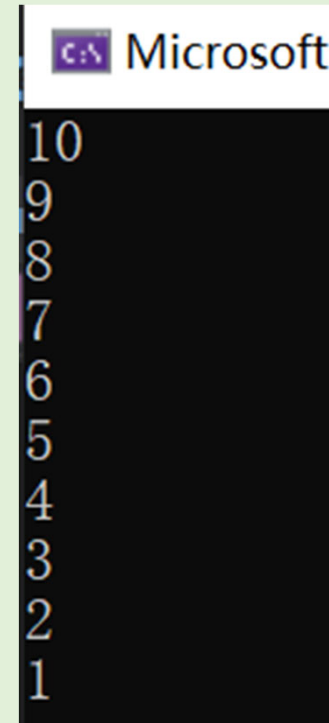


```
#include <iostream>
using namespace std;
void print_grade(int grade[10])
{
    int i;
    for (i = 0; i < 10; i++)
        cout << grade[i] << "\n";
}

int main()
{
    int grade[10] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
    print_grade(grade);    //函数调用
    return 0;
}
```

```
#include <iostream>
using namespace std;
void print_grade(int grade[10])
{
    int i;
    for (i = 0; i < 10; i++)
        cout << grade[i] << "\n";
}

int main()
{
    int grade[10] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
    print_grade(grade);    //函数调用
    return 0;
}
```



```
C:\N Microsoft
10
9
8
7
6
5
4
3
2
1
```





## 4.1 函数的基本使用

### 4.1.2 参数

函数的形式参数:

用于接收传递值的变量，出现在被调用函数的定义中，只能在该函数内部使用。在函数被调用时创建并被分配资源，在函数返回时删除并回收资源。

函数的实际参数:

用于传递值的变量，出现在调用者(不一定是main函数)的语句段中，实参不能在被调用的函数中使用。

在函数调用的过程中，程序通常将实参的值传递给形参。形参和实参的命名不需要相同。



## 4.1 函数的基本使用

### 4.1.2 参数

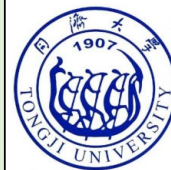
按地址传递: call-by-reference

在调用函数时, 传递给函数的是参数的地址。相当于直接将输入函数的变量放入函数的语句中操作。

此时函数体内对于参数的操作, 会改变函数体外 (调用它的程序) 中的参数变量的值。(教材P258页章节8.2.2将引用用作函数参数)。


数组名做为函数参数: 此时函数按地址传递参数, 对于形式参数的操作会改变实际参数的值, 此时要求形参和实参的类型完全一致

```
void swap_grade(int grade[10]) //因页面缩放问题未一句一行
{
    int temp = grade[0];
    grade[0] = grade[9];
    grade[9] = temp; }
void print_grade1(int grade[10])
{
    int i;
    for (i = 0; i < 10; i++)
        cout << grade[i] << ' ';
    cout << "\n";
}
int main()
{
    int grade[10] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
    print_grade1(grade);
    swap_grade(grade);
    print_grade1(grade);
    return 0; }
```





```
void swap_grade(int grade[10]) //因页面缩放问题未一句一行
{ //形式参数改变
    int temp = grade[0];
    grade[0] = grade[9];
    grade[9] = temp; }
void print_grade1(int grade[10])
{
    int i;
    for (i = 0; i < 10; i++)
        cout << grade[i] << ' ';
    cout << "\n";
}
int main()
{
    int grade[10] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
    print_grade1(grade);
    swap_grade(grade);
    print_grade1(grade); //实际参数改变
    return 0; }
```

 Microsoft Visual Studio 调试控制台

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





## 5.1 一维数组的定义及使用

### 5.1.4 一维数组的基本使用

**数组名作为函数参数：**此时**要求形参和实参完全一致**，而不是像简单变量一样允许在函数参数调用时进行类型转换。**数组名作为函数参数时**，不一样的**形参和实参的类型**会导致程序报错

```
void swap_grade(int grade[10]) //因页面缩放问题未一句一行
{
    int temp = grade[0];
    grade[0] = grade[9];
    grade[9] = temp; }
void print_grade1(int grade[10])
{
    int i;
    for (i = 0; i < 10; i++)
        cout << grade[i] << ' ';
    cout << "\n";
}
int main()
{
    double grade[10] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
    print_grade1(grade);
    swap_grade(grade);
    print_grade1(grade);
    return 0; }
```



```
void swap_grade(int grade[10]) //因页面缩放问题未一句一行
{ //形式参数为int类型数组
    int temp = grade[0];
    grade[0] = grade[9];
    grade[9] = temp; }
void print_grade1(int grade[10])
{ //形式参数为int类型数组
    int i;
    for (i = 0; i < 10; i++)
        cout << grade[i] << ' ';
    cout << "\n";
}
int main()
{ //实际参数为double类型数组
    double grade[10] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
    print_grade1(grade);
    swap_grade(grade);
    print_grade1(grade);
    return 0; }
```





```
void swap_grade(int grade[10]) //因页面缩放问题未一句一行
{ //形式参数为int类型数组
```

#### 错误列表

整个解决方案

❌ 错误 6

⚠️ 警告 0

ℹ️ 6消息 的 0



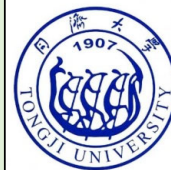
生成 + IntelliSense

	代码	说明
abc	E0167	"double *" 类型的实参与 "int *" 类型的形参不兼容
abc	E0167	"double *" 类型的实参与 "int *" 类型的形参不兼容
abc	E0167	"double *" 类型的实参与 "int *" 类型的形参不兼容
❌	C2664	"int print_grade1(int [])": 无法将参数 1 从 "double [10]" 转换为 "int []"
❌	C2664	"void swap_grade(int [])": 无法将参数 1 从 "double [10]" 转换为 "int []"
❌	C2664	"int print_grade1(int [])": 无法将参数 1 从 "double [10]" 转换为 "int []"

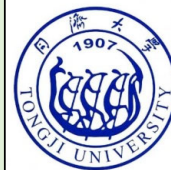
```
int main()
```

```
{ //实际参数为double类型数组
    double grade[10] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
    print_grade1(grade);
    swap_grade(grade);
    print_grade1(grade);
    return 0; }
```

```
void swap_grade(int grade[10]) //因页面缩放问题未一句一行
{
    int temp = grade[0];
    grade[0] = grade[9];
    grade[9] = temp; }
void print_grade1(int grade[10])
{
    int i;
    for (i = 0; i < 10; i++)
        cout << grade[i] << ' ';
    cout << "\n";
}
int main()
{
    int grade[10] = { 10, 9, 8, 7, 6, 5, 4, 3, 2 };
    print_grade1(grade);
    swap_grade(grade);
    print_grade1(grade);
    return 0; }
```



```
void swap_grade(int grade[10]) //因页面缩放问题未一句一行
{ //形式参数数组长度为10
    int temp = grade[0];
    grade[0] = grade[9];
    grade[9] = temp; }
void print_grade1(int grade[10])
{ //形式参数数组长度为10
    int i;
    for (i = 0; i < 10; i++)
        cout << grade[i] << ' ';
    cout << "\n";
}
int main()
{ //实际参数数组长度为9
    int grade[9] = { 10, 9, 8, 7, 6, 5, 4, 3, 2 };
    print_grade1(grade);
    swap_grade(grade);
    print_grade1(grade);
    return 0; }
```





```
void swap_grade(int grade[10]) //因页面缩放问题未一句一行
```

```
{ //形式参数数组长度为10
```

```
    int temp = grade[0];
```

```
    grade[0] = grade[9];
```

```
    grade[9] = temp; }
```

```
void print_grade1(int grade[10])
```

```
{ //形式参数数组长度为10
```

```
    int i;
```

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

```
        cout << grade[i] << " ";
```

```
    cout << "\n";
```

```
}
```

```
int main()
```

```
{ //实际参数数组长度为9
```

```
    int grade[9] = { 10, 9, 8, 7, 6, 5, 4, 3, 2 };
```

```
    print_grade1(grade);
```

```
    swap_grade(grade);
```

```
    print_grade1(grade);
```

```
    return 0; }
```

已引发异常

Run-Time Check Failure #2 - Stack around the variable 'grade' was corrupted.

[复制详细信息](#)

return 0;



## 5.1 一维数组的定义及使用

### 5.1.5 基于一维数组的排序算法

一维数据排序指将元素**数据类型为数值**类型的数组，依据每个元素的数值大小按**升序或降序**将数组元素**重新排列**。

**基础的**数组排序方法有**选择排序，冒泡排序，插入排序**。





## 5.1 一维数组的定义及使用

### 5.1.5 基于一维数组的排序算法

- 对于排序算法过程的可视化展示，参考网站  
<https://visualgo.net/zh/sorting>。



## 5.1 一维数组的定义及使用

### 5.1.5 基于一维数组的排序算法

#### 选择排序：

- (1) 从 $n$ 个数的序列中选出最小的数(递增)，与第1个数交换位置；
- (2) 除第1个数外，其余 $n-1$ 个数再按(1)的方法选出次小的数，与第2个数交换位置；
- (3) 重复流程(1)-(2)  $n-1$ 遍，最后得到排序后的递增序列。

```
void select_sort(int a[]) //因页面缩放问题未一句一行
```

```
{ int i, j, min, temp;
```

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

```
{ min = a[i] //假定最小值
```

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

```
    //找最小值
```

```
if (a[j] < min)
```

```
{
```

```
    //交换最小值
```

```
} }
```

```
int main()
```

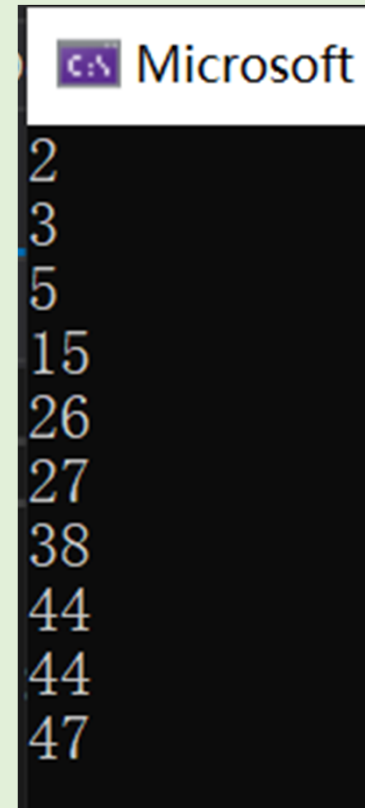
```
{ int grade[10] = { 3, 44, 38, 5, 47, 15, 26, 27, 2, 44 }; int i, n=10;
```

```
select_sort(grade, n);
```

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

```
    //输出排序结果
```

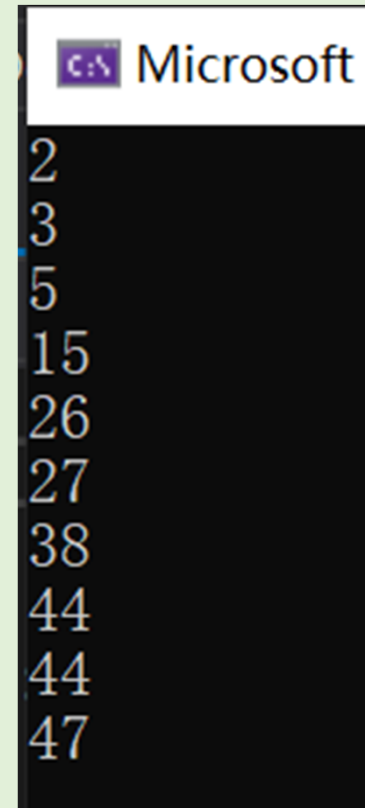
```
return 0; }
```



```
void select_sort(int grade[],int n) //因页面缩放问题未一句一行
```

```
{    int i, j, min, temp;
    for (i = 0;i < n-1;i++)
    {    min = i; //假定最小值
        for (j = i + 1;j < n;j++)
            if (grade[j] < grade[min]) //找最小值
                min = j;
        if (i != min)
        {
            temp = grade[i];
            grade[i] = grade[min]; //交换最小值
            grade[min] = temp;
        } } }
```

```
int main()
{    int grade[10] = { 3, 44, 38, 5, 47, 15, 26, 27, 2, 44 }; int i, n=10;
    select_sort(grade,n);
    for (i = 0; i < n; i++)
        cout << grade[i] << "\n" ; //输出排序结果
    return 0; }
```





## 5.1 一维数组的定义及使用

### 5.1.5 基于一维数组的排序算法

#### 冒泡排序：

(1) 从第一个元素开始，对数组中两两相邻的元素比较，将值较小的元素放在前面，值较大的元素放在后面，这一轮比较完毕后， $n$ 个元素中最大的数存放在 $a[n-1]$ 中；

(2) 然后对 $a[0]$ 到 $a[n-2]$ 的 $n-1$ 个数进行同(1)的操作，次最大数放入 $a[n-2]$ 元素内，完成第二趟排序；依次类推，进行 $n-1$ 趟排序后，最后得到排序后的递增序列。



```
void bubble_sort(int grade[],int n) //因页面缩放问题未一句一行
```

```
{  int i, j, min, temp;
```

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

```
        //两两相邻的元素比较并交换
```

```
}
```

```
int main()
```

```
{
```

```
    int grade[10] = { 3, 44, 38, 5, 47, 15, 26, 27, 2, 44 };
```

```
    int i,n=10;
```

```
    bubble_sort(grade,n);
```

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

```
        cout << grade[i] << "\n";
```

```
    return 0; }
```

Microsoft

2  
3  
5  
15  
26  
27  
38  
44  
44  
47



```
void bubble_sort(int grade[],int n) //因页面缩放问题未一句一行
{
    int i, j, min, temp;
    for (i = 0;i < n;i++)
        for (j = 1;j < n - i;j++) //两两相邻的元素比较并交换
            if (grade[j - 1] > grade[j])
            {
                temp = grade[j - 1];
                grade[j - 1] = grade[j];
                grade[j] = temp;
            }
}

int main()
{
    int grade[10] = { 3, 44, 38, 5, 47, 15, 26, 27, 2, 44 };
    int i,n=10;
    bubble_sort(grade,n);
    for (i = 0; i < 10; i++)
        cout << grade[i] << "\n";
    return 0; }
```

```
Microsoft
2
3
5
15
26
27
38
44
44
47
```



## 5.1 一维数组的定义及使用

### 5.1.5 基于一维数组的排序算法

#### 插入排序:

(1) 从第二个元素 $a[1]$  开始, 其前面的1个元素已经排好序, 此时将这个元素 $a[1]$ 按大小顺序插入到前1个元素中的对应位置。

(2) 然后对 $a[2]$ 到 $a[n-1]$ 的 $n-1$ 个数进行同 (1) 的操作, 即重复过程 (1)  $n-1$ 次, 最后得到排序后的递增序列。





```
void insert_sort(int grade[], int n) //因页面缩放问题未一句一行
```

```
{  int i, j, temp;
```

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

```
        for (j = i; j > 0; j--)
```

//将元素插入到前面已经排好的数中



```
}
```

```
int main()
```

```
{  int grade[10] = { 3, 44, 38, 5, 47, 15, 26, 27, 2, 44 };
```

```
    int i, n=10;
```

```
    insert_sort(grade, n);
```

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

```
        cout << grade[i] << "\n";
```

```
    return 0; }
```

Microsoft

2  
3  
5  
15  
26  
27  
38  
44  
44  
47



```
void insert_sort(int grade[], int n) //因页面缩放问题未一句一行
```

```
{    int i, j, temp;
```

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

```
        for (j = i; j > 0; j--)
```

//将元素插入到前面已经排好的数中

```
            if (grade[j] < grade[j-1])
```

```
            {
```

```
                temp = grade[j];
```

```
                grade[j] = grade[j-1];
```

```
                grade[j-1] = temp;
```

```
            }
```

```
}
```

```
int main()
```

```
{    int grade[10] = { 3, 44, 38, 5, 47, 15, 26, 27, 2, 44 };
```

```
    int i, n=10;
```

```
    insert_sort(grade, n);
```

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

```
        cout << grade[i] << "\n";
```

```
    return 0; }
```

Microsoft

2  
3  
5  
15  
26  
27  
38  
44  
44  
47



思考:

哪个方法好? 怎么衡量?



衡量算法的好坏，包括两个非常重要的指标：运行时间 & 占用空间

可是，如果代码都还没有运行起来，我怎么能预知到代码运行所花的时间呢？



由于运行环境和输入规模的影响，代码的绝对执行时间是无法估计的。但我们却可以预估出代码的基本操作执行次数。



记  $O(f(n))$  为算法  $f(n)$  的渐进时间复杂度，简称时间复杂度。  
即，把时间规模简化成一个基本操作执行次数的数量级，如  $n$ ,  $n^2$  等



## 5.1 一维数组的定义及使用

### 5.1.5 基于一维数组的排序算法

- 基础的数组排序方法有选择排序，冒泡排序，插入排序，这三个排序算法的时间复杂度为 $O(N^2)$ ，即排序算法的效率与数组长度的平方成正比。
- 存在效率更高的排序算法，如归并排序，快速排序，哈希排序等算法。可以证明，确定的排序算法的时间复杂度最低为 $O(N \log N)$ 。（具体内容详见之后的算法课程）



# 目录

基本概念                      数组概念/数组下标

一维数组声明                声明格式/数组长度

一维数组初始化            初始化方法

一维数组的基本使用            下标引用/遍历/做函数参数/拆分合并

基于一维数组的排序算法    选择排序/插入排序/冒泡排序