



西北工业大学  
NORTHWESTERN POLYTECHNICAL UNIVERSITY

---

# 程序设计基础 Programming in C++

U10G13027/U10G13015

---

主讲：魏英，计算机学院

## 第6章 数组

```
#include<iostream>
using namespace std;
int m(int x, int y, int z)
{ int max1,max2,min1,min2;
  max1=(x>y)? x:y; max2=(max1>z)? max1:z;
  min1=(x<y)? x:y; min2=(min1<z)? min1:z;
  return(max2);
  return(min2);
}
int main()
{ int a,b,c,d;
  cin>>a>>b>>c;
  d=m(a,b,c);
  cout<<d<<endl;
  return 0;
}
```

※ 我希望函数m的功能为求出最大值和最小值，程序应该如何改动？



- ▶ 6.1 一维数组的定义和引用
- ▶ 6.2 多维数组的定义和引用
- ▶ 6.3 数组与函数
- ▶ 6.4 字符串
- ▶ \*6.5 C++字符串类
- ▶ 6.6 数组应用程序举例

## 6.1 一维数组的定义和引用

---

- ▶ 数组就是一组**相同**类型数据的集合。
- ▶ 使用数组，可以用一个名字（数组名）来表示大批数据（数组元素），并且能够通过循环批处理大量数据。
- ▶ `cin >> a;`
- ▶ `cin >> a1 >> a2 >> a3 >> a4 >> a5 >> a6 >> a7 >> a8 >> a9 >> a10;`
- ▶ `for(i=0;i<100;i++) cin >> a[i];`

## 6.1.1 一维数组的定义

- ▶ 1. 一维数组的定义
- ▶ 一维数组的定义形式为：

**元素类型**    **数组名[常量表达式];**

```
int A[10];
```

```
int B[10], C[15]; //多个数组定义
```

```
int E[10], m, n, F[15]; //数组和变量混合在一起定义
```

## 6.1.1 一维数组的定义

---

- ▶ (1) 一维数组是由元素类型、数组名和长度组成的构造类型。例如：

```
int A[10], B[20]; // 元素是整型  
double F1[8], F2[10]; // 元素是双精度浮点型  
char S1[80], S2[80]; // 元素是字符型
```

## 6.1.1 一维数组的定义

---

- ▶ (2) 数组名必须符合C++标识符规则。
- ▶ (3) 常量表达式表示数组中元素的个数，称为数组长度。常量表达式的值必须为正整数且大于等于1。
- ▶ (4) 数组一经定义，数组长度就始终不变。

如： `int n; cin >> n; int a[n];`    **错误！**

## 6.1.1 一维数组的定义

- ▶ 2. 一维数组的内存形式
- ▶ C++规定数组元素是连续存放的，即在内存中一个元素紧跟着一个元素线性排列。

数组名

元素集合

34	45	0	-6	-77	1	90	.....	72
[0]	[1]	[2]	[3]	[4]	[5]	[6]		[n-1]



## 6.1.2 一维数组的初始化

- ▶ 一维数组的初始化
- ▶ 可以在一维数组定义时对它进行初始化，初始化的语法形式如下：

**元素类型**    **数组名**[**常量表达式**]={**初值列表**};

```
int A[5]={1,2,3,4,5} , B[3]={7,8,9};  
//一维数组初始化
```

## 6.1.2 一维数组的初始化

- ▶ (1) 初值列表提供的元素个数不能超过数组长度，但可以小于数组长度。如果初值个数小于数组长度，则只初始化前面的数组元素，剩余元素初始化为0。例如：

```
int A[5]={1,8,9};
```

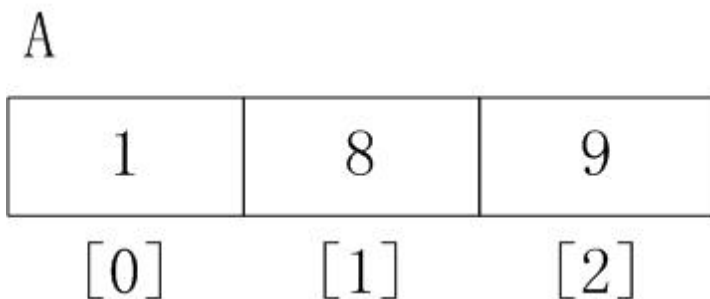
A

1	8	9	0	0
[0]	[1]	[2]	[3]	[4]

## 6.1.2 一维数组的初始化

- ▶ (2) 在提供了初值列表的前提下，数组定义时可以用不用指定数组长度，编译器会根据初值个数自动确定数组的长度。例如：

```
int A[]={1,8,9};
```



- ▶ 下面的表达式能够计算出数组A的长度：

```
sizeof(A) / sizeof(int)
```

- ▶ (3) 若数组未进行初始化，静态数组的元素均初始化为0；在函数体内定义的动态数组，其元素没有初始化，为一个随机值。
- ▶ 如：`static int A[10];` //默认各元素的值为0  
`int A[10];` //各元素的值为随机数

- ▶ 一维数组的引用
- ▶ 数组必须先定义后使用，且只能逐个引用数组元素的值而不能一次引用整个数组全部元素的值。
- ▶ 数组元素引用是通过下标得到的，一般形式为：

**数组名[下标表达式]**

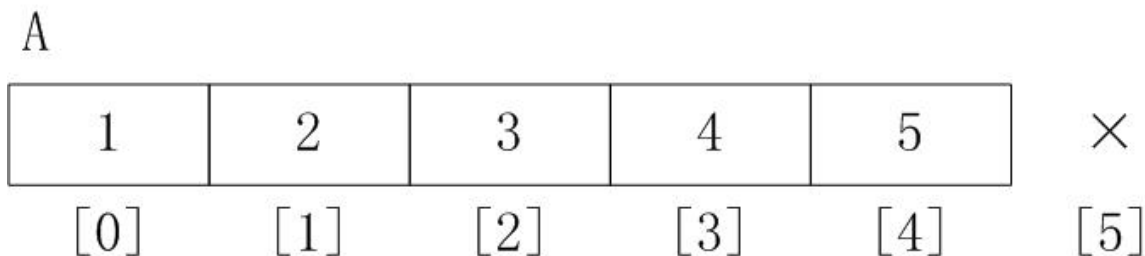
- ▶ (1) 注意下标的表示形式
- ▶ 下标表达式可以是常量、变量、表达式，但必须是正整数，不允许为负。
- ▶ 数组元素下标总是从0开始。

```
int A[5]={1,2,3,4,5}, x,i=3;  
x = A[0] ;  
A[1+2]=10;  
A[i]++;
```

## 6.1.3 一维数组的引用

- ▶ (2) 下标值不能超过数组长度，否则导致数组下标越界的**严重错误**。例如：

```
int A[5]={1,2,3,4,5};  
A[5]=10; //错误，没有A[5]元素
```



**注意：**

数组下标越界会使数据存取超过程序合法的内存空间，这样就可能会改写其他函数栈空间的数据，进而产生很严重的异常错误，甚至引起程序崩溃。**C++编译器不会检查数组是否越界**，需要程序员自己小心控制。



## 6.1.3 一维数组的引用

- ▶ (3) 整个数组不允许进行赋值运算、算术运算等操作，只有元素才可以，例如：

```
int A[10], B[10], C[10];  
A = B; //错误  
A = B + C; //错误  
A[0] = B[0]; //正确，数组元素赋值  
A[2] = B[2]+C[2]; //正确，数组元素赋值
```

- ▶ 从数组的内存形式来看，数组元素的下标是有序递增的，这个特点使得可以利用循环来批量处理数组元素。
  - ▶ （1）遍历数组元素
    - ▶ 利用循环逐个引用数组元素。
  - ▶ （2）数组元素复制
    - ▶ 利用循环通过元素逐个赋值，可以达到两个数组“赋值”的效果。

## 6.1.3 一维数组的引用

### 例6.1

```
1  #include <iostream>
2  using namespace std;
3  int main() //一维数组的输入输出
4  {
5      int i, A[100];
6      for(i=0;i<100;i++) cin>>A[i];
7      for(i=100-1;i>=0;i--) cout<<A[i]<<" ";
8      return 0;
9  }
```

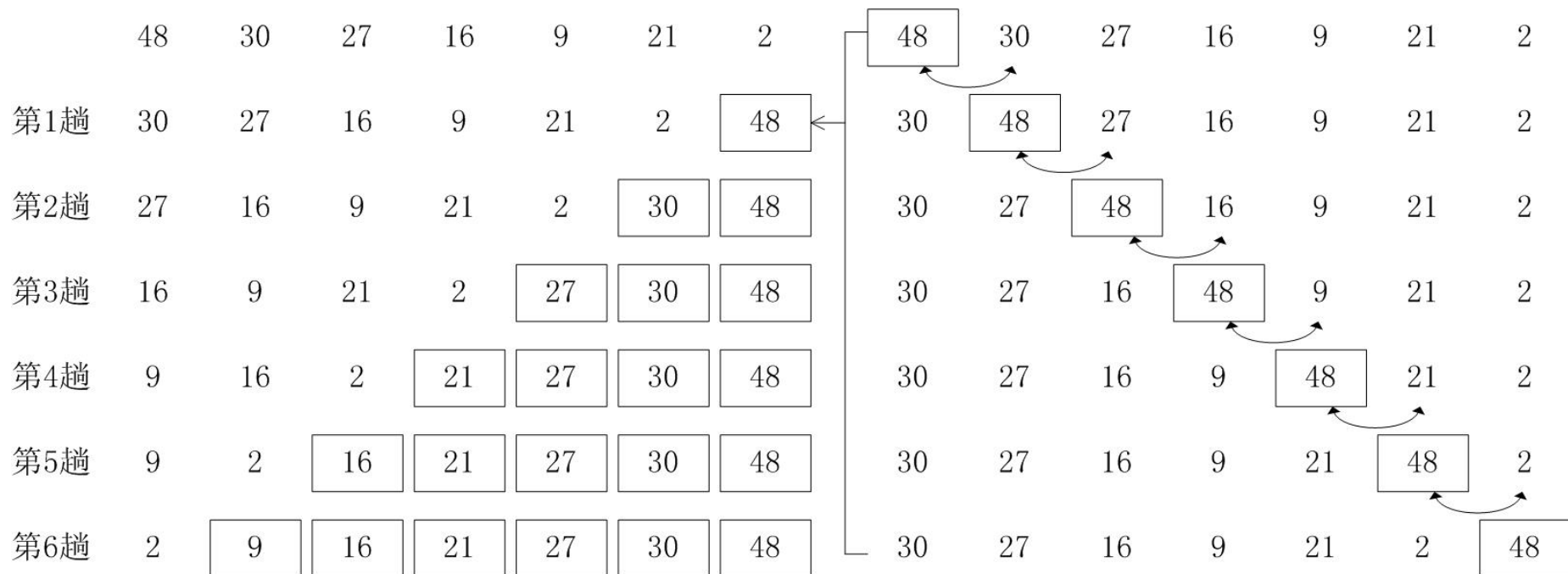
## 6.1.3 一维数组的引用

### 例6.2

```
1  #include <iostream>
2  int main()
3  {
4      int A[5]={1,2,3,4,5} , B[5] , i;
5      for (i=0; i<5; i++)
6          B[i] = A[i]; //元素一一复制
7      return 0;
8  }
```

## 6.6 数组应用程序举例

图6.10 冒泡排序



## 6.6 数组应用程序举例

30	27	27	27	27	27
27	30	16	16	16	16
16	16	30	9	9	9
9	9	9	30	21	21
21	21	21	21	30	2
2	2	2	2	2	<b>30</b>
<b>48</b>	<b>48</b>	<b>48</b>	<b>48</b>	<b>48</b>	<b>48</b>

27	16	16	16	16
16	27	9	9	9
9	9	27	21	21
21	21	21	27	2
2	2	2	2	<b>27</b>
<b>30</b>	<b>30</b>	<b>30</b>	<b>30</b>	<b>30</b>
<b>48</b>	<b>48</b>	<b>48</b>	<b>48</b>	<b>48</b>

16	9	9	9
9	16	16	16
21	21	21	2
2	2	2	<b>21</b>
<b>27</b>	<b>27</b>	<b>27</b>	<b>27</b>
<b>30</b>	<b>30</b>	<b>30</b>	<b>30</b>
<b>48</b>	<b>48</b>	<b>48</b>	<b>48</b>

9	9	9
16	16	2
2	2	<b>16</b>
<b>21</b>	<b>21</b>	<b>21</b>
<b>27</b>	<b>27</b>	<b>27</b>
<b>30</b>	<b>30</b>	<b>30</b>
<b>48</b>	<b>48</b>	<b>48</b>

9	2
2	<b>9</b>
<b>16</b>	<b>16</b>
<b>21</b>	<b>21</b>
<b>27</b>	<b>27</b>
<b>30</b>	<b>30</b>
<b>48</b>	<b>48</b>

## 6.6 数组应用程序举例

经第一趟（共6次）比较后，大数48沉底，小数浮起

经第二趟（共5次）比较后，次大数30沉底，排在倒数第二个位置

经第三趟（共4次）比较后，27沉底，排在倒数第三个位置

经第四趟（共3次）比较后，21沉底，排在倒数第四个位置

经第五趟（共2次）比较后，16沉底，排在倒数第五个位置

经第六趟（共1次）比较后，9沉底，排在倒数第六个位置

可推知，7个数要比较6趟。如果有 $n$ 个数，要进行 $n-1$ 趟比较。第 $j$ 趟要进行 $n-j$ 次两两比较（因为已有 $j-1$ 个数排好序）。

## 6.6 数组应用程序举例

### 例6.11

```
1  #include <iostream>
2  using namespace std;
3  #define N 7 //数组元素个数
4  int main()
5  {
6      int A[N], i, j, t; //注意数组下标从0开始
7      for (i=0; i<N; i++) cin>>A[i];
8      for(j=0 ; j<N-1 ; j++) //冒泡排序法
9          for(i=0 ; i<N-1-j; i++)
10             if(A[i] > A[i+1]) //<升序 >降序
11                 t=A[i],A[i]=A[i+1],A[i+1]=t;
12     for (i=0; i<N; i++) cout<<A[i]<<" ";
13     return 0;
14 }
```



## 6.2 多维数组的定义和引用

- ▶ 1. 多维数组的定义
- ▶ C++允许定义多维数组，其中二维数组的定义形式为：

**元素类型**    数组名 [ 常量表达式1 ] [ 常量表达式2 ] ;

**元素类型**    数组名 [ 常量表达式1 ] [ 常量表达式2 ] ... [ 常量表达式n ] ;

```
int A[3][4]; //定义二维数组
int B[3][4][5]; //定义三维数组
int C[3,4,5,6]; //错误!
```

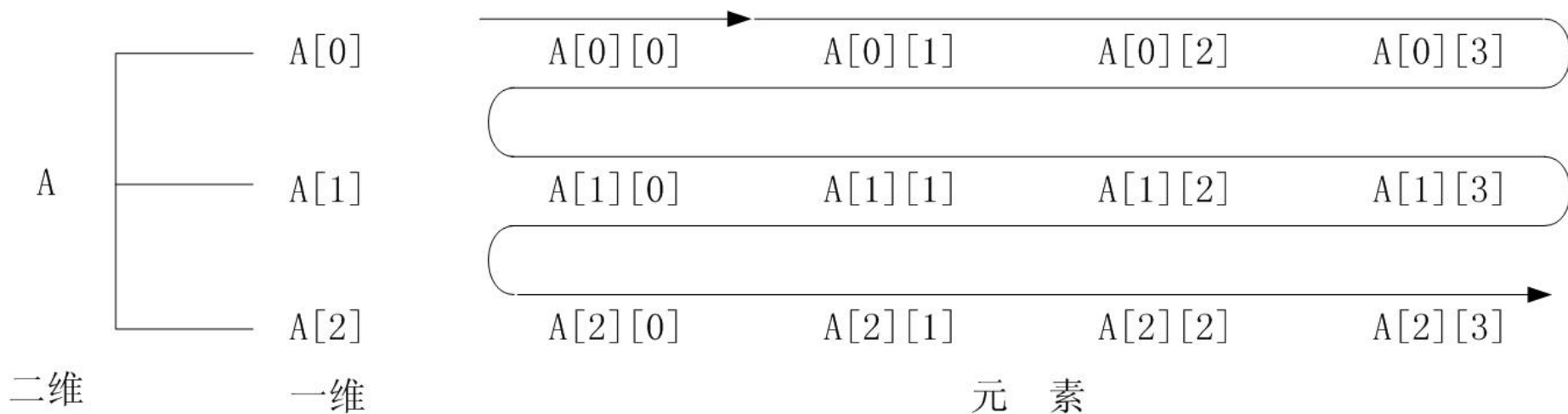
## 6.2.1 多维数组的定义

---

- ▶ 多维定义实际上是反复递归一维定义：即一维数组的每一个元素又是一个一维数组，就构成了二维数组。
- ▶ 本质上，C++的多维数组都是一维数组，这是由内存形式的线性排列决定的。因此，不能按几何中的概念来理解多维，多维数组不过是借用“维”的数学说法表示连续内存单元。

## 6.2.1 多维数组的定义

图6.2 二维数组的内存结构



## 6.2.1 多维数组的定义

例：int a[3][4];

假设每个元素占2个

字节的存储空间，

则存储结构图示：

	⋮		
2000	87	a[0][0]	} 0行
2002	73	a[0][1]	
2004	91	a[0][2]	
2006	76	a[0][3]	
2008	82	a[1][0]	} 1行
2010	89	a[1][1]	
2012	67	a[1][2]	
2014	73	a[1][3]	
2016	93	a[2][0]	} 2行
2018	70	a[2][1]	
2020	80	a[2][2]	
2022	82	a[2][3]	
	⋮		

## 6.2.2 多维数组的初始化

- ▶ 多维数组的初始化
- ▶ 可以在多维数组定义时对它进行初始化，这里以二维数组来说明，初始化有两种形式。
- ▶ ①初值按多维形式给出：

**元素类型**    **数组名**[常量表达式1][常量表达式2] = {{**初值列表1**},{**初值列表2**},...};

```
int A[2][3]={ {1,2,3},{4,5,6}}; //初值按二维形式
```

## 6.2.2 多维数组的初始化

► ②初值按一维形式给出：

**元素类型**    数组名[常量表达式1][常量表达式2] = {**初值列表**};

```
int A[2][3]={ 1,2,3,4,5,6 }; //初值按一维形式
```

## 6.2.2 多维数组的初始化

- ▶ 初值列表提供的元素个数不能超过数组长度，但可以小于数组长度。如果初值个数小于数组长度，则只初始化前面的数组元素；剩余元素初始化为0。这个规则两种初始化形式都适用，例如：

// 只对每行的前若干个元素赋初值

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

A

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

## 6.2.2 多维数组的初始化

//只对前若干行的前若干个元素赋初值

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

A

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



## 6.2.2 多维数组的初始化

//一维形式部分元素赋初值

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

A

[0]	1	2	3	4
[1]	5	0	0	0
[2]	0	0	0	0
	[0]	[1]	[2]	[3]

## 6.2.2 多维数组的初始化

- 在提供了初值列表的前提下，多维数组定义时可以用不用指定第1维的数组长度，但其余维的长度必须指定，编译器会根据列出的元素个数自动确定第1维的长度。例如：

```
int A[][2][3]={1,2,3,4,5,6,7,8,9,10,11,12};  
//正确  
int B[2][][3]={1,2,3,4,5,6,7,8,9,10,11,12};  
//错误，只能省略第1维  
int C[2][2][]={1,2,3,4,5,6,7,8,9,10,11,12};  
//错误，只能省略第1维
```



为什么其余维的长度必须要指定呢？

---

因为编译器需要确认多维数组的结构是唯一的

例如二维数组总共有12个元素，那么就会有 $2 \times 6$ 、 $3 \times 4$ 、 $4 \times 3$ 、 $6 \times 2$ 等不同形式的结构，指定了第2维为4，编译器就能确定二维数组是 $3 \times 4$ 。

## 6.2.2 多维数组的初始化

---

- ▶ 如果多维数组未进行初始化，那么静态数组的元素均初始化为0；如果动态数组的元素没有初始化，为一个随机值。

## 6.2.3 多维数组的引用

- ▶ 多维数组的引用
- ▶ 多维数组元素的引用与一维类似，也只能逐个引用数组元素的值而不能一次引用整个数组，引用的一般形式为：

数组名[下标表达式1][下标表达式2] ...[下标表达式n]

```
int b[3][4], i, j, sum=0;  
b[0][0]=10;  
b[1][2]=b[0][0]+10;  
b[2-1][2*2-1]= 5
```

- ▶ 下标表达式用来表示元素在数组中的位置，可以是常量、变量、表达式，但必须正整数，不允许为负。
- ▶ 每个维的下标总是从0开始。

## 6.2.3 多维数组的引用

```
int A[3][4]={1,2,3},x;  
x = A[0][1]; //x=2  
A[2][2] = 50; //则数组A变为
```

A

[0]	1	2	3	0
[1]	0	0	0	0
[2]	0	0	50	0
	[0]	[1]	[2]	[3]

## 6.2.3 多维数组的引用

例6.3 给二维数组输入数据，并以行列形式输出

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int A[3][4],i,j;
6      for (i=0;i<3;i++)
7          for (j=0;j<4;j++) cin >> A[i][j];
8      for (i=0;i<3;i++) {
9          for (j=0;j<4;j++) //内循环输出一行
10             cout<<A[i][j]<<" ";
11             cout<<endl; //每输出一行换行
12     }
13     return 0;
14 }
```



- ▶ 数组的定义： `int A[10]; int B[3][4];`
- ▶ 数组的下标：从0开始
- ▶ 数组的存储：连续存储
- ▶ 数组的引用：只能单独引用数组中的各个元素而不能一次引用给整个数组。
- ▶ 数组的遍历：

```
for(i=0;i<10;i++)  
    s=s+A[i];
```

```
for(i=0;i<3;i++)  
    for(j=0;j<4;j++)  
        cout<<B[i][j]<<' ';
```

- 【例6.4】 求矩阵的转置矩阵：

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

## 6.2.3 多维数组的引用

### 例6.4

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int A[2][3]={1,2,3},{4,5,6}
           ,AT[3][2], i, j;
6      for (i=0; i<2; i++) //求矩阵A的转置
7          for(j=0;j<3;j++) AT[j][i]=A[i][j];
8      cout<<"A="<<endl;
9      for (i=0; i<2; i++) { //输出矩阵A
10         for(j=0;j<3;j++) cout<<A[i][j]<<" ";
11         cout<<endl;
12     }
```

## 6.2.3 多维数组的引用

### 例6.4

```
13      cout<<"AT="<<endl;
14      for (i=0; i<3; i++) { //输出转置矩阵AT
15          for(j=0;j<2;j++) cout<<AT[i][j]<<" ";
16          cout<<endl;
17      }
18      return 0;
19 }
```

## 6.3 数组与函数

- ▶ 数组作为函数的参数
- ▶ 一维数组元素可以直接作为函数实参使用，其用法与变量相同。

```
int max(int a,int b);
```

```
int A[5]={1,2,3,4,5} , c=2, x;
```

```
x=max(c,-10); //使用变量作为函数实参
```

```
x=max(A[2],-10); //使用数组元素作为函数实参
```

## 6.3.1 数组作为函数的参数

- ▶ 数组作为函数的形参，基本形式为：

```
返回类型 函数名( 类型 数组名[常量表达式], ... )  
{  
    函数体  
}
```

```
double average(double A[100], int n)  
{  
    ... //函数体  
}
```

## 6.3.2 数组参数的传递机制

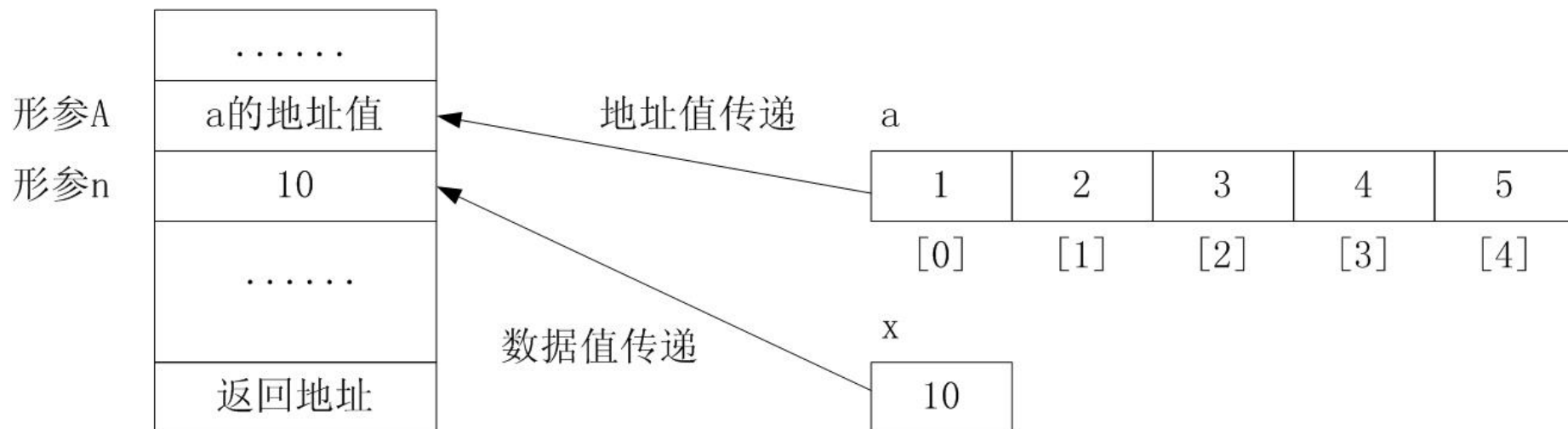
### ► 数组参数的传递机制

```
void fun(int A[10],int n);  
int main()  
{  
    int a[10]={1,2,3,4,5} , x=5;  
    fun(a,x); //实参分别是数组和整型变量  
}
```

## 6.3.2 数组参数的传递机制

- ▶ 如果实参使用数组名调用，本质上是**将这个数组的首地址传递到形参中**。
- ▶ 尽管数组数据很多，但它们均从一个首地址连续存放，这个首地址对应的正是数组名。

图6.3 数组首地址传递示意





## 6.3.2 数组参数的传递机制

- ▶ 数组实参a传的是数组首地址，形参A定义为数组形式，它现在的地址与实参数组a一样，则**本质上形参数组A就是实参数组a**（内存中两个对象所处位置相同，则它们实为同一个对象）。

```
void fun(int A[10],int n);  
int main()  
{  
    int a[10]={1,2,3,4,5} , x=5;  
    fun(a,x);  
}
```

- ▶ 这样的传递机制使得当数组作为函数参数时，有下面的特殊性。
- ▶ （1）由于形参数组就是实参数组，所以在被调函数中使用形参就是在间接使用实参，这点与变量作为函数参数的情况是不同的。

## 6.3.2 数组参数的传递机制

```
void fun(int A[5],int n)
{
    A[1]=100; //A[1]实质就是实参a[1]
    n=10; //赋值给形参n, 不影响实参x
}
void caller()
{
    int a[5]={1,2,3,4,5},x=5;
    fun(a,x);
    cout<<a[1]<<" , "<<x<<endl; //a[1]=100,x=5
}
```

- ▶ 【例6.6】 编写函数求一个二维数组中最大的元素及其下标。
- ▶ 分析
- ▶ 用max存放元素最大值，采用枚举法逐一比较二维数组中的每一个元素A[i][j]和max，若A[i][j]大于max说明有一个更大的值出现，则令max=A[i][j]且记录r=i和c=j，遍历完所有元素，则A[r][c]就是最大的元素。
- ▶ 由于max必然是数组中的一个元素值，故设置max的初值为A中一个元素值，例如A[0][0]。

- ▶ 由于函数需要返回最大元素值及下标行、列三个数据，而函数返回只能是一个数据，所以使用数组B传递到函数中，将下标行、列值“带回”。

## 6.3.2 数组参数的传递机制

例6.6 编写函数求一个二维数组中最大的元素及其下标。

```
1  #include <iostream>
2  using namespace std;
3  int findmax(int A[3][4],int B[2])
4  {
5      int i,j,max,r=0,c=0;
6      max=A[r][c]; //max初值设为A[0][0]
7      for (i=0; i<3; i++)//枚举二维数组所有元素
8          for (j=0; j<4; j++)
9              if (A[i][j]>max) {
10                  r=i,c=j; //记录此时的下标
11                  max=A[r][c]; //新的最大元素值;
12              }
13      B[0]=r, B[1]=c; //下标行、列通过B数组返回
14      return max; //最大值通过函数值返回
15 }
```

## 6.3.2 数组参数的传递机制

### 例6.6

```
16  int main()
17  {
18      int A[3][4]={ {7,5,-2,4}, {5,1,9,7}
                    , {3,2,-1,6}}, B[2], max;
19      max=findmax(A,B);
20      cout<<"max:A["<<B[0]<<"]["<<B[1]
          <<"]="<<max<<endl;
21      return 0;
22  }
```

- ▶ (2) 既然形参数组就是实参数组，所以函数调用时不会为形参数组分配存储空间。
- ▶ 形参数组不过是用数组定义这样的形式来表明它是个数组，能够接收实参传来的地址，形参数组的长度说明也无实际作用。因此形参数组甚至可以不用给出长度。



## 6.3.2 数组参数的传递机制

假设有函数调用：

```
int a[15];  
f(a);
```

则以下函数定义：

```
void f(int A[100]); //形参数组长度完全由实参数组确定，因此函数中并不能按100个元素处理  
void f(int A[10]); //形参数组长度完全由实参数组确定，因此函数中并不能按10个元素处理  
void f(int A[]); //表明形参是数组形式即可
```

均是正确的。

- ▶ (3) 虽然实参数组将地址传到了被调函数中，但被调函数并不知道实参数组的具体长度，那么假定的大小对于实参数组来说容易数组越界。
- ▶ 实际编程中可以采用以下方法来解决：
- ▶ 函数调用时再给出一个参数来表示实参数组的长度；

## 6.3.2 数组参数的传递机制

例6.7 编写函数，求数组各元素的平均值。

```
1  #include <iostream>
2  using namespace std;
3  double average(double A[],int n)
4  {
5      int i; double s=0; //累加初值为0
6      for (i=0; i<n; i++) s=s+A[i]; //先累加
7      return n!=0 ? s/n : 0.0; //计算平均值
8  }
9  int main()
10 {
11     double A[3]={1,2,3};
12     double B[5]={1,2,3,4,5};
13     cout<<"A="<<average(A,3)<<endl;
14     cout<<"B="<<average(B,5)<<endl;
15     return 0;
16 }
```

- ▶ （4）多维数组作为函数的参数，形参数组第1维可以与实参相同，也可以不相同；可以是任意长度，也可以不写长度；但其他维的长度需要相同。

## 6.3.2 数组参数的传递机制

例如有函数调用：

```
int a[5][10]  
f(a);
```

则函数定义：

```
void f(int A[5][10]); //正确  
void f(int A[2][10]); //正确  
void f(int A[][10]); //正确  
void f(int A[][]); //错误，第2维长度必须给出  
void f(int A[5][5]); //错误，第2维长度必须相同  
void f(int A[50]); //错误，必须是二维数组
```

## 复习:

- ▶ 数组实参a传的是数组首地址，形参b定义为数组形式，它现在的地址与实参数组a一样，则**本质上形参数组b就是实参数组a**（内存中两个对象所处位置相同，则它们实为同一个对象）。

```
void fun(int b[10],int n)
{
    ....
}
int main()
{
    int a[10]={1,2,3,4,5} , x=5;
    fun(a,x);
}
```

## 复习:

//fun函数的作用为将数组a的每个元素值加1

```
void fun(int b[5])
{
    int i;
    for(i=0;i<5;i++)
        b[i]=b[i]+1;
}

int main()
{
    int a[5]={1,2,3,4,5} ;
    fun(a);
}
```

## 复习:

例: 编写一个函数swap用于交换两个数的值

如, a=3, b=4, 调用函数swap后, 得a=4, b=3

方法1:

```
void swap(int x,int y)
{ int t;
  t=x; x=y; y=t;
}

int main( )
{ int a,b;
  cin>>a>>b;
  swap(a,b);
  cout<<a<<","<<b<<endl;
  return 0;
}
```

方法2:

```
void swap(int b[2])
{ int t;
  t=b[0]; b[0]=b[1]; b[1]=t;
}

int main( )
{ int a[2];
  cin>>a[0]>>a[1];
  swap(a);
  cout<<a[0]<<","<<a[1]<<endl;
  return 0;
}
```



- ▶ 字符数组
- ▶ 用来存放字符型数据的数组称为字符数组，其元素是一个字符，定义形式为：

```
char 字符数组名[常量表达式], .....;
```

例如：

```
char s[20]; //定义字符数组
```

```
char s[4]={'J','a','v','a'}; //字符数组初始化
```

- ▶ 1. 字符串的概念
- ▶ C++语言规定字符串是以'\0'（ASCII值为0）字符作为结束符的字符数组。
- ▶ 在程序中可以通过判断数组元素是否为空字符来判断字符串是否结束，换言之，只要遇到数组元素是空字符，就表示字符串在此位置上结束。
- ▶ 由于字符串实际存放在字符数组中，所以定义字符数组时数组的长度至少为字符串长度加1（空字符也要占位）。

- ▶ **字符串常量**是字符串的常量形式，它是以一对双引号括起来的字符序列。
- ▶ C++总是在编译时为字符串常量自动在其后增加一个空字符。
- ▶ 区分 “a” 和 ‘a’ 和a

```
char c;  
char s[10]="hello."  
c='a'; //正确  
c="a"; //错误
```

## 6.4.2 字符串

(1) C++语言总是在编译时为字符串常量自动在其后增加一个空字符，例如"Hello"的存储形式为：

H	e	l	l	o	\0
[0]	[1]	[2]	[3]	[4]	[5]

(2) 即使人为在后面加上空字符也是如此，例如"Hello\0"的存储形式为：

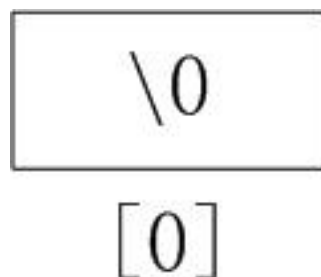
H	e	l	l	o	\0	\0
[0]	[1]	[2]	[3]	[4]	[5]	[6]

## 6.4.2 字符串

(3) 如果在字符串常量中插入空字符，则字符串常量的长度会比看到的字符数目少，例如"ABC\0DEF"的存储形式为：

A	B	C	\0	D	E	F	\0
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

(4) 空字符串尽管字符串长度是0，但它依然要占据字符数组空间，例如空字符串""的存储形式为：



- ▶ (5) 空字符 (`'\0'`) 容易与字符 `'0'` 混淆, 其实它们是有区别的。
- ▶ `'\0'` 的ASCII值为0,
- ▶ `'0'` 的ASCII值为48。
- ▶ 输出时,
- ▶ `'0'` 会在屏幕上显示0这个符号,
- ▶ 而 `'\0'` 什么也没有。

- ▶ 文本信息用途非常广，无处不在。如姓名、通信地址、邮箱等，即使像邮政编码这样的数字，也属于文本信息范畴。
- ▶ 有了字符串的概念，C++程序能方便地表示文本信息。
- ▶ 尽管字符串不是C++语言的内置数据类型，但应用程序通常都将它当作基本类型来用，称为C风格字符串（C-style string）。

## 6.4.3 字符串的输入和输出

- ▶ 字符串的输入和输出有三种方法。
- ▶ 1. 逐个字符输入输出
- ▶ 2. 使用格式化输入输出函数，将整个字符串一次输入或输出。

```
char str[80]; //定义字符串也即定义字符数组
scanf("%s",str); //使用%s格式，输入字符串
printf("%s",str); //使用%s格式，输出字符串
scanf("%s",str[0]); //错误，%s格式要求字符串
printf("%s",str[0]); //错误，%s格式要求字符串
```



3. 使用标准输入输出流，将整个字符串一次输入或输出。

```
char str[80];  
cin >> str; //输入字符串  
cin.getline(str,79); //输入带空格的字符串  
cout << str; //输出字符串
```

- ▶ 4. 使用字符串输入输出函数
- ▶ (1) gets函数


```
char *gets(char *s);
```

- ▶ gets函数输入一个字符串到字符数组s中。s是字符数组或指向字符数组的指针，其长度应该足够大，以便能容纳输入的字符串。
- ▶ gets函数可以输入空格和TAB，但不能输入回车。
- ▶ gets函数输入完成后，在字符串末尾自动添加空字符。

## 6.4.3 字符串的输入和输出

s

C	o	m	p	u	t	e	r	\0	.....
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	



```
char str[80];  
gets(str); //输入字符串
```

Computer ↙

- ▶ 说明：
- ▶ gets函数可以输入空格和TAB，但不能输入回车。
- ▶ gets函数输入完成后，在字符串末尾自动添加空字符。

## 6.4.3 字符串的输入和输出

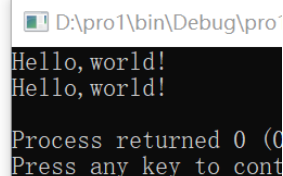
- ▶ 3. 使用字符串输入输出函数
- ▶ (2) puts函数

```
int puts(char *s);
```

- ▶ puts函数输出s字符串，遇到空字符结束，输完后再输出一个换行（'\n'）。s是字符数组或指向字符数组的指针，返回值表示输出字符的个数。

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    char str[13]="Hello,world!";
    puts(str);
    cout<<str<<endl;
    return 0;
}
```



```
Hello, world!
Hello, world!

Process returned 0 (0)
Press any key to continue...
```

### 6.4.3 字符串的输入和输出

```
char str[80]="Programming";  
puts(str); //输出字符串
```

程序运行屏幕

Programming



- ▶ 字符串处理函数
- ▶ C++标准库提供了兼容C语言的字符串处理函数，其头文件为string.h。

## 6.4.5 字符串处理函数

- ▶ (1) 字符串复制函数strcpy (string copy)

```
char str1[10],str2[]="Computer";  
strcpy(str1,str2); //复制str2到str1
```

- ▶ (2) 字符串连接函数strcat (string concatenate)

```
char str1[10]="ABC", str2[]="123";  
strcat(str1,str2); //在str1后面连接str2,str2未变化
```

- ▶ (3) 字符串比较函数strcmp (string compare)

```
if (strcmp(str1,str2)==0)..... //比较字符串相等  
if (strcmp(str1,str2)>0)..... //比较str1大于str2
```



## 复习:

---

- ▶ 字符串
- ▶ `char c[10]="hello"; cout<<c;`
- ▶ 字符串的输入输出:
- ▶ `cin>>c; cout<<c;`
- ▶ `gets(c); puts(c);`
- ▶ 字符串处理函数: `strcpy, strcat, strcmp`

- ▶ 使用C风格字符串，字符串本质上是字符数组。为了存放字符串，必须定义一个字符数组。由于字符数组总是有固定大小的，对于字符串处理，如复制、合并等，如果没有足够的数组长度，就容易导致数组越界。因此用字符数组来存放字符串不是安全的方法。

- ▶ C++为此提供了一种新的自定义类型：字符串类 `string`。采用类来实现字符串，具有如下特点：
- ▶ ①采用动态内存管理，不必担心存储空间是否足够，甚至都不用有字符数组的概念；
- ▶ ②能够检测和控制诸如越界之类的异常，提高使用的安全性；
- ▶ ③封装字符串多种处理操作，功能增强；
- ▶ ④可以按运算符形式操作字符串，使用简单。
- ▶ 因此，C++程序中使用 `string` 类型，比使用C风格字符串更方便、更安全。

## 6.5.1 字符串对象的定义和引用

---

- ▶ 使用string类需要将其头文件包含到程序中，预处理命令为：

```
#include <string> //不能写为string.h
```

## 6.5.1 字符串对象的定义和引用

- ▶ 1. 字符串对象的定义和初始化
- ▶ 定义和初始化字符串对象，与变量的方法类似。如果string对象没有初始化则一律是空字符串，即串中没有任何字符。需要注意的是C++字符串不需要NULL字符结尾。

```
string str1; //定义string对象
string str2 , str3 ; //定义多个string对象
string str2="Java"; //string对象复制初始化
string str3("C++"); //string对象直接初始化
```

## 6.5.1 字符串对象的定义和引用

- ▶ 2. 字符串对象的引用
- ▶ 与变量类似，直接使用string对象名就表示它的引用

```
string str1; //使用string对象  
str1 = "Pascal"; //正确
```

```
char str1[100] = "Pascal"; //使用字符数组  
char str2[100];  
str2="Pascal"; //错误！字符数组不能直接赋值
```

- ▶ 3. 字符串对象的输入和输出
- ▶ 可以在输入输出语句中直接使用string对象来输入输出字符串

```
cin  >> str1; //输入字符串到str1对象中存放  
cout << str1; //输出str2对象中的字符串  
getline(cin,str1); //输入带有空格或tab的字符串
```

- ▶ 字符串对象的操作
- ▶ string对象允许使用运算符进行操作，实现类似C风格字符串的处理。如复制（strcpy）、连接（strcat）、比较（strcmp）等。



- ▶ 1. 字符串赋值
- ▶ string对象可以使用赋值运算符，其功能是字符串复制。

```
str1 = "Pascal"; //字符串常量复制到string对象中  
strcpy(S1, "Pascal"); //C风格字符串复制到字符数组中
```

- ▶ 2. 字符串连接运算
- ▶ string对象允许使用加号（+）和复合赋值（+=）运算符来实现两个字符串连接操作。

```
str1="12" , str2="AB" , str3="CD";
```

```
①str1 = str2 + str3; //str1结果为ABCD
```

```
②str1 = str1 + str2; //str1结果为12AB
```

```
③str1 = str1 + "PHP"; //str1结果为12PHP
```

```
④str1 += str3; //str1结果为12CD
```

```
⑤str1 += "PHP"; //str1结果为12PHP
```

- ▶ 3. 字符串关系运算
- ▶ string对象可以使用关系运算符来对字符串进行比较

```
str1="ABC" , str2="XYZ";  
str1 > str2; //结果为假  
str1 == str2; //结果为假  
str1 == "ABC"; //结果为真
```

- ▶ 4. 其他操作
- ▶ string对象可以调用其成员函数来实现字符串处理，这里列举一些重要的操作，更详细的内容可以查阅C++标准库手册。

```
str1="ABCDEFGHIIJK";  
//获取字符串的长度  
n = str1.size(); //n为11  
n = str1.length(); //n为11  
//检查字符串是否为空字符串  
b = str1.empty(); //b为假
```

## 6.5.2 字符串对象的操作

//得到子字符串

`str2 = str1.substr(2,4);` //从下标2开始的4个字符,  
`str2`为CDEF

//查找子字符串

`n = str1.find("DEF",pos);` //从`pos`开始查找字符串  
"DEF"在`str1`中的位置, `n`为3

//删除字符

`str1.erase(3,5);` //从下标3开始往后删5个字符, `str1`变  
为ABC~~IJK~~

//增加字符

`str1.append("12345",1,3);` //在`str1`末尾增加  
"12345"串下标从1开始的3个字符, 即"234"

## 6.5.2 字符串对象的操作

### //字符串替换和插入操作

`str1.replace(p0,n0,S1,n);` // 删除从p0开始的n0个字符，然后在p0处插入字符串S1前n个字符

`str1.replace(p0,n0,str2,pos,n);` // 删除从p0开始的n0个字符，然后在p0处插入字符串str2中pos开始的n个字符

`str1.insert(p0,S1,n);` // 在p0位置插入字符串S1前n个字符

`str1.insert(p0,str2,pos,n);` // 在p0位置插入字符串str2中pos开始的n个字符

## 6.5.3 字符串对象数组

- ▶ 字符串对象数组
- ▶ 可以定义字符串对象数组，即数组元素是字符串对象，定义形式与数组类似，例如：

```
string SY[5]={"Jiang","Cao","Zou","Liu","Wei"};  
//定义字符串对象数组且初始化
```

SY[0]	J	i	a	n	g
SY[1]	C	a	o		
SY[2]	Z	h	o	u	
SY[3]	L	i	u		
SY[4]	W	e	i		

## 6.5.3 字符串对象数组

```
string SY[5]={“C++”, “Java”, “C”, “PHP”, “CSharp”, “Basic”};  
//长度 3,2,4,1,5
```

char

```
sa[5][20]={“C++”, “Java”, “C”, “PHP”, “CSharp”, “Basic”};  
//长度均是20
```

sa

sa[0]	C	+	+	\0			
sa[1]	J	a	v	a	\0		
sa[2]	C	\0					
sa[3]	P	H	P	\0			
sa[4]	C	S	h	a	r	p	\0
sa[5]	B	a	s	i	c	\0	



### ▶ 1. 排序

- ▶ 排序问题是程序设计中的典型问题，它有很广泛的应用，其功能是将一个数据元素序列的无序序列调整为有序序列。

### ▶ (1) 冒泡排序法

- ▶ 冒泡排序法 (bubble sort) 的基本思想是通过相邻两个记录之间的比较和交换, 使关键码较小的记录逐渐从底部移向顶部 (上升), 关键码较大的记录逐渐从顶部移向底部 (沉底), 冒泡由此得名。设由  $A[1] \sim A[n]$  组成的  $n$  个数据, 冒泡排序的过程可以描述为:

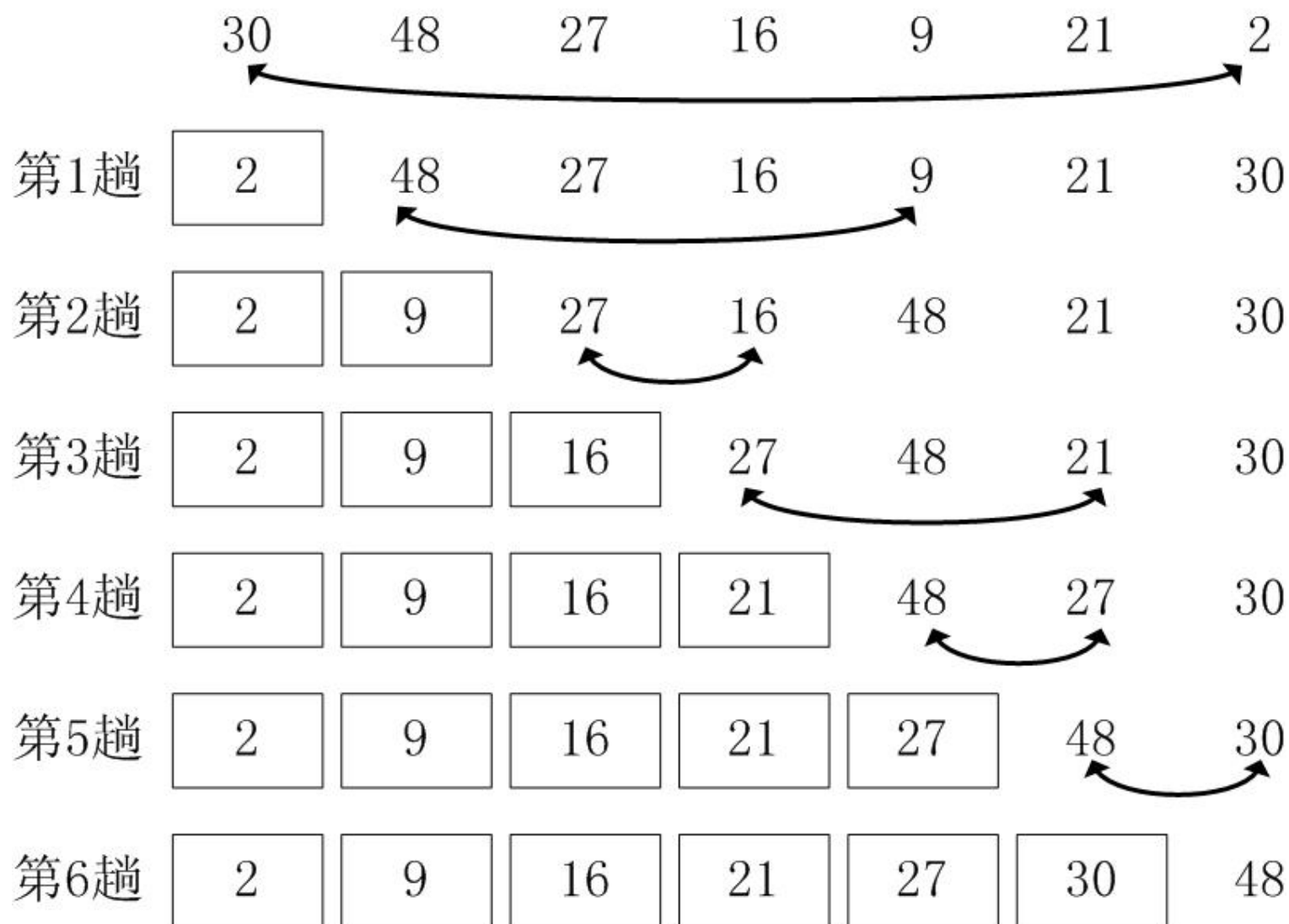
### ▶ (2) 选择排序法

- ▶ 选择排序法 (selection sort) 的基本思想是第 $i$ 趟选择排序通过 $n-i$ 次关键码的比较, 从 $n-i+1$ 个记录中选出关键码最小的记录, 并和第 $i$ 个记录进行交换。设由 $A[1] \sim A[n]$ 组成的 $n$ 个数据, 选择排序的过程可以描述为:

- ▶ (2) 选择排序法
- ▶ ①首先在 $A[1] \sim A[n]$ 区间内进行比较，从 $n$ 个记录中选出最小的记录 $A[k]$ ，若 $k$ 不为1则将 $A[1]$ 和 $A[k]$ 交换， $A[1]$ 为具有最小关键码的元素，称第1趟排序结束。
- ▶ ②然后在 $A[i] \sim A[n]$ 区间内，进行第 $i$ 趟排序，从 $n-i+1$ 个记录中选出最小的记录 $A[k]$ ，若 $k$ 不为 $i$ 则将 $A[i]$ 和 $A[k]$ 交换。重复进行 $n-1$ 趟后，整个排序过程结束。

## 6.6 数组应用程序举例

图6.11 选择排序



## 6.6 数组应用程序举例

### 例6.12

```
1  #include <iostream>
2  #include <ctime>
3  using namespace std;
4  void SelectionSort(int A[],int n)
5  {    //选择排序 n为数组元素个数
6      int i,j,k,t;
7      for(i=0; i<n-1; i++) { //选择排序法
8          k=i;
9          for(j=i+1; j<n; j++) //一趟选择排序
10             if (A[j] < A[k]) k=j;
11             if(i!=k) t=A[i],A[i]=A[k],A[k]=t;
12     }
13 }
```

## 6.6 数组应用程序举例

### 例6.12

```
14  #define N 7
15  int main()
16  {
17      int A[N], i;
18      srand((unsigned int)time(0));
19      for(i=0; i<N; i++) { //随机产生N个数
20          A[i] = rand()%100;
21          cout<<A[i]<<" ";
22      }
23      cout<<endl;
24      SelectionSort(A, N);
25      for(i=0; i<N; i++) cout<<A[i]<<" ";
26      return 0;
27  }
```

### ▶ 2. 查找

#### ▶ (1) 顺序查找法

- ▶ 顺序查找的基本思想是让关键字与序列中的数逐个比较，直到找出与给定关键字相同的数为止或序列结束，一般应用于无序序列查找。



## 6.6 数组应用程序举例

### 例6.15

```
1  #include <iostream>
2  using namespace std;
3  int Search(int A[],int n,int find)
4  {    // 顺序查找 n=序列元素个数 find=欲查找数据
5      int i;
6      for (i=0; i<n ; i++)
7          if (A[i]==find) return i;
8      return -1; //未找到
9  }
```

## 6.6 数组应用程序举例

### 例6.15

```
9  #define N 10
10 int main()
11 {
12     int A[N]={18,-3,-12,34,101,211,12,
13             90,77,45}, i, find;
14     cin>>find;
15     i=Search(A,N,find);
16     if(i>=0) cout<<"A["<<i<<"]="<<find<<endl;
17     else cout<<"not found"<<endl;
18     return 0;
19 }
```

### ▶ 2. 查找

#### ▶ (2) 二分查找法

- ▶ 对于有序序列，可以采用二分查找法进行查找。它的基本思想是：**升序排列**的 $n$ 个元素集合 $A$ 分成个数大致相同的两部分，取 $A[n/2]$ 与欲查找的 $find$ 作比较，如果相等则表示找到 $find$ ，算法终止。如果 $find < A[n/2]$ ，则在 $A$ 的前半部继续搜索 $find$ ，如果 $find > A[n/2]$ ，则在 $A$ 的后半部继续搜索 $find$ 。

## 6.6 数组应用程序举例

### 例6.16

```
1  #include <iostream>
2  using namespace std;
3  int BinarySearch(int A[],int n,int find)
4  {    //二分查找 n=序列元素个数 find=欲查找数据
5      int low,upper,mid;
6      low=0 , upper=n-1; //左右两部分
7      while(low<=upper) {
8          mid = low + (upper-low)/2;
9          //不用(upper+low)/2, 避免upper+low溢出
10         if( A[mid] < find) low = mid+1;
11         else if (A[mid]>find) upper=mid-1;
12         else return mid; //找到
13     }
14     return -1; //未找到
15 }
```

## 6.6 数组应用程序举例

### 例6.16

```
15  #define N 10
16  int main()
17  {
18      int A[N]={8,24,30,47,62,68,83,
                90,92,95},i,find;
19      cin>>find;
20      i=BinarySearch(A,N,find);
21      if(i >= 0) cout<<"A["<<i<<"]="
                <<find<<endl;
22      else cout<<"not found"<<endl;
23      return 0;
24  }
```

**CP<sup>®</sup>程序设计**