西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

批量数据的表示与处理

2、多维数组的定义与初始化

6.2 多维数组的定义和引用

- C语言允许多维数组。

6.2.1 多维数组的定义

- 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], ......;
```

- 多维数组的通用定义形式为：

```
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] , ......;
```

6.2.1 多维数组的定义

- 例如：

```
int A[3][4]; //定义二维数组
```

- 又如：

```
int B[3][4][5]; //定义三维数组
int C[3][4][5][6]; //定义四维数组
```

6.2.1 多维数组的定义

- 1．定义说明
- （1）多维数组的元素类型、数组名和常量表达式的含义和要求完全与一维数组类似。
- （2）显然，这里用方括号（[ ]）对应了维数，有多少对方括号就称多少维。约定多维数组越往左称为“高维”，越往右称为“低维”，最左边的称为“第1维”，往右依次类推。第1维的数组长度由常量表达式1决定，其余依此类推。

6.2.1 多维数组的定义

- （3）本质上，C语言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。

- 多维定义实际上是反复递归一维定义：即N维数组是一个集合，包含多个元素，每一个元素又是一个N－1维数组。

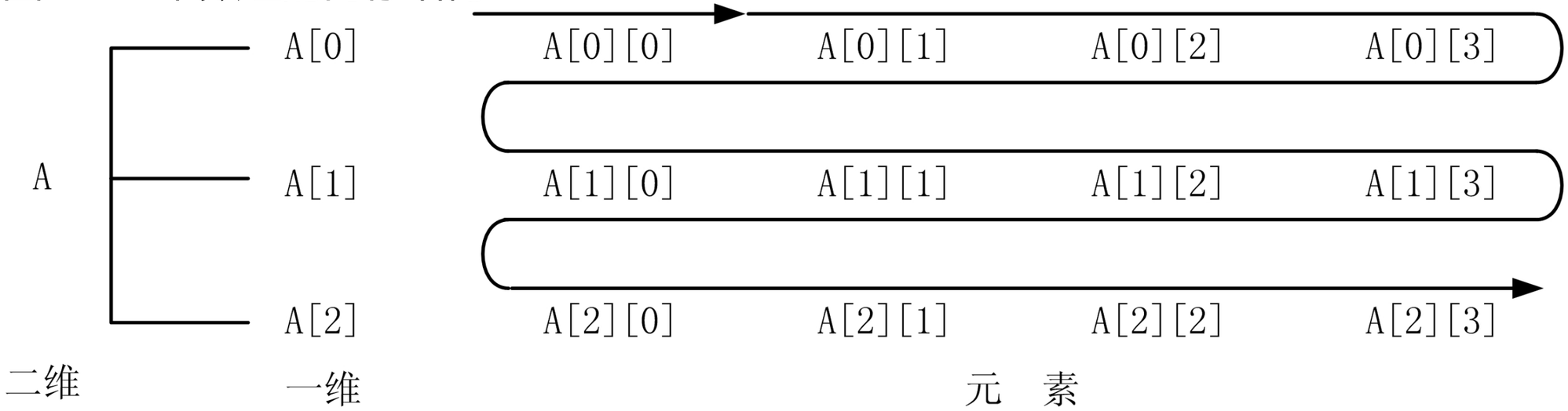
6.2.1 多维数组的定义

- 例如二维数组

```
int A[3][4];
```

6.2.1 多维数组的定义

图6.2 二维数组的内存结构

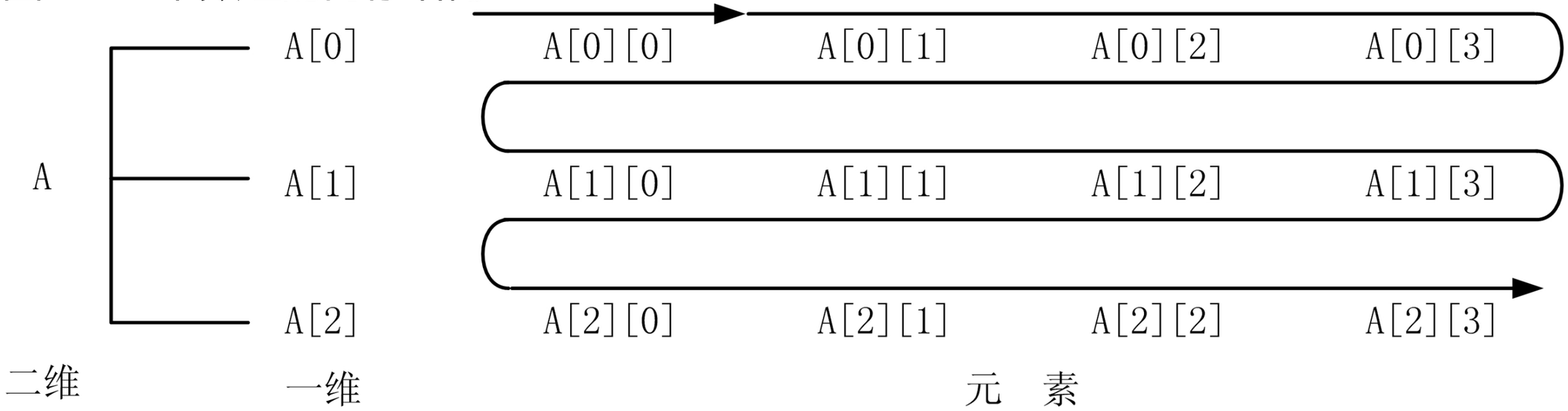


①若A是二维数组，则A[0]、A[1]、A[2]是它的元素，是一维数组，如图所示。

②若A[0]是一维数组，则A[0][0]、A[0][1]、A[0][2]、A[0][3]是它的元素。

6.2.1 多维数组的定义

图6.2 二维数组的内存结构



③A[0]的下一个是A[1]，A[1]的下一个是A[2]，其余依次类推。

④A[0][0]的下一个A[0][1]，A[0][3]的下一个是A[1][0]，其余依次类推。

6.2.1 多维数组的定义

- 2. 多维数组的内存形式
- C语言在编译时会将任何多维数组的引用转化为一维数组的形式，例如：

```
int A[3][4][5];
```

- 实质上与

```
int A[60];
```

- 等价

6.2.1 多维数组的定义

- 则元素：

```
A[i][j][k]
```

- 实质上是：

```
A[i*4*5+j*5+k]
```

- 多维数组的内存形式就是这样的一维数组内存形式，元素也是连续存放的。

6.2.2 多维数组的初始化

- 多维数组的初始化
- 可以在多维数组对象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},.....};
```

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};
```

- 例如下面两种写法完全等价：

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

- （1）可以用一维初值形式来对二维数组初始化，本质上是因为二维数组的内存形式就是一维数组的内存形式。不过一维形式的初值写法不如二维形式清晰，元素对应关系不容易直接看出。

6.2.2 多维数组的初始化

- （2）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。

6.2.2 多维数组的初始化

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

6.2.2 多维数组的初始化

- （3）在提供了初值列表的前提下，多维数组定义时可以不用指定第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
```

- 因为每个第1维元素（数组）的个数为2×3，为了能存储12个初值，至少第1维长度为2。

6.2.2 多维数组的初始化

- 下面的表达式能够计算出第1维的长度：

```
sizeof A / (sizeof(int) * 2 * 3)
//数组内存长度/(元素内存长度*2*3)
```

6.2.2 多维数组的初始化

- 这种情况下，列出的元素个数可能少于数组长度。例如：

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

- 第1维长度依然为2。即第1维长度的计算原则是确保多维数组能够容纳列出的元素个数所必需的最小长度。

6.2.2 多维数组的初始化

- 为什么其余维的长度必须要指定呢？那是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2×6、3×4、4×3、6×2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3×4。

6.2.2 多维数组的初始化

- （4）如果多维数组未进行初始化，那么在函数体外定义的静态数组对象，其元素均初始化为0；在函数体内定义的动态数组对象，其元素没有初始化，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对象全部元素的值，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]
```

6.2.3 多维数组的引用

- 下标表达式用来索引元素在数组中的位置，可以是常量、变量及其表达式，但必须是无符号整型数据，不允许为负。每个维的下标总是从0开始，与其内存形式对应，而且相互独立。所谓相互独立是指多维数组中的多个下标表达式相互是不关联的，各自索引在本维上的元素。

6.2.3 多维数组的引用

- 例如：

```
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	4
[1]	5	0	0	0
[2]	0	0	50	0
	[0]	[1]	[2]	[3]


程序设计