郑州大学

程序设计基础（C）—第6章 数组

郑州大学软件学院/网络空间安全学院

Lecturer：宋轩
Office ： 行政楼-306
Email ： songxuan@zzu.edu.cn

第6章 数组——多维数组

C语言中的数组有多个下标。C标准中的多维数组常用来表示按行、列排列的信息构成的表格。

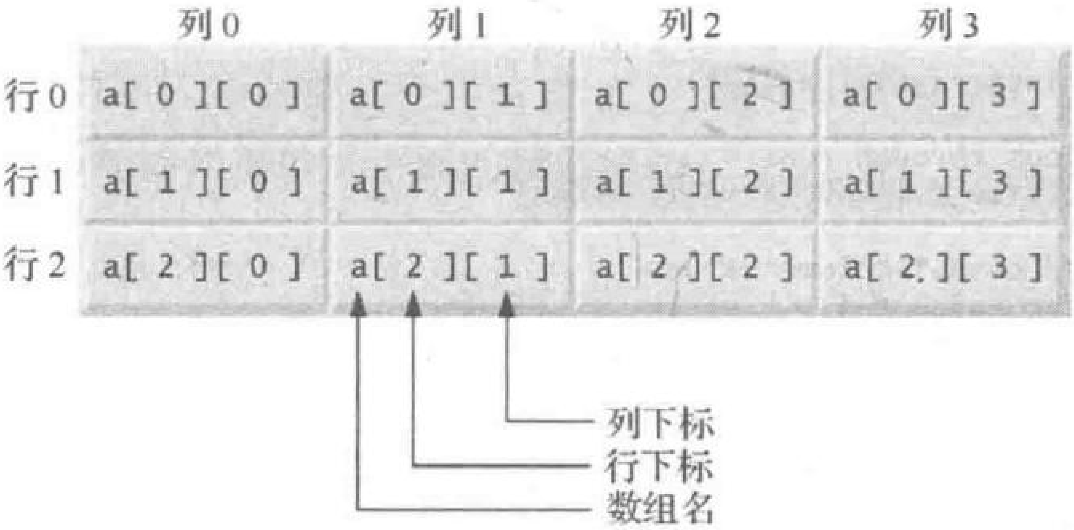
为了确定表中的元素，必须指定两个下标：

第一个下标确定的是元素所在行号，

第二个下标确定的是元素所在列号。

由两个下标确定的表格或数组称为二维数组。

C语言中的数组有多个下标。C标准中的多维数组常用来表示按行、列排列的信息构成的表格。



	列 0	列 1	列 2	列 3
行 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
行 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
行 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

1. 双下标数组的初始化

定义时初始化：

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

如果没有指定足够的初始值，那么该行中剩余元素的数组将被初始化为0，如：

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

结果是b[0][0]=1;

b[0][1]=0;

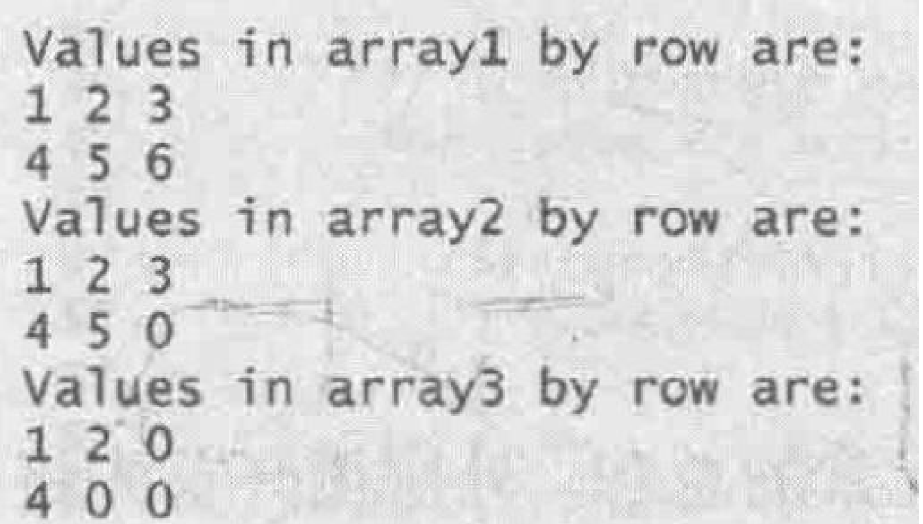
b[1][0]=3;

b[1][1]=4;

数组初始化的例子

```
#include <stdio.h>

void printArray(int a[][3]); // 函数原型

int main(void)
{
   int array1[2][3] = { { 1, 2, 3 }, { 4, 5, 6 } };
   puts("Values in array1 by row are:");
   printArray(array1);

   int array2[2][3] = { 1, 2, 3, 4, 5 };
   puts("Values in array2 by row are:");
   printArray(array2);

   int array3[2][3] = { { 1, 2 }, { 4 } };
   puts("Values in array3 by row are:");
   printArray(array3);
}
```

```
//函数定义
void printArray(int a[][3])
{
  // 行循环
  for (size_t i = 0; i <= 1; ++i) {
     // 列循环
    for (size_t j = 0; j <= 2; ++j) {
      printf("%d ", a[i][j]);
    }
    printf("\n");
  }
}
```

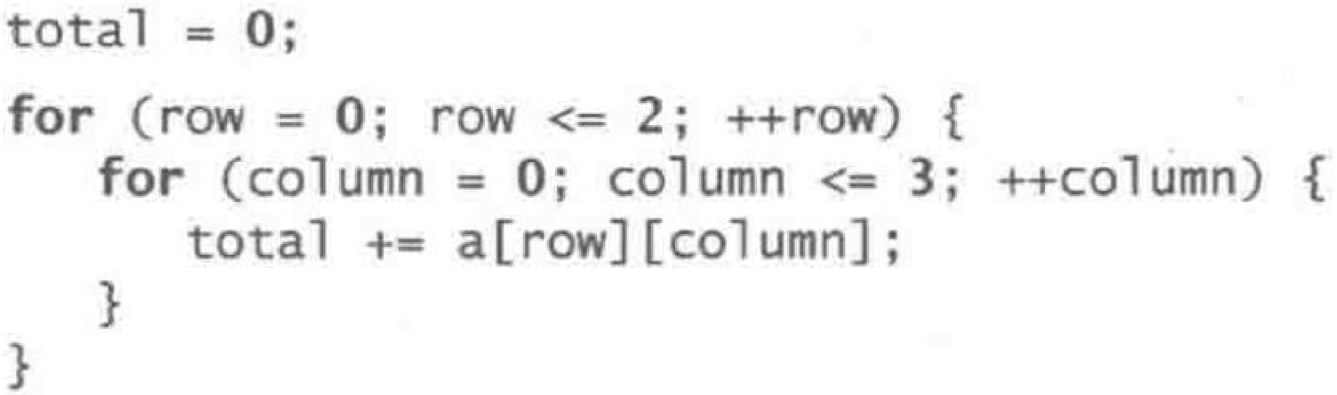


```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```

2. 设置某行元素的元素值

```
for (column = 0; column <=3; ++column) {
        a[2][column] = 0;
}
```

3. 计算双下标数组中所有元素值得总和



```
total = 0;

for (row = 0; row <= 2; ++row) {
    for (column = 0; column <= 3; ++column) {
        total += a[row][column];
    }
}
```

4. 对双下标数组的处理

问题描述：

	第1次	第2次	第3次	第4次
学生A	77	68	86	73
学生B	96	87	89	78
学生C	70	90	86	81

解决：（1）全体学生的最低分
（2）全体学生的最低分
（3）某个学生的平均分
（4）表格形式打印

4. 对双下标数组的处理

```
// Fig. 6.22: fig06_22.c
// Two-dimensional array manipulations.
#include <stdio.h>
#define STUDENTS 3
#define EXAMS 4

// 函数原型
int minimum(const int grades[][EXAMS], size_t pupils, size_t tests);
int maximum(const int grades[][EXAMS], size_t pupils, size_t tests);
double average(const int setOfGrades[], size_t tests);
void printArray(const int grades[][EXAMS], size_t pupils, size_t tests);
```

```
int main(void)
{
   // initialize student grades for three students (rows)三个学生成绩的初始化
   int studentGrades[STUDENTS][EXAMS] =
      { { 77, 68, 86, 73 },
        { 96, 87, 89, 78 },
        { 70, 90, 86, 81 } };

   // output array studentGrades
   puts("The array is:");
   printArray(studentGrades, STUDENTS, EXAMS);

   // determine smallest and largest grade values
   printf("\n\nLowest grade: %d\nHighest grade: %d\n",
      minimum(studentGrades, STUDENTS, EXAMS),
      maximum(studentGrades, STUDENTS, EXAMS));

   // calculate average grade for each student
   for (size_t student = 0; student < STUDENTS; ++student) {
      printf("The average grade for student %u is %.2f\n",
         student, average(studentGrades[student], EXAMS));
   }
}
```

```
// Find the minimum grade
int minimum(const int grades[][EXAMS], size_t pupils, size_t tests)
{
  int lowGrade = 100; // initialize to highest possible grade

  // loop through rows of grades
  for (size_t i = 0; i < pupils; ++i) {

    // loop through columns of grades
    for (size_t j = 0; j < tests; ++j) {

      if (grades[i][j] < lowGrade) {
        lowGrade = grades[i][j];
      }
    }
  }

  return lowGrade; // return minimum grade
}
```

```
// Find the maximum grade
int maximum(const int grades[][EXAMS], size_t pupils, size_t tests)
{
   int highGrade = 0; // initialize to lowest possible grade

   // loop through rows of grades
   for (size_t i = 0; i < pupils; ++i) {

      // loop through columns of grades
      for (size_t j = 0; j < tests; ++j) {

         if (grades[i][j] > highGrade) {
            highGrade = grades[i][j];
         }
      }
   }

   return highGrade; // return maximum grade
}
```

```
// Determine the average grade for a particular student
double average(const int setOfGrades[], size_t tests)
{
  int total = 0; // sum of test grades

  // total all grades for one student
  for (size_t i = 0; i < tests; ++i) {
    total += setOfGrades[i];
  }

  return (double) total / tests; // average
}
```

```
// Print the array
void printArray(const int grades[][EXAMS], size_t pupils, size_t tests)
{
   // output column heads
   printf("%s", "                  [0]  [1]  [2]  [3]");

   // output grades in tabular format
   for (size_t i = 0; i < pupils; ++i) {

      // output label for row
      printf("\nstudentGrades[%u] ", i);

      // output grades for one student
      for (size_t j = 0; j < tests; ++j) {
         printf("%-5d", grades[i][j]);
      }
   }
}
```

第6章 数组——可变长数组

如何在程序运行时动态地确定数组的大小呢?

以前程序员只能使用**动态分配内存**技术。

现在为了应对在编译时数组大小无法确定的情况，C提供了“可变长数组（VLA）”——数组的长度以表达式的形式表示，而表达式的值要在运行时才能确定。

在C99中使用可变长数组

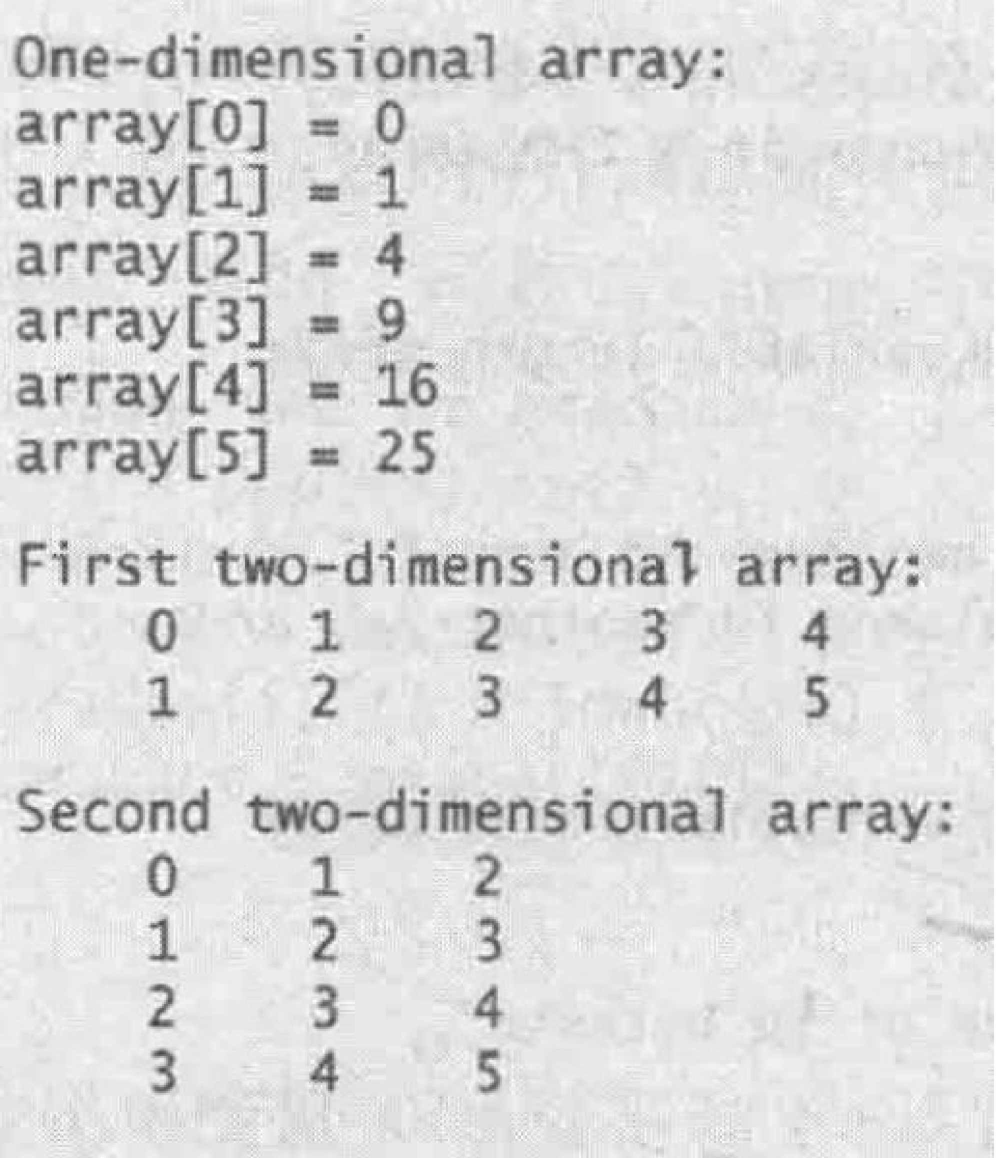
```
#include <stdio.h>

// 函数原型
void print1DArray(size_t size, int array[size]);
void print2DArray(size_t row, size_t col, int array[row][col]);

int main(void)
{
   printf("%s", "Enter size of a one-dimensional array: ");
   int arraySize; // size of 1-D array
   scanf("%d", &arraySize);

   int array[arraySize]; // declare 1-D variable-length array

   printf("%s", "Enter number of rows and columns in a 2-D array: ");
   int row1, col1;    // number of rows and columns in a 2-D array
   scanf("%d %d", &row1, &col1);
```

```
int array2D1[row1][col1];   // declare 2-D variable-length array

 printf("%s",
   "Enter number of rows and columns in another 2-D array: ");
 int row2, col2; // number of rows and columns in another 2-D array
 scanf("%d %d", &row2, &col2);

 int array2D2[row2][col2];   // declare 2-D variable-length array

 // test sizeof operator on VLA
 printf("\nsizeof(array) yields array size of %d bytes\n",
   sizeof(array));
```

```
// assign elements of 1-D VLA
for (size_t i = 0; i < arraySize; ++i) {
   array[i] = i * i;
}

// assign elements of first 2-D VLA
for (size_t i = 0; i < row1; ++i) {
   for (size_t j = 0; j < col1; ++j) {
      array2D1[i][j] = i + j;
   }
}

// assign elements of second 2-D VLA
for (size_t i = 0; i < row2; ++i) {
   for (size_t j = 0; j < col2; ++j) {
      array2D2[i][j] = i + j;
   }
}
```

```
   puts("\nOne-dimensional array:");
   print1DArray(arraySize, array); // pass 1-D VLA to function

   puts("\nFirst two-dimensional array:");
   print2DArray(row1, col1, array2D1); // pass 2-D VLA to function

   puts("\nSecond two-dimensional array:");
   print2DArray(row2, col2, array2D2); // pass other 2-D VLA to function
}
```

```
void print1DArray(size_t size, int array[size])
{
  // output contents of array
  for (size_t i = 0; i < size; i++) {
    printf("array[%d] = %d\n", i, array[i]);
  }
}

void print2DArray(size_t row, size_t col, int array[row][col])
{
  // output contents of array
  for (size_t i = 0; i < row; ++i) {
    for (size_t j = 0; j < col; ++j) {
      printf("%5d", array[i][j]);
    }

    puts("");
  }
}
```



```
One-dimensional array:
array[0] = 0
array[1] = 1
array[2] = 4
array[3] = 9
array[4] = 16
array[5] = 25

First two-dimensional array:
    0    1    2    3    4
    1    2    3    4    5

Second two-dimensional array:
    0    1    2
    1    2    3
    2    3    4
    3    4    5
```

Questions & Answers