Arrays

Objectives

- One-Dimensional Arrays
- Array Initialization
- Arrays as Function Arguments
- Case Study: Computing Averages and Standard Deviations
- Two-Dimensional Arrays
- Common Programming and Compiler Errors

Introduction (continued)

- One of the simplest data structures, called an **array**, is used to store and process a set of values, all of the same data type, that forms a logical group

Introduction (continued)

Grades	Codes	Prices
98	x	10.96
87	a	6.43
92	m	2.58
79	n	.86
85		12.27
		6.39

Figure 8.1 Three lists of items

One-Dimensional Arrays

- A **one-dimensional array**, also called a **single-dimensional array** and a **single-subscript array**, is a list of values of the same data type that is stored using a single group name

One-Dimensional Arrays (continued)

- To create a one-dimensional array:
  - `#define NUMELS 5`
  - `int grades[NUMELS];`
- In C, the starting index value for all arrays is 0
- Each item in an array is called an **element** or **component** of the array
- Any element can be accessed by giving the name of the array and the element's position
  - The position is the element's **index** or **subscript**
  - Each element is called an **indexed variable** or a **subscripted variable**

One-Dimensional Arrays (continued)

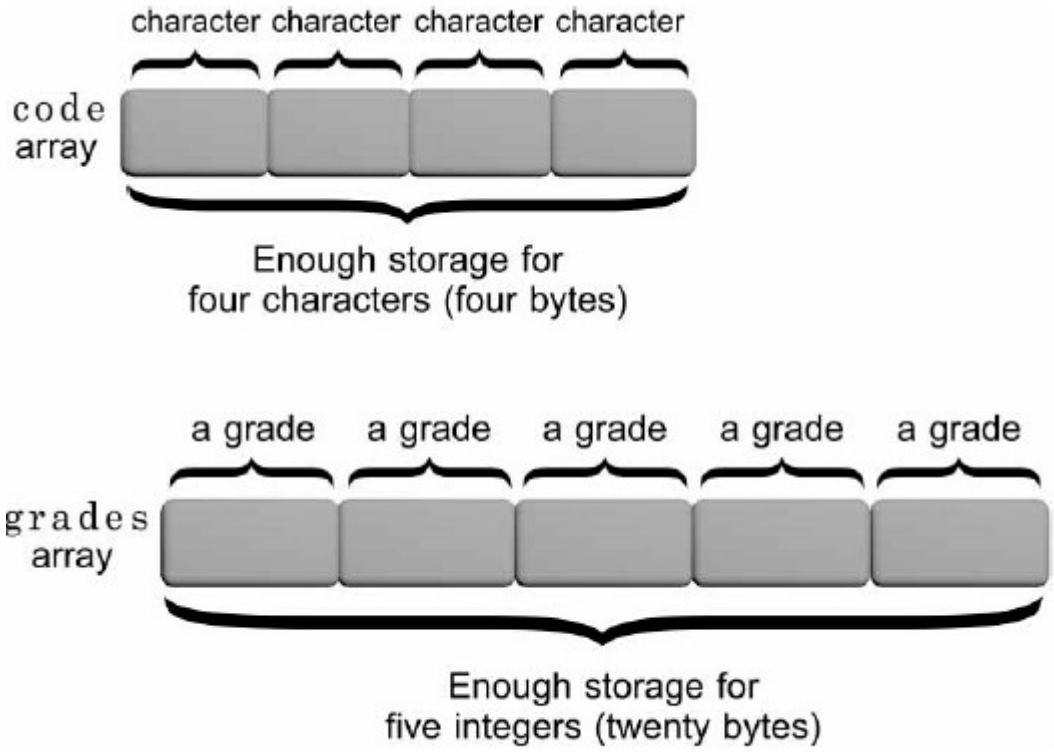


Figure 8.3 The `code` and `grades` arrays in memory

One-Dimensional Arrays (continued)

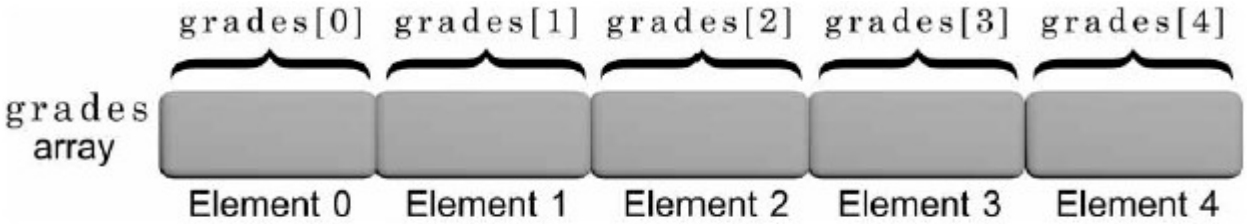


Figure 8.4 Identifying individual array elements

One-Dimensional Arrays (continued)

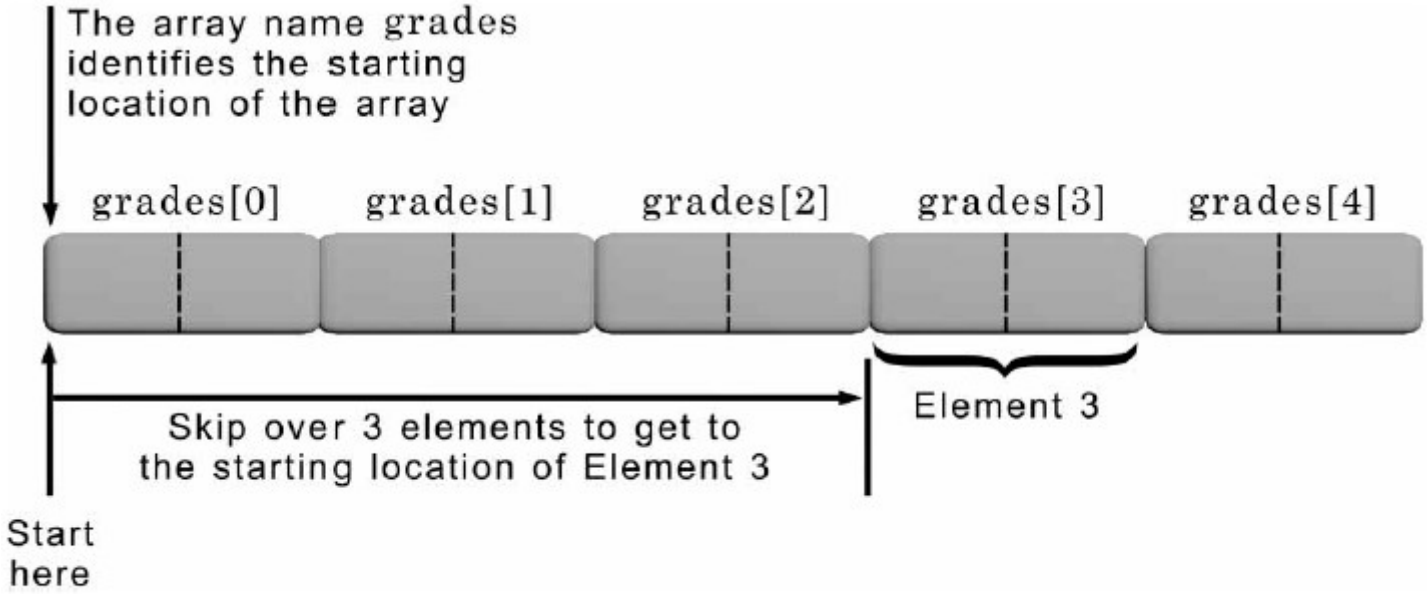


Figure 8.5 Accessing element 3

One-Dimensional Arrays (continued)

- Subscripted variables can be used anywhere scalar variables are valid
  - `grades[0] = 98;`
  - `grades[1] = grades[0] - 11;`
- Any expression that evaluates an integer may be used as a subscript

```
#define NUMELS 5
total = 0; /* initialize total to zero */
for (i = 0; i < NUMELS; i++)
  total = total + grades[i]; /* add a grade */
```

Input and Output of Array Values

- Individual array elements can be assigned values using individual assignment statements or, interactively, using the `scanf()` function

```
#define NUMELS 5
for(i = 0; i < NUMELS; i++)
{
  printf("Enter a grade: ");
  scanf("%d", &grades[i]);
}
```

- Be careful: C does not check the value of the index being used (called a **bounds check)**

Input and Output of Array Values (continued)

Program 8.1

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    #define MAXGRADES 5
 5    int grades[MAXGRADES];
 6    int i;
 7
 8    /* input the grades */
 9    for (i = 0; i < MAXGRADES; i++)
10    {
11      printf("Enter a grade: ");
12      scanf("%d", &grades[i]);
13    }
14
15    /* display the grades */
16    for (i = 0; i < MAXGRADES; i++)
17      printf("grades %d is %d\n", i, grades[i]);
18
19    return 0;
20  }
```

Sample output:

```
Enter a grade: 85
Enter a grade: 90
Enter a grade: 78
Enter a grade: 75
Enter a grade: 92
grades 0 is 85
grades 1 is 90
grades 2 is 78
grades 3 is 75
grades 4 is 92
```

Input and Output of Array Values (continued)

Program 8.2

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    #define MAXGRADES 5
 5    int grades[MAXGRADES];
 6    int i, total = 0;
 7
 8    /* input the grades */
 9    for (i = 0; i < MAXGRADES; i++)    {
10      printf("Enter a grade: ");
11      scanf("%d", &grades[i]);
12    }
13
14    /* display and total the grades */
15    printf("\nThe total of the grades ");
16    for (i = 0; i < MAXGRADES; i++)
17    {
18      printf("%d ", grades[i]);
19        total += grades[i];
20    }
21
22    printf("is %d\n", total);   /* display the total */
23
24    return 0;
25  }
```

Statement is outside of the second `for` loop; `total` is displayed only once, after all values have been added

Array Initialization

- The individual elements of all global and `static` arrays (local or global) are, by default, set to 0 at compilation time
- The values within `auto` local arrays are undefined
- Examples of initializations:
  - `int grades[5] = {98, 87, 92, 79, 85};`
  - `double length[7] = {8.8, 6.4, 4.9, 11.2};`
  - `char codes[6] = {'s', 'a', 'm', 'p', 'l', e'};`
  - `char codes[] = {'s', 'a', 'm', 'p', 'l', 'e'};`
  - `char codes[] = "sample"; /* size is 7 */`

Array Initialization (continued)

```
1 #define SIZE1 20
2 #define SIZE2 25
3 #define SIZE3 15
4
5 int gallons[SIZE1]; /* a global array */
6 static int dist[SIZE2]; /* a static global array */
7
8 int main()
9 {
10   int miles[SIZE3]; /* an auto local array */
11   static int course[SIZE3]; /* static local array */
12     .
13     .
14   return 0;
15 }
```

Array Initialization (continued)

- The **NULL** character, which is the escape sequence \0, is automatically appended to all strings by the C compiler

Array Initialization (continued)

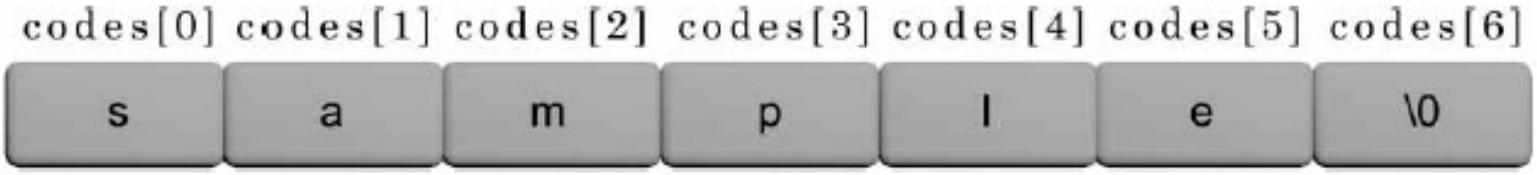


Figure 8.6 A string is terminated with a special sentinel

Array Initialization (continued)

Program 8.3

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    #define MAXELS 5
 5    int nums[MAXELS] = {2, 18, 1, 27, 16};
 6    int i, max;
 7
 8    max = nums[0];
 9
10    for (i = 1; i < MAXELS; i++)
11       if (max < nums[i])
12         max = nums[i];
13
14    printf("The maximum value is %d\n", max);
15
16    return 0;
17  }
```

Arrays as Function Arguments

- Individual array elements are passed to a function by including them as subscripted variables in the function call argument list
  - `findMin(grades[2], grades[6]);`
  - Pass by value
- When passing a complete array to a function, the called function receives access to the actual array, rather than a copy of the values in the array
  - `findMax(grades);`
  - Pass by reference

Arrays as Function Arguments (continued)

Program 8.4

```
 1  #include <stdio.h>
 2  #define MAXELS 5
 3
 4  void findMax(int [MAXELS]); /* function prototype */
 5
 6  int main()
 7  {
 8    int nums[MAXELS] = {2, 18, 1, 27, 16};
 9
10    findMax(nums);
11
12    return 0;
13  }
14
15  void findMax(int vals[MAXELS]) /* find the maximum value */
16  {
17    int i, max = vals[0];
18
19    for (i = 1; i < MAXELS; i++)
20      if (max < vals[i])
21        max = vals[i];
22
23    printf("The maximum value is %d\n", max);
24  }
```

Size can be omitted

Arrays as Function Arguments (continued)

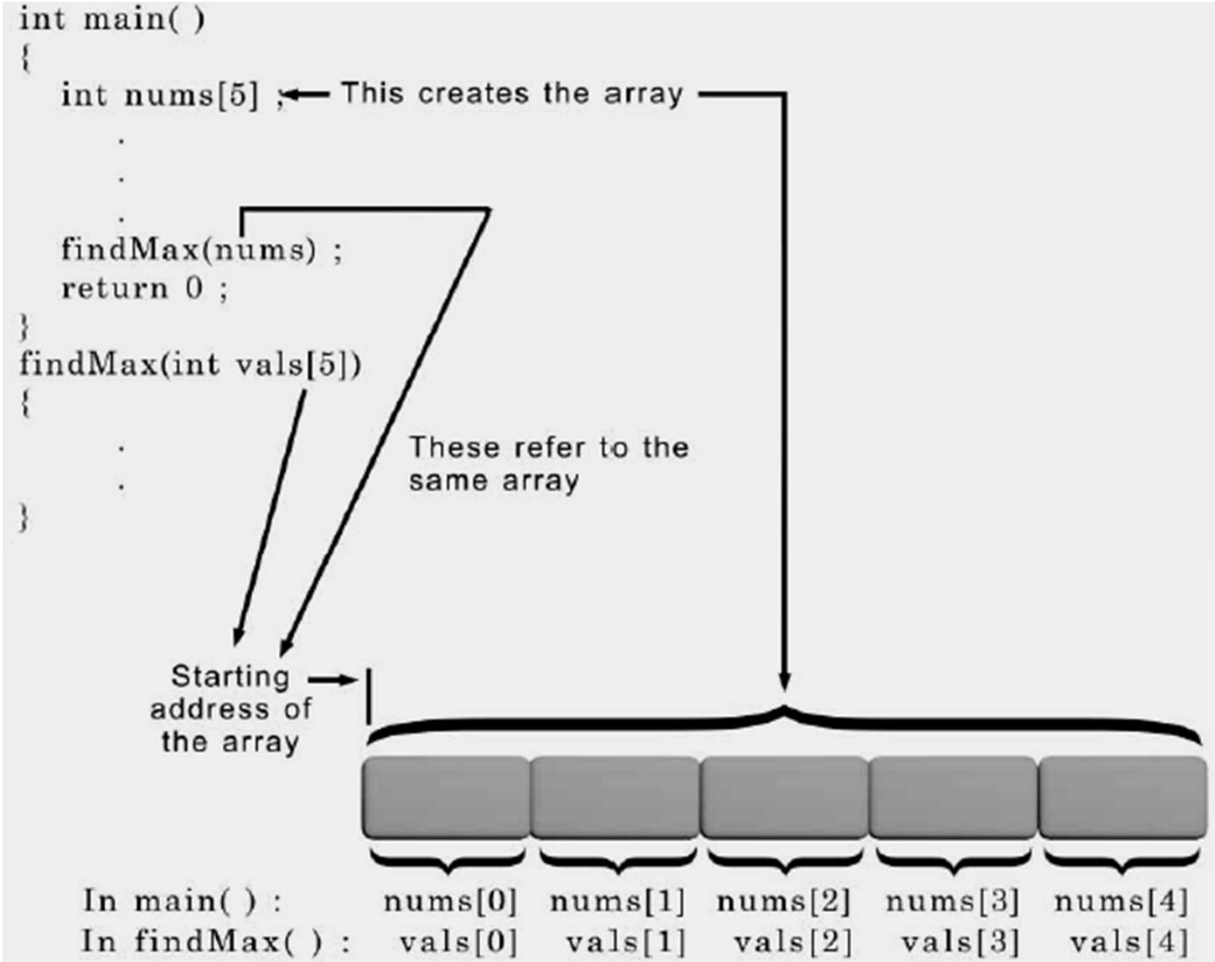


Figure 8.7 Only one array is created

Arrays as Function Arguments (continued)

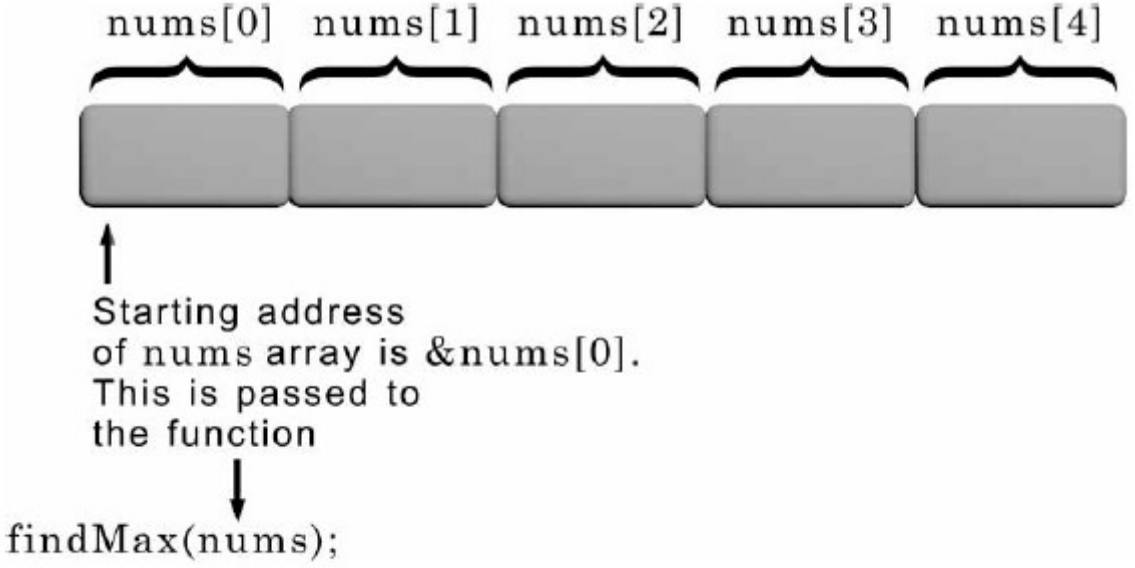


Figure 8.8 The starting address of the array is passed

Arrays as Function Arguments (continued)

Program 8.5

```
 1  #include <stdio.h>
 2  int findMax(int [], int); /* function prototype */
 3
 4  int main()
 5  {
 6    #define MAXELS 5
 7    int nums[MAXELS] = {2, 18, 1, 27, 16};
 8
 9    printf("The maximum value is %d\n", findMax(nums, MAXELS));
10
11    return 0;
12  }
13
14  int findMax(int vals[], int numels)
15  {
16    int i, max = vals[0];
17
18    for (i = 1; i < numels; i++)
19      if (max < vals[i])
20        max = vals[i];
21
22    return (max);
23  }
```

6.1.4 APPLYING EXAMPLES

【ex. 6.1】Compute average and give the level with letter

```
#define N  60
main()
{   int  score[N],  i;
    float ave=0;       char grade;
    for(i=0;i<N;i++)
      { scanf("%d",&score[i]);
            ave=ave+score[i];
          }
     ave=ave/N;
     for(i=0;i<N;i++)
        {    if(score[i]>=ave+10) grade='A'
             else if (score[i]>=ave-10) grade='B'
                    else grade='C';
             printf("%5d:%d%3c\n",i+1,score[i],grade);
      } }
```

Select Sorting

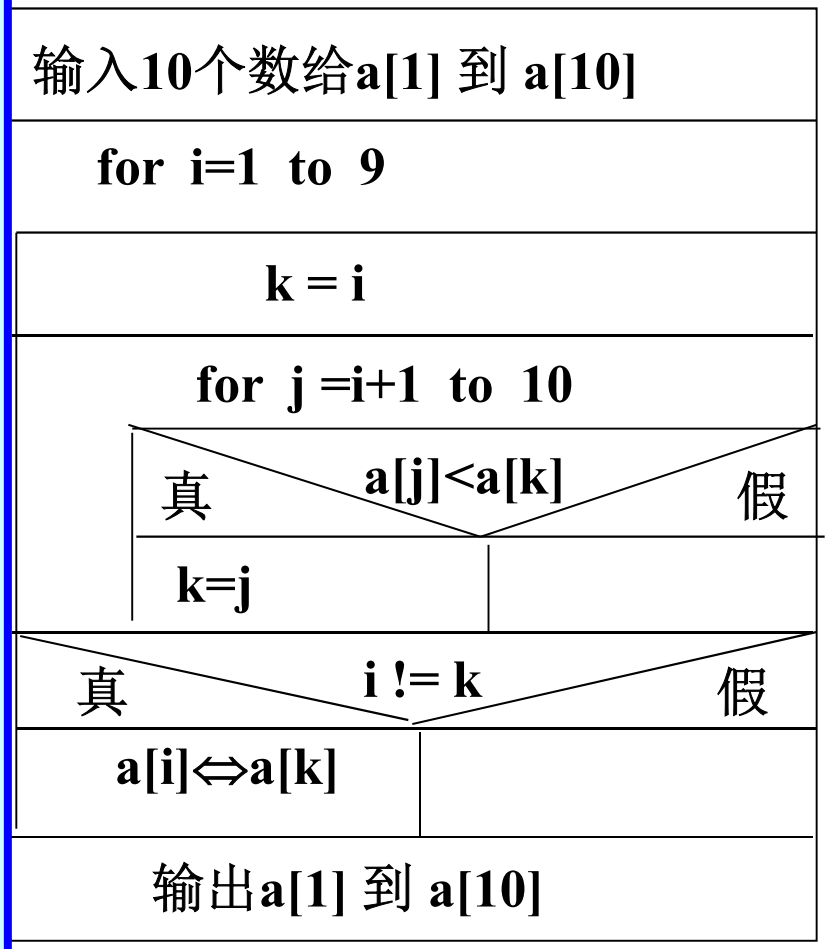
过程：

（1）首先通过9次比较，从10个数中找出最小的， 将它与第1个数交换—第一趟选择排序，结果最小的数被安置在第1个元素位置上

（2）再通过8次比较，从剩余的9个数中找出次小的数，将它与第2个数交换—第二趟选择排序

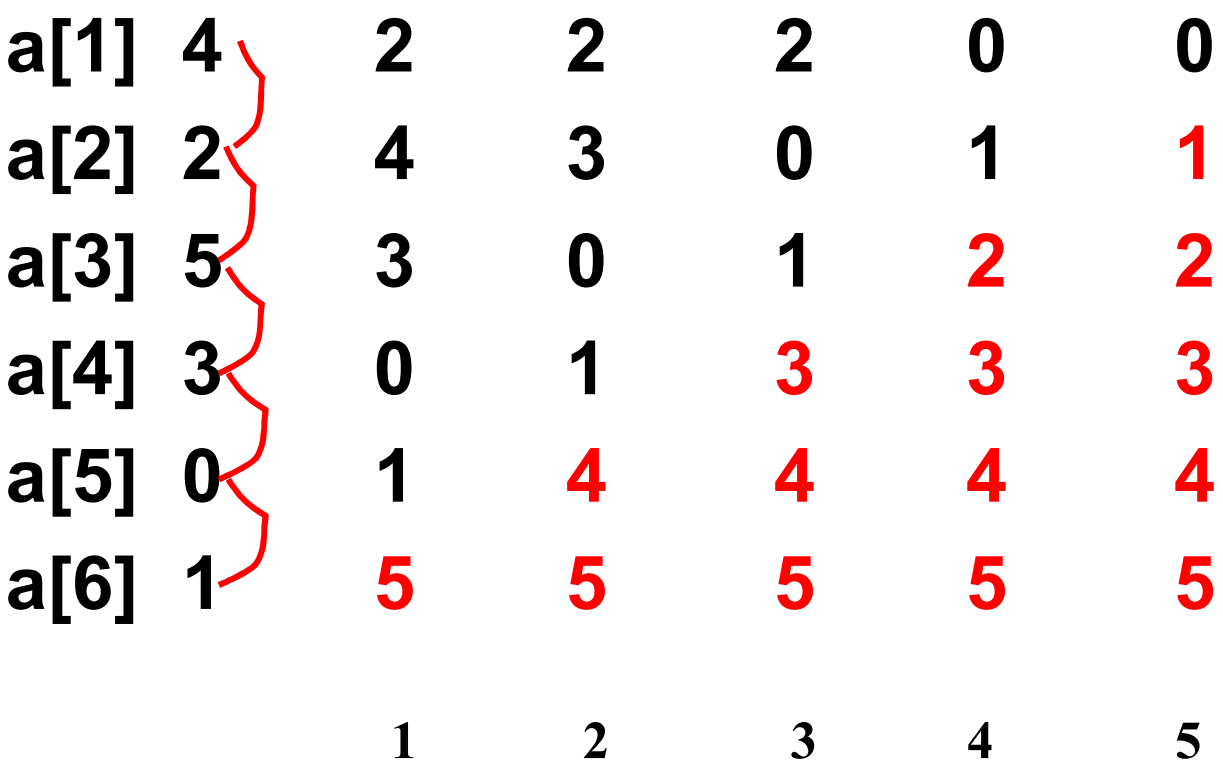
（3）重复上述过程，共经过9趟排序后，排序结束

```
#include <stdio.h>
void main()
{   int a[11],i,j,k,x;
    printf("Input 10 numbers:\n");
    for(i=1;i<=10;i++)
       scanf("%d",&a[i]);
    printf("\n");
   for(i=1;i<=9;i++)
   {  k=i;
      for(j=i+1;j<=10;j++)
            if(a[j]<a[k])  k=j;
      if(i!=k)
      {  x=a[i]; a[i]=a[k]; a[k]=x;}
   }
    printf("The sorted numbers:\n");
    for(i=1;i<=10;i++)
          printf("%d ",a[i]);
}
```

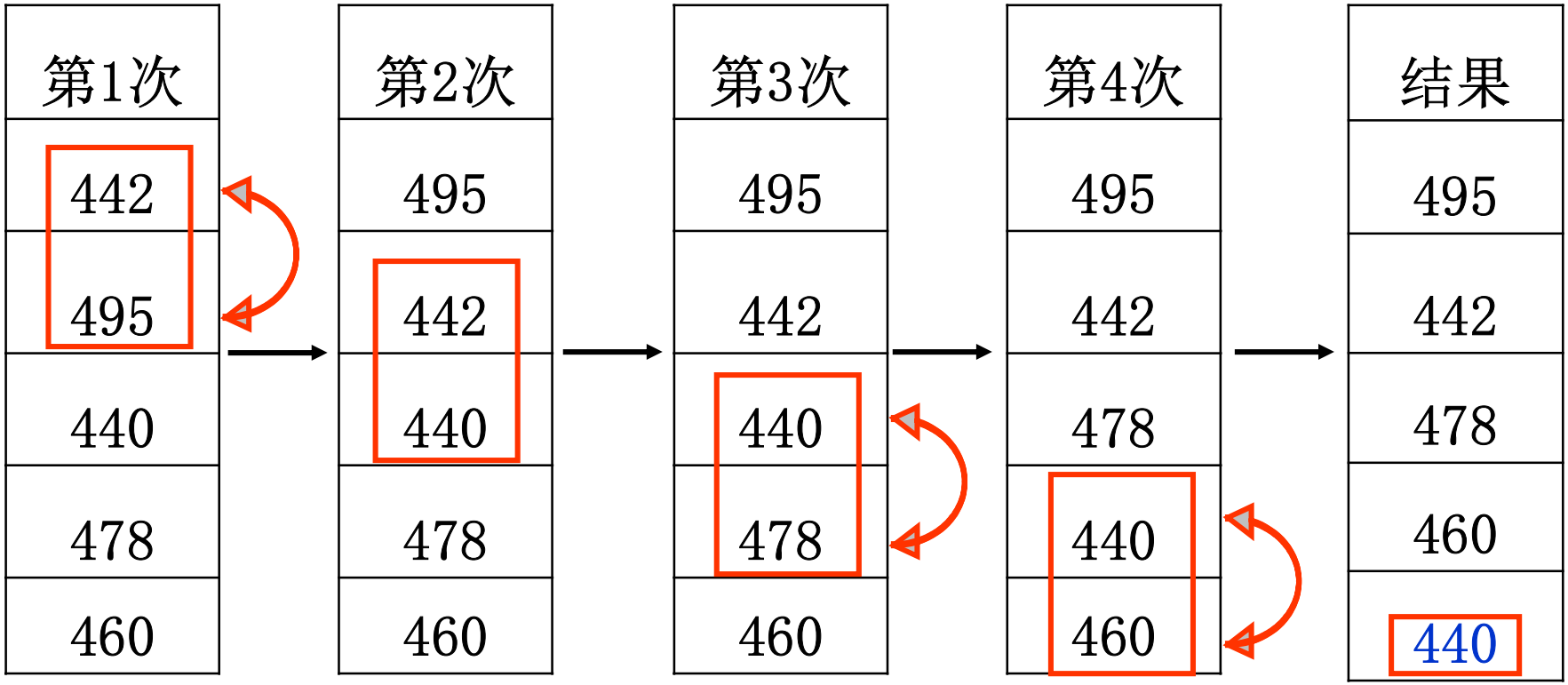


【ex. 6.3 】Input 10 numbers，Output them in decrease sort

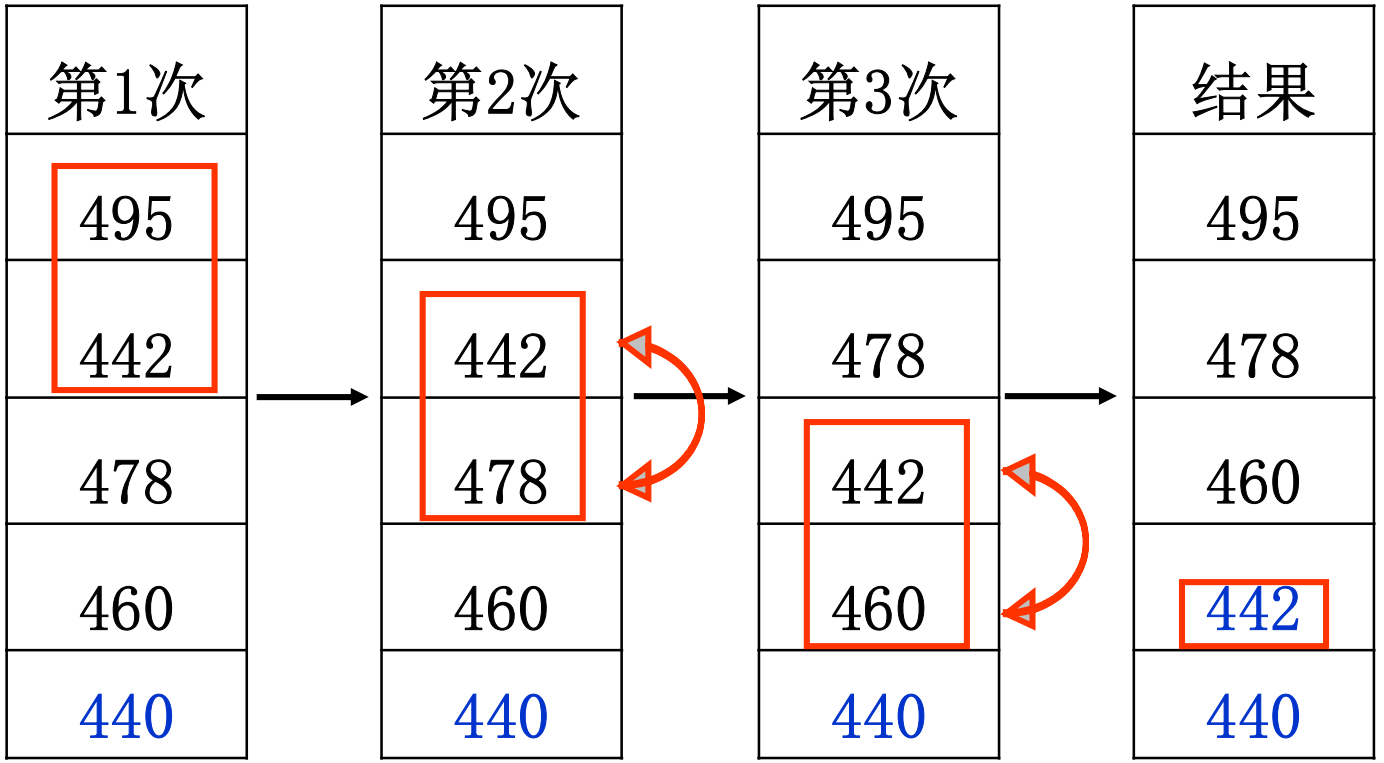
Method 1：bubble



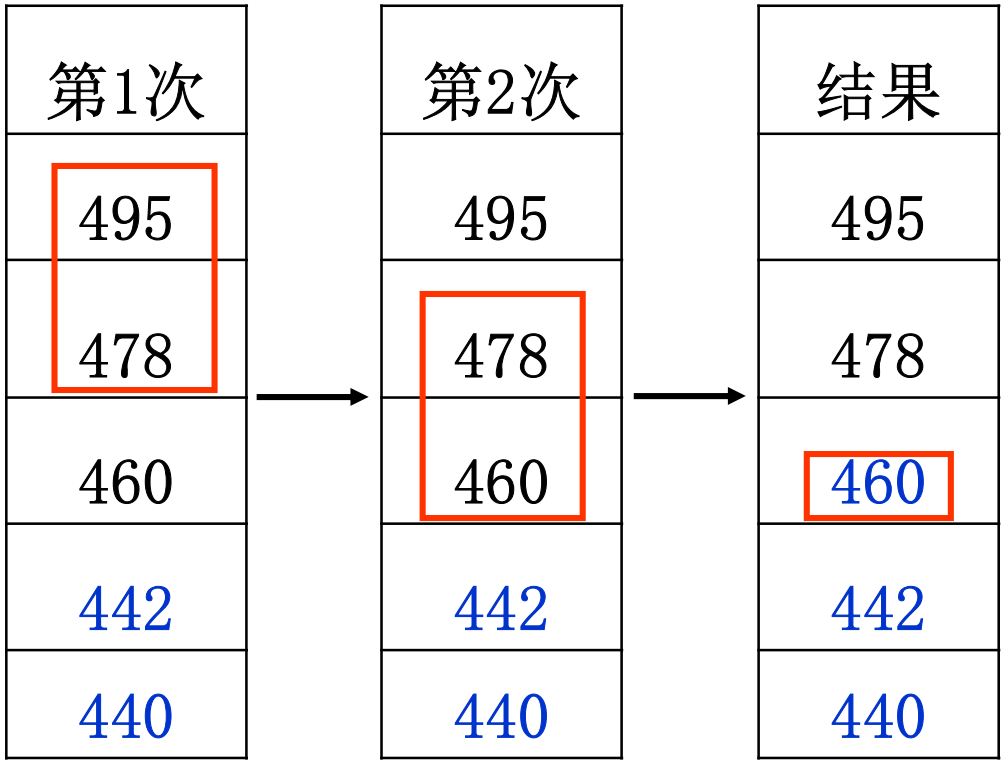
- 冒泡法第一轮：
  - 使最小的数放在最后一个位置上



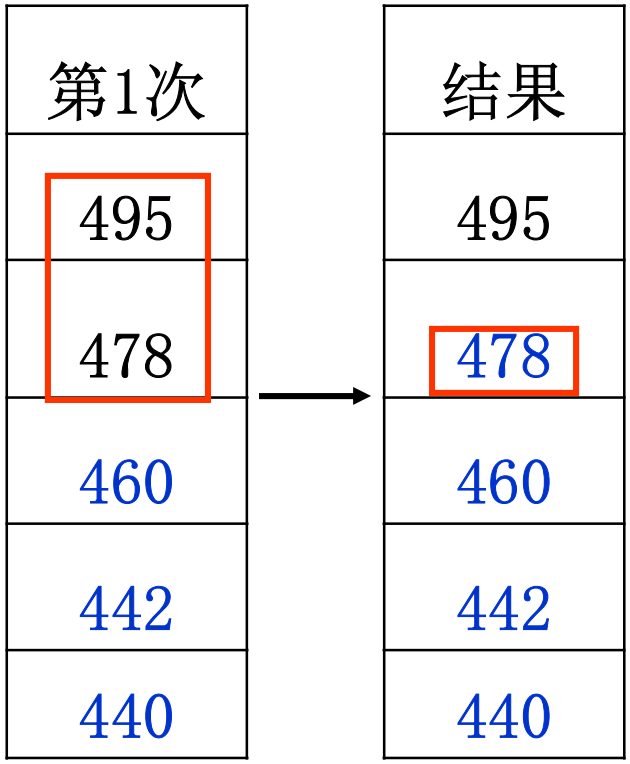
- 冒泡法第二轮：
  - 使次小的数放在倒数第二个位置



- 第三轮排序：


第1次
495
478
460
442
440
第2次
495
478
460
442
440
结果
495
478
460
442
440

- 第四轮排序：


第1次
495
478
460
442
440
结果
495
478
460
442
440

String-handling Function in the Standard Library

1. gets()

call form：gets(string variable)

function：read a string from keyboard （may include space）.

2. puts()

call form：puts(string variable)

3. strcmp()

call form：strcmp(string1 ,string2)

“string”may be string constant or string variable.

Two strings are passed as arguments.An integer is returned that is less than,equal to,or greater than zero,depending on whether s1 is lexicographically less than,equal to,or greater than s2.

4．strcpy()

call form：strcpy(string variable, string)

The character in the string s2 are copied into s1

5．strcat()

call form：strcat(string variable, string)

This function takes two strings as arguments,concatenates them,and puts the result in s1.The programmer must ensure that s1 points to enough space to hold the result. The string s1 is returned.

6．strlen()

call form：strlen(string)

A count of the number of charaters before ‘\0’ is returned.

7．strlwr()

call form：strlwr(string)

function：change uppercase letter in string to lowercase letter.

8．strupr()

call form：strupr(string)

function：change lowercase letter in string to uppercase letter.

Two-Dimensional Arrays

- A two-dimensional array, or table, consists of both rows and columns of elements

```
int val[3][4];
```

Two-Dimensional Arrays (continued)

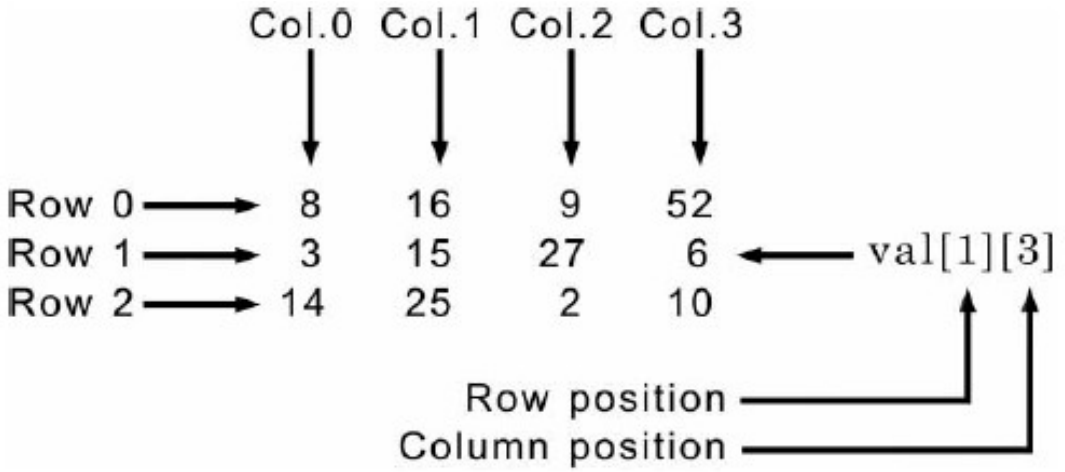


Figure 8.9 Each array element is identified by its row and column

Two-Dimensional Arrays (continued)

- Initialization:

```
#define NUMROWS 3
#define NUMCOLS 4
int val[NUMROWS][NUMCOLS] = { {8,16,9,52},
                              {3,15,27,6},
                              {14,25,2,10} };
```

- The inner braces can be omitted:

```
int val[NUMROWS][NUMCOLS] = {8,16,9,52,3,15,27,
  6,14,25,2,10};
```

- Initialization is done in row order

Two-Dimensional Arrays (continued)

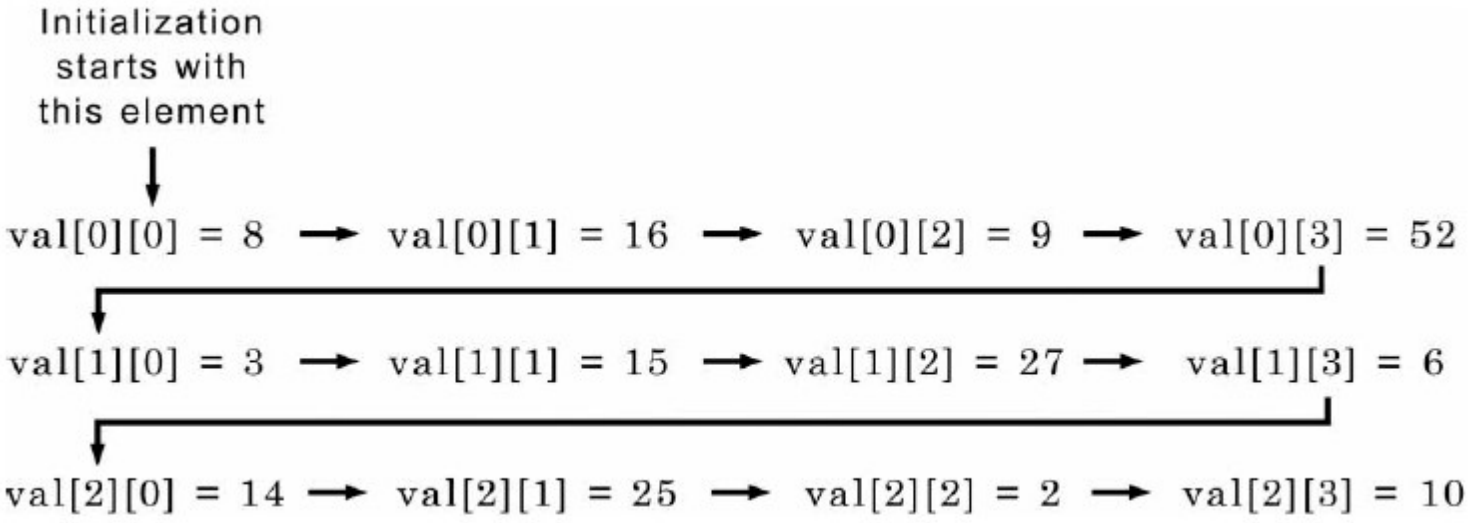


Figure 8.10 Storage and initialization of the `val[]` array

Two-Dimensional Arrays (continued)

Program 8.7

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    #define NUMROWS 3
 5    #define NUMCOLS 4
 6    int val[NUMROWS][NUMCOLS] = {8,16,9,52,3,15,27,6,14,25,2,10};
 7    int i, j;
 8
 9    printf("\nDisplay of val array by explicit element");
10    printf("\n%2d %2d %2d %2d",
11           val[0][0],val[0][1],val[0][2],val[0][3]);
12    printf("\n%2d %2d %2d %2d",
13           val[1][0],val[1][1],val[1][2],val[1][3]);
14    printf("\n%2d %2d %2d %2d",
15           val[2][0],val[2][1],val[2][2],val[2][3]);
16
17    printf("\n\nDisplay of val array using a nested for loop");
18    for (i = 0; i < NUMROWS; i++)
19    {
20      printf("\n"); /* start a new line for each row */
21      for (j = 0; j < NUMCOLS; j++)
22        printf("%2d ", val[i][j]);
23    }
24    printf("\n");
25
26    return 0;
27  }
```

Two-Dimensional Arrays (continued)

- The display produced by Program 8.7 is

```
Display of val array by explicit element
8 16 9 52
3 15 27 6
14 25 2 10

Display of val array using a nested for loop
8 16 9 52
3 15 27 6
14 25 2 10
```

Two-Dimensional Arrays (continued)

Program 8.8

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4    #define NUMROWS 3
 5    #define NUMCOLS 4
 6    int val[NUMROWS][NUMCOLS] = {8,16,9,52,3,15,27,6,14,25,2,10};
 7    int i, j;
 8
 9    /* multiply each element by 10 and display it */
10    printf("\nDisplay of multiplied elements\n");
11    for (i = 0; i < NUMROWS; i++)
12    {
13      printf("\n"); /* start a new line */
14      for (j = 0; j < NUMCOLS; ++j)
15      {
16        val[i][j] = val[i][j] * 10;
17        printf("%3d ", val[i][j]);
18      } /* end of inner loop */
19    } /* end of outer loop */
20    printf("\n");
21
22    return 0;
23  }
```

Two-Dimensional Arrays (continued)

Program 8.9

```
 1  #include <stdio.h>
 2  #define ROWS 3
 3  #define COLS 4
 4
 5  void display(int [ROWS][COLS]); /* function prototype */
 6
 7  int main()
 8  {
 9    int val[ROWS][COLS] = {8,16,9,52,
10                           3,15,27,6,
11                           14,25,2,10};
12
13    display(val);
14
15    return 0;
16  }
17
18  void display(int nums[ROWS][COLS])
19  {
20    int rowNum, colNum;
21
22    for (rowNum = 0; rowNum < ROWS; rowNum++)
23    {
24      for(colNum = 0; colNum < COLS; colNum++)
25        printf("%4d",nums[rowNum][colNum]);
26      printf("\n");
27    }
28  }
```

Row size can be omitted

Two-Dimensional Arrays (continued)

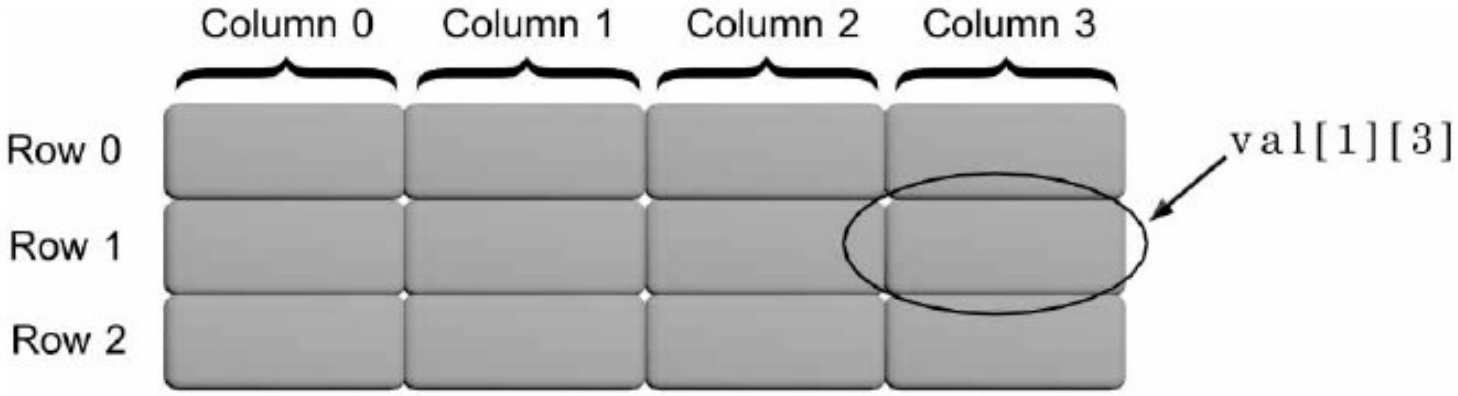


Figure 8.11 Storage of the `val` array

Two-Dimensional Arrays (continued)

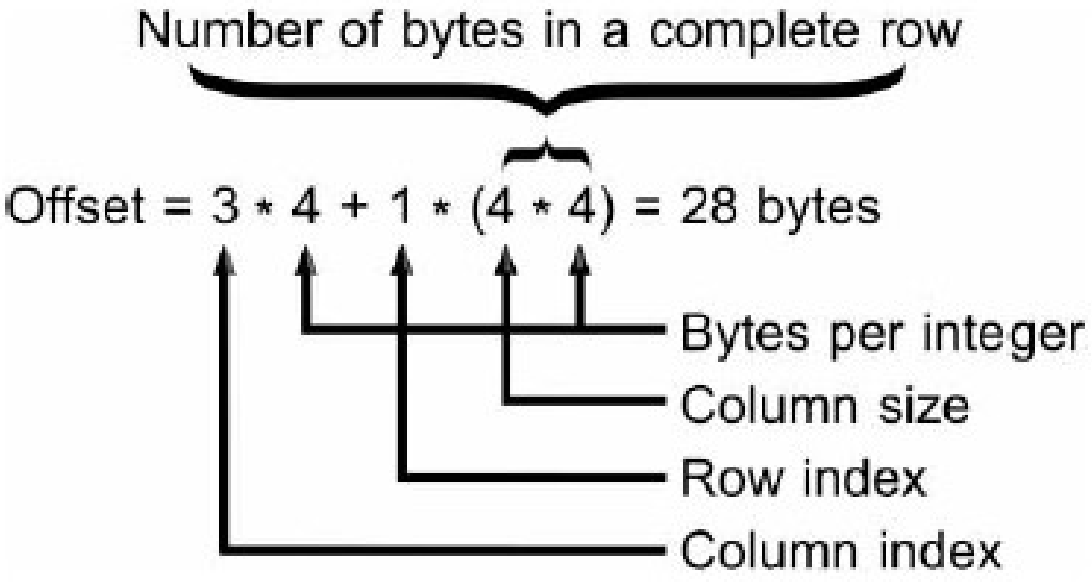


Figure 8.12 Determining an element's offset

【ex. 6.5 】

```
#define Row 2
#define Col 3
main()
    { int i, j, array[Row][Col];
      for(i=0; i<Row; i++)
          for(j=0; j<Col; j++)
            {printf("please input array[%2d][%2d]:",i,j);
              scanf("%d",&array[i][j]);
             }
       printf("\n");
       /*输出2维数组array*/
       for(i=0;i<Row;i++)
          { for(j=0;j<Col;j++)
                printf("%d\t", array[i][j]);
           printf("\n");
          }
       getch();
    }
```

6.2.3 INITIALIZATIONS OF TWO-DIMENSIONAL ARRAYS

1. Assignment by line

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

   {1,2,3,4} a[0][0]~a[0][3]

   {5,6,7,8} a[1][0]~a[1][3]

   {9,10,11,12} a[2][0]~a[2][3]

2. Assignment by sequence in storage

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

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

   ex.: int a[3][4]={{1},{5},{9}};

   int a[3][4]={{1},{0,6},{0,0,11}};

【ex. 6.7】There are M students, N courses.Design a program to give the average of every one and the average of every course.

	col 0	1	2	3	4 N
row0	78	85	83	65	0
1	88	91	89	93	0
2	72	65	54	75	0
3	86	88	75	60	0
4	69	60	50	72	0
M 5	0	0	0	0	0

```
for(i=0;i<M;i++)
    { for(j=0;j<N;j++)
          score[i][N] += score[i][j];
      score[i][N] /= N;
    }

for(j=0;j<N;j++)
    { for(i=0;i<M;i++)
          score[M][j] += score[i][j];
      score[M][j] /=M;
    }
```

```
#define M  5
#define N  4
main()
{ int i,j;
   float  score[M+1][N+1]={{78,85,83,65}, {88,91,89,93},
                          {72,65,54,75},{86,88,75,60},  {69,60,50,72}};
```

Larger Dimensional Arrays

- A three-dimensional array can be viewed as a book of data tables (the third subscript is called the **rank**)
  - `int response[4][10][6];`
- A four-dimensional array can be represented as a shelf of books where the fourth dimension is used to declare a desired book on the shelf
- A five-dimensional array can be viewed as a bookcase filled with books where the fifth dimension refers to a selected shelf in the bookcase
- Arrays of three, four, five, six, or more dimensions can be viewed as mathematical *n*-tuples

Common Programming Errors

- Forgetting to declare the array
- Using a subscript that references a nonexistent array element
- Not using a large enough conditional value in a `for` loop counter to cycle through all the array elements
- Forgetting to initialize the array

Common Compiler Errors

Error	Typical Unix-based Compiler Error Message	Typical Windows-based Compiler Error Message
Designating a variable as an `extern` in one file, without declaring the variable as a `global` in another file	`ERROR: Undefined symbol: ex` (Note: use the `-bloadmap` or `–bnoquiet` option to obtain more information about the error.)	`Link error: unresolved external symbol …`
Applying the indirection operator to a nonpointer variable	`(S) Operand of indirection operator must be a pointer expression.`	`error: illegal indirection`
Not passing an address in a call to a function whose parameter is declared as a pointer	`(W) Function argument assignment between types "int*" and "int" is not allowed.`	`error: function cannot convert parameter from dataType to dataType*`
Assigning a value, rather than an address, to a pointer	`(W) Operation between types "int*" and "int" is not allowed.`	`error: cannot convert parameter from dataType to dataType*`
Attempting to take the address of a constant	`(W) Operation between types "int" and "const int*" is not allowed.`	`error: & on constant`
Attempting to use a variable that is not within scope	`(S) Undeclared identifier…`	`error: undeclared identifier …`

Summary

- A single-dimensional array is a data structure that can store a list of values of the same data type
- Elements are stored in contiguous locations
  - Referenced using the array name and a subscript
- Single-dimensional arrays may be initialized when they are declared
- Single-dimensional arrays are passed to a function by passing the name of the array as an argument

Summary (continued)

- A two-dimensional array is declared by listing both a row and a column size with the data type and name of the array
- Two-dimensional arrays may be initialized when they are declared
- Two-dimensional arrays are passed to a function by passing the name of the array as an argument