

02_06_C Arrays

Objectives

In this chapter, you'll:

- Use the array data structure to represent lists and tables of values.
- Define an array, initialize an array and refer to individual elements of an array.
- Define symbolic constants.
- Pass arrays to functions.
- Use arrays to store, sort and search lists and tables of values.
- Define and manipulate multidimensional arrays.
- Create variable-length arrays whose size is determined at execution time.
- Understand security issues related to input with `scanf`, output with `printf` and arrays.

Operators	Associativity	Type
[] () ++ (postfix) -- (postfix)	left to right	highest
+ - ! ++ (prefix) -- (prefix) (type)	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
&&	left to right	logical AND
	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment
,	left to right	comma

Fig. 6.2 | Operator precedence and associativity.

```
1 // Fig. 6.3: fig06_03.c
2 // Initializing the elements of an array to zeros.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main(void)
7 {
8     int n[5]; // n is an array of five integers
9
10    // set elements of array n to 0
11    for (size_t i = 0; i < 5; ++i) {
12        n[i] = 0; // set element at location i to 0
13    }
14
15    printf("%s%13s\n", "Element", "Value");
16
17    // output contents of array n in tabular format
18    for (size_t i = 0; i < 5; ++i) {
19        printf("%7u%13d\n", i, n[i]);
20    }
21 }
```

Fig. 6.3 | Initializing the elements of an array to zeros. (Part I of 2.)

Element	Value
0	0
1	0
2	0
3	0
4	0

Fig. 6.3 | Initializing the elements of an array to zeros. (Part 2 of 2.)

```
1 // Fig. 6.4: fig06_04.c
2 // Initializing the elements of an array with an initializer list.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main(void)
7 {
8     // use initializer list to initialize array n
9     int n[5] = {32, 27, 64, 18, 95};
10
11    printf("%s%13s\n", "Element", "Value");
12
13    // output contents of array in tabular format
14    for (size_t i = 0; i < 5; ++i) {
15        printf("%7u%13d\n", i, n[i]);
16    }
17 }
```

Fig. 6.4 | Initializing the elements of an array with an initializer list. (Part I of 2.)

Element	Value
0	32
1	27
2	64
3	18
4	95

Fig. 6.4 | Initializing the elements of an array with an initializer list. (Part 2 of 2.)

```
1 // Fig. 6.5: fig06_05.c
2 // Initializing the elements of array s to the even integers from 2 to 10.
3 #include <stdio.h>
4 #define SIZE 5 // maximum size of array
5
6 // function main begins program execution
7 int main(void)
8 {
9     // symbolic constant SIZE can be used to specify array size
10    int s[SIZE]; // array s has SIZE elements
11
12    for (size_t j = 0; j < SIZE; ++j) { // set the values
13        s[j] = 2 + 2 * j;
14    }
15
16    printf("%s%13s\n", "Element", "Value");
17
18    // output contents of array s in tabular format
19    for (size_t j = 0; j < SIZE; ++j) {
20        printf("%7u%13d\n", j, s[j]);
21    }
22 }
```

Fig. 6.5 | Initializing the elements of array s to the even integers from 2 to 10. (Part 1 of 2.)

Element	Value
0	2
1	4
2	6
3	8
4	10

Fig. 6.5 | Initializing the elements of array s to the even integers from 2 to 10. (Part 2 of 2.)

```
1 // Fig. 6.6: fig06_06.c
2 // Computing the sum of the elements of an array.
3 #include <stdio.h>
4 #define SIZE 12
5
6 // function main begins program execution
7 int main(void)
8 {
9     // use an initializer list to initialize the array
10    int a[SIZE] = {1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45};
11    int total = 0; // sum of array
12
13    // sum contents of array a
14    for (size_t i = 0; i < SIZE; ++i) {
15        total += a[i];
16    }
17
18    printf("Total of array element values is %d\n", total);
19 }
```

Total of array element values is 383

Fig. 6.6 | Computing the sum of the elements of an array.

```
1 // Fig. 6.7: fig06_07.c
2 // Analyzing a student poll.
3 #include <stdio.h>
4 #define RESPONSES_SIZE 40 // define array sizes
5 #define FREQUENCY_SIZE 11
6
7 // function main begins program execution
8 int main(void)
9 {
10    // initialize frequency counters to 0
11    int frequency[FREQUENCY_SIZE] = {0};
12
13    // place the survey responses in the responses array
14    int responses[RESPONSES_SIZE] = {1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
15                                1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
16                                5, 6, 7, 5, 6, 4, 8, 6, 8, 10};
17
18    // for each answer, select value of an element of array responses
19    // and use that value as an index in array frequency to
20    // determine element to increment
21    for (size_t answer = 0; answer < RESPONSES_SIZE; ++answer) {
22        ++frequency[responses[answer]];
23    }
24
```

Fig. 6.7 | Analyzing a student poll. (Part I of 2.)

```
25 // display results
26 printf("%s%17s\n", "Rating", "Frequency");
27
28 // output the frequencies in a tabular format
29 for (size_t rating = 1; rating < FREQUENCY_SIZE; ++rating) {
30     printf("%6d%17d\n", rating, frequency[rating]);
31 }
32 }
```

Rating	Frequency
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3

Fig. 6.7 | Analyzing a student poll. (Part 2 of 2.)

```
1 // Fig. 6.9: fig06_09.c
2 // Roll a six-sided die 60,000,000 times
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <time.h>
6 #define SIZE 7
7
8 // function main begins program execution
9 int main(void)
10 {
11     unsigned int frequency[SIZE] = {0}; // clear counts
12
13     srand(time(NULL)); // seed random number generator
14
15     // roll die 60,000,000 times
16     for (unsigned int roll = 1; roll <= 60000000; ++roll) {
17         size_t face = 1 + rand() % 6;
18         ++frequency[face]; // replaces entire switch of Fig. 5.12
19     }
20 }
```

Fig. 6.9 | Roll a six-sided die 60,000,000 times. (Part 1 of 2.)

```
21     printf("%s%17s\n", "Face", "Frequency");
22
23 // output frequency elements 1-6 in tabular format
24 for (size_t face = 1; face < SIZE; ++face) {
25     printf("%4d%17d\n", face, frequency[face]);
26 }
27 }
```

Face	Frequency
1	9997167
2	10003506
3	10001940
4	9995833
5	10000843
6	10000711

Fig. 6.9 | Roll a six-sided die 60,000,000 times. (Part 2 of 2.)

```
1 // Fig. 6.10: fig06_10.c
2 // Treating character arrays as strings.
3 #include <stdio.h>
4 #define SIZE 20
5
6 // function main begins program execution
7 int main(void)
8 {
9     char string1[SIZE]; // reserves 20 characters
10    char string2[] = "string literal"; // reserves 15 characters
11
12    // read string from user into array string1
13    printf("%s", "Enter a string (no longer than 19 characters): ");
14    scanf("%19s", string1); // input no more than 19 characters
15
16    // output strings
17    printf("string1 is: %s\nstring2 is: %s\n"
18          "string1 with spaces between characters is:\n",
19          string1, string2);
```

Fig. 6.10 | Treating character arrays as strings. (Part I of 2.)

```
20
21     // output characters until null character is reached
22     for (size_t i = 0; i < SIZE && string1[i] != '\0'; ++i) {
23         printf("%c ", string1[i]);
24     }
25
26     puts("");
27 }
```

Enter a string (no longer than 19 characters): **Hello there**
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o

Fig. 6.10 | Treating character arrays as strings. (Part 2 of 2.)

```
1 // Fig. 6.11: fig06_11.c
2 // Static arrays are initialized to zero if not explicitly initialized.
3 #include <stdio.h>
4
5 void staticArrayInit(void); // function prototype
6 void automaticArrayInit(void); // function prototype
7
8 // function main begins program execution
9 int main(void)
10 {
11     puts("First call to each function:");
12     staticArrayInit();
13     automaticArrayInit();
14
15     puts("\n\nSecond call to each function:");
16     staticArrayInit();
17     automaticArrayInit();
18 }
19
```

Fig. 6.11 | Static arrays are initialized to zero if not explicitly initialized. (Part 1 of 4.)

```
20 // function to demonstrate a static local array
21 void staticArrayInit(void)
22 {
23     // initializes elements to 0 before the function is called
24     static int array1[3];
25
26     puts("\nValues on entering staticArrayInit:");
27
28     // output contents of array1
29     for (size_t i = 0; i <= 2; ++i) {
30         printf("array1[%u] = %d ", i, array1[i]);
31     }
32
33     puts("\nValues on exiting staticArrayInit:");
34
35     // modify and output contents of array1
36     for (size_t i = 0; i <= 2; ++i) {
37         printf("array1[%u] = %d ", i, array1[i] += 5);
38     }
39 }
40
```

Fig. 6.11 | Static arrays are initialized to zero if not explicitly initialized. (Part 2 of 4.)

```
41 // function to demonstrate an automatic local array
42 void automaticArrayInit(void)
43 {
44     // initializes elements each time function is called
45     int array2[3] = {1, 2, 3};
46
47     puts("\n\nValues on entering automaticArrayInit:");
48
49     // output contents of array2
50     for (size_t i = 0; i <= 2; ++i) {
51         printf("array2[%u] = %d ", i, array2[i]);
52     }
53
54     puts("\nValues on exiting automaticArrayInit:");
55
56     // modify and output contents of array2
57     for (size_t i = 0; i <= 2; ++i) {
58         printf("array2[%u] = %d ", i, array2[i] += 5);
59     }
60 }
```

Fig. 6.11 | Static arrays are initialized to zero if not explicitly initialized. (Part 3 of 4.)

First call to each function:

Values on entering staticArrayInit:

array1[0] = 0 array1[1] = 0 array1[2] = 0

Values on exiting staticArrayInit:

array1[0] = 5 array1[1] = 5 array1[2] = 5

Values on entering automaticArrayInit:

array2[0] = 1 array2[1] = 2 array2[2] = 3

Values on exiting automaticArrayInit:

array2[0] = 6 array2[1] = 7 array2[2] = 8

Second call to each function:

Values on entering staticArrayInit:

array1[0] = 5 array1[1] = 5 array1[2] = 5 — values preserved from last call

Values on exiting staticArrayInit:

array1[0] = 10 array1[1] = 10 array1[2] = 10

Values on entering automaticArrayInit:

array2[0] = 1 array2[1] = 2 array2[2] = 3 — values reinitialized after last call

Values on exiting automaticArrayInit:

array2[0] = 6 array2[1] = 7 array2[2] = 8

Fig. 6.11 | Static arrays are initialized to zero if not explicitly initialized. (Part 4 of 4.)

```
1 // Fig. 6.12: fig06_12.c
2 // Array name is the same as the address of the array's first element.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main(void)
7 {
8     char array[5]; // define an array of size 5
9
10    printf("    array = %p\n&array[0] = %p\n    &array = %p\n",
11          array, &array[0], &array);
12 }
```

```
array = 0031F930
&array[0] = 0031F930
    &array = 0031F930
```

Fig. 6.12 | Array name is the same as the address of the array's first element.

```
1 // Fig. 6.13: fig06_13.c
2 // Passing arrays and individual array elements to functions.
3 #include <stdio.h>
4 #define SIZE 5
5
6 // function prototypes
7 void modifyArray(int b[], size_t size);
8 void modifyElement(int e);
9
10 // function main begins program execution
11 int main(void)
12 {
13     int a[SIZE] = {0, 1, 2, 3, 4}; // initialize array a
14
15     puts("Effects of passing entire array by reference:\n\nThe ");
16     puts("values of the original array are:");
17
18     // output original array
19     for (size_t i = 0; i < SIZE; ++i) {
20         printf("%3d", a[i]);
21     }
22
23     puts(""); // outputs a newline
24
```

Fig. 6.13 | Passing arrays and individual array elements to functions. (Part I of 4.)

```
25     modifyArray(a, SIZE); // pass array a to modifyArray by reference
26     puts("The values of the modified array are:");
27
28     // output modified array
29     for (size_t i = 0; i < SIZE; ++i) {
30         printf("%3d", a[i]);
31     }
32
33     // output value of a[3]
34     printf("\n\nEffects of passing array element "
35           "by value:\n\nThe value of a[3] is %d\n", a[3]);
36
37     modifyElement(a[3]); // pass array element a[3] by value
38
39     // output value of a[3]
40     printf("The value of a[3] is %d\n", a[3]);
41 }
42
```

Fig. 6.13 | Passing arrays and individual array elements to functions. (Part 2 of 4.)

```
43 // in function modifyArray, "b" points to the original array "a"
44 // in memory
45 void modifyArray(int b[], size_t size)
46 {
47     // multiply each array element by 2
48     for (size_t j = 0; j < size; ++j) {
49         b[j] *= 2; // actually modifies original array
50     }
51 }
52
53 // in function modifyElement, "e" is a local copy of array element
54 // a[3] passed from main
55 void modifyElement(int e)
56 {
57     // multiply parameter by 2
58     printf("Value in modifyElement is %d\n", e *= 2);
59 }
```

Fig. 6.13 | Passing arrays and individual array elements to functions. (Part 3 of 4.)

Effects of passing entire array by reference:

The values of the original array are:

0 1 2 3 4

The values of the modified array are:

0 2 4 6 8

Effects of passing array element by value:

The value of a[3] is 6

Value in modifyElement is 12

The value of a[3] is 6

Fig. 6.13 | Passing arrays and individual array elements to functions. (Part 4 of 4.)

```
1 // in function tryToModifyArray, array b is const, so it cannot be
2 // used to modify its array argument in the caller
3 void tryToModifyArray(const int b[])
4 {
5     b[0] /= 2; // error
6     b[1] /= 2; // error
7     b[2] /= 2; // error
8 }
```

Fig. 6.14 | Using the `const` type qualifier with arrays.

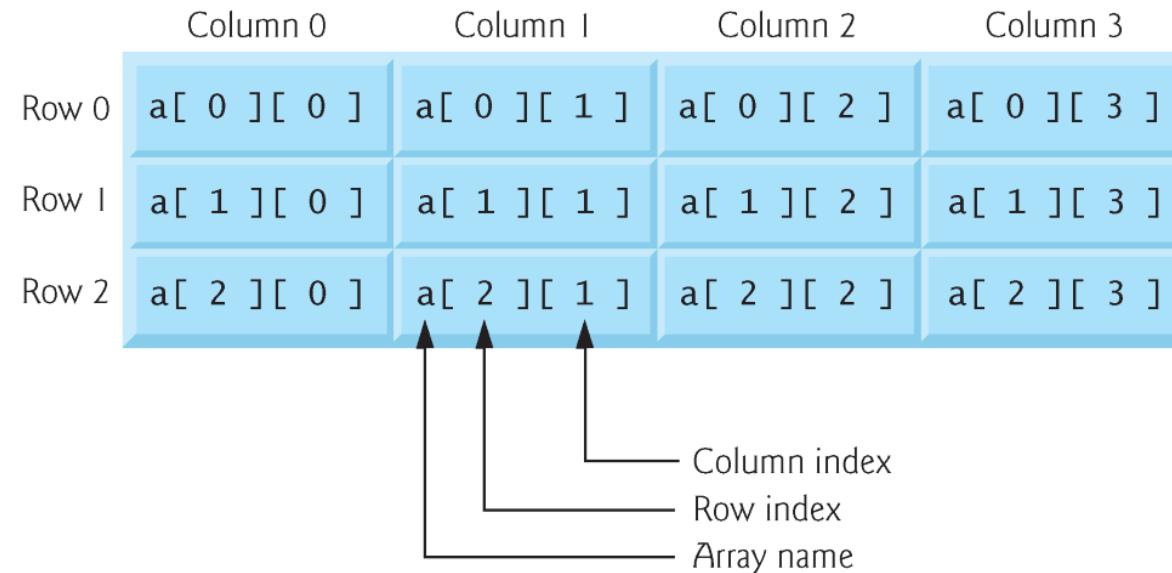


Fig. 6.20 | Two-dimensional array with three rows and four columns.

```
1 // Fig. 6.21: fig06_21.c
2 // Initializing multidimensional arrays.
3 #include <stdio.h>
4
5 void printArray(int a[][]); // function prototype
6
7 // function main begins program execution
8 int main(void)
9 {
10    int array1[2][3] = {{1, 2, 3}, {4, 5, 6}};
11    puts("Values in array1 by row are:");
12    printArray(array1);
13
14    int array2[2][3] = {1, 2, 3, 4, 5};
15    puts("Values in array2 by row are:");
16    printArray(array2);
17
18    int array3[2][3] = {{1, 2}, {4}};
19    puts("Values in array3 by row are:");
20    printArray(array3);
21 }
22
```

Fig. 6.21 | Initializing multidimensional arrays. (Part 1 of 2.)

```
23 // function to output array with two rows and three columns
24 void printArray(int a[][][3])
25 {
26     // loop through rows
27     for (size_t i = 0; i <= 1; ++i) {
28
29         // output column values
30         for (size_t j = 0; j <= 2; ++j) {
31             printf("%d ", a[i][j]);
32         }
33
34         printf("\n"); // start new line of output
35     }
36 }
```

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

Fig. 6.21 | Initializing multidimensional arrays. (Part 2 of 2.)

```
1 // Fig. 6.22: fig06_22.c
2 // Two-dimensional array manipulations.
3 #include <stdio.h>
4 #define STUDENTS 3
5 #define EXAMS 4
6
7 // function prototypes
8 int minimum(const int grades[][][EXAMS], size_t pupils, size_t tests);
9 int maximum(const int grades[][][EXAMS], size_t pupils, size_t tests);
10 double average(const int setOfGrades[], size_t tests);
11 void printArray(const int grades[][][EXAMS], size_t pupils, size_t tests);
12
13 // function main begins program execution
14 int main(void)
15 {
16     // initialize student grades for three students (rows)
17     int studentGrades[STUDENTS][EXAMS] =
18         { { 77, 68, 86, 73 },
19         { 96, 87, 89, 78 },
20         { 70, 90, 86, 81 } };
21
22     // output array studentGrades
23     puts("The array is:");
24     printArray(studentGrades, STUDENTS, EXAMS);
```

Fig. 6.22 | Two-dimensional array manipulations. (Part I of 7.)

```
25
26 // determine smallest and largest grade values
27 printf("\n\nLowest grade: %d\nHighest grade: %d\n",
28     minimum(studentGrades, STUDENTS, EXAMS),
29     maximum(studentGrades, STUDENTS, EXAMS));
30
31 // calculate average grade for each student
32 for (size_t student = 0; student < STUDENTS; ++student) {
33     printf("The average grade for student %u is %.2f\n",
34         student, average(studentGrades[student], EXAMS));
35 }
36 }
37
```

Fig. 6.22 | Two-dimensional array manipulations. (Part 2 of 7.)

```
38 // Find the minimum grade
39 int minimum(const int grades[][][EXAMS], size_t pupils, size_t tests)
40 {
41     int lowGrade = 100; // initialize to highest possible grade
42
43     // Loop through rows of grades
44     for (size_t i = 0; i < pupils; ++i) {
45
46         // Loop through columns of grades
47         for (size_t j = 0; j < tests; ++j) {
48
49             if (grades[i][j] < lowGrade) {
50                 lowGrade = grades[i][j];
51             }
52         }
53     }
54
55     return lowGrade; // return minimum grade
56 }
57
```

Fig. 6.22 | Two-dimensional array manipulations. (Part 3 of 7.)

```
58 // Find the maximum grade
59 int maximum(const int grades[][][EXAMS], size_t pupils, size_t tests)
60 {
61     int highGrade = 0; // initialize to lowest possible grade
62
63     // Loop through rows of grades
64     for (size_t i = 0; i < pupils; ++i) {
65
66         // Loop through columns of grades
67         for (size_t j = 0; j < tests; ++j) {
68
69             if (grades[i][j] > highGrade) {
70                 highGrade = grades[i][j];
71             }
72         }
73     }
74
75     return highGrade; // return maximum grade
76 }
77
```

Fig. 6.22 | Two-dimensional array manipulations. (Part 4 of 7.)

```
78 // Determine the average grade for a particular student
79 double average(const int setOfGrades[], size_t tests)
80 {
81     int total = 0; // sum of test grades
82
83     // total all grades for one student
84     for (size_t i = 0; i < tests; ++i) {
85         total += setOfGrades[i];
86     }
87
88     return (double) total / tests; // average
89 }
90
```

Fig. 6.22 | Two-dimensional array manipulations. (Part 5 of 7.)

```
91 // Print the array
92 void printArray(const int grades[][][EXAMS], size_t pupils, size_t tests)
93 {
94     // output column heads
95     printf("%s", " [0] [1] [2] [3] ");
96
97     // output grades in tabular format
98     for (size_t i = 0; i < pupils; ++i) {
99
100         // output label for row
101         printf("\nstudentGrades[%u] ", i);
102
103         // output grades for one student
104         for (size_t j = 0; j < tests; ++j) {
105             printf("%-5d", grades[i][j]);
106         }
107     }
108 }
```

Fig. 6.22 | Two-dimensional array manipulations. (Part 6 of 7.)

The array is:

	[0]	[1]	[2]	[3]
studentGrades[0]	77	68	86	73
studentGrades[1]	96	87	89	78
studentGrades[2]	70	90	86	81

Lowest grade: 68

Highest grade: 96

The average grade for student 0 is 76.00

The average grade for student 1 is 87.50

The average grade for student 2 is 81.75

Fig. 6.22 | Two-dimensional array manipulations. (Part 7 of 7.)

```
1 // Fig. 6.23: fig06_23.c
2 // Using variable-length arrays in C99
3 #include <stdio.h>
4
5 // function prototypes
6 void print1DArray(size_t size, int array[size]);
7 void print2DArray(int row, int col, int array[row][col]);
8
9 int main(void)
10 {
11     printf("%s", "Enter size of a one-dimensional array: ");
12     int arraySize; // size of 1-D array
13     scanf("%d", &arraySize);
14
15     int array[arraySize]; // declare 1-D variable-length array
16
17     printf("%s", "Enter number of rows and columns in a 2-D array: ");
18     int row1, col1; // number of rows and columns in a 2-D array
19     scanf("%d %d", &row1, &col1);
20
21     int array2D1[row1][col1]; // declare 2-D variable-length array
22 }
```

Fig. 6.23 | Using variable-length arrays in C99. (Part 1 of 5.)

```
23     printf("%s",
24         "Enter number of rows and columns in another 2-D array: ");
25     int row2, col2; // number of rows and columns in another 2-D array
26     scanf("%d %d", &row2, &col2);
27
28     int array2D2[row2][col2]; // declare 2-D variable-length array
29
30     // test sizeof operator on VLA
31     printf("\nsizeof(array) yields array size of %d bytes\n",
32             sizeof(array));
33
34     // assign elements of 1-D VLA
35     for (size_t i = 0; i < arraySize; ++i) {
36         array[i] = i * i;
37     }
38
39     // assign elements of first 2-D VLA
40     for (size_t i = 0; i < row1; ++i) {
41         for (size_t j = 0; j < col1; ++j) {
42             array2D1[i][j] = i + j;
43         }
44     }
45
```

Fig. 6.23 | Using variable-length arrays in C99. (Part 2 of 5.)

```
46     // assign elements of second 2-D VLA
47     for (size_t i = 0; i < row2; ++i) {
48         for (size_t j = 0; j < col2; ++j) {
49             array2D2[i][j] = i + j;
50         }
51     }
52
53     puts("\nOne-dimensional array:");
54     print1DArray(arraySize, array); // pass 1-D VLA to function
55
56     puts("\nFirst two-dimensional array:");
57     print2DArray(row1, col1, array2D1); // pass 2-D VLA to function
58
59     puts("\nSecond two-dimensional array:");
60     print2DArray(row2, col2, array2D2); // pass other 2-D VLA to function
61 }
62
63 void print1DArray(size_t size, int array[size])
64 {
65     // output contents of array
66     for (size_t i = 0; i < size; i++) {
67         printf("array[%d] = %d\n", i, array[i]);
68     }
69 }
```

Fig. 6.23 | Using variable-length arrays in C99. (Part 3 of 5.)

```
70
71 void print2DArray(size_t row, size_t col, int array[row][col])
72 {
73     // output contents of array
74     for (size_t i = 0; i < row; ++i) {
75         for (size_t j = 0; j < col; ++j) {
76             printf("%5d", array[i][j]);
77         }
78         puts("");
79     }
80 }
81 }
```

Fig. 6.23 | Using variable-length arrays in C99. (Part 4 of 5.)

```
Enter size of a one-dimensional array: 6
Enter number of rows and columns in a 2-D array: 2 5
Enter number of rows and columns in another 2-D array: 4 3
```

`sizeof(array)` yields array size of 24 bytes

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

Fig. 6.23 | Using variable-length arrays in C99. (Part 5 of 5.)