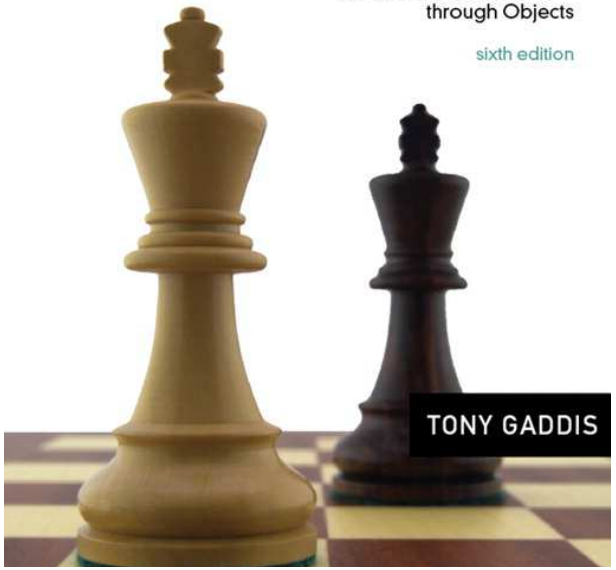
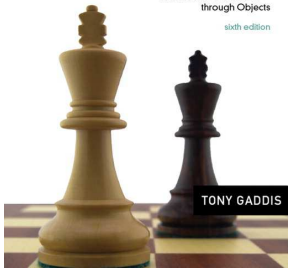
STARTING OUT WITH C++
From Control Structures through Objects
sixth edition

Chapter 7:

Arrays



STARTING OUT WITH C++
From Control Structures through Objects
sixth edition



7.1

Arrays Hold Multiple Values

Arrays Hold Multiple Values

- A single variable can only hold one value

```
int test;
```

95

Enough memory for 1 `int`

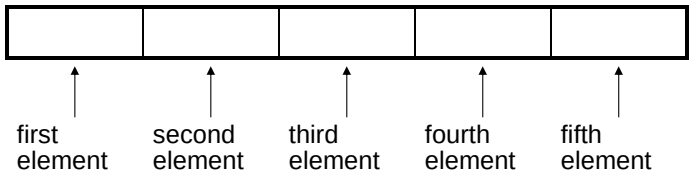
- What if we need to store many test scores
- Array: variable that can store multiple values of the same type
- Values are stored in a block of continuous memory cells

Array - Memory Layout

- Declared using `[]` operator:

```
int tests[5];
```

- The above definition allocates the following memory:



Array Terminology

In the definition `int tests[5];`

- `int` is the data type of the array elements
- `tests` is the name of the array
- `5,` in `[5],` is the size declarator. It shows the number of elements in the array

Arrays of any data type can be defined

```
float temperatures[100];
char name[41];
double grades[30];
```


Array Terminology

- The size of an array (the amount of memory used by the array) is:
  - the total number of bytes allocated for it
  - (number of elements) * (number of bytes for each element)
- Examples:

  `int tests[5]` is an array of 20 bytes, assuming 4 bytes for an `int`

  `long double measures[10]`is an array of 80 bytes, assuming 8 bytes for a `long double`

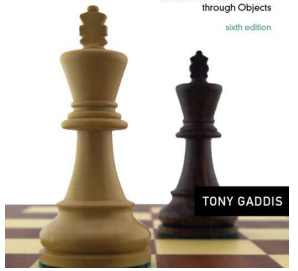
Size Declarators

- Named constants are commonly used as size declarators.

```
const int SIZE = 5;
int tests[SIZE];
```

- This eases program maintenance when the size of the array needs to be changed.

STARTING OUT WITH C++
From Control Structures through Objects
sixth edition

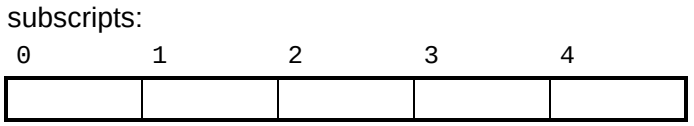


7.2

Accessing Array Elements

Accessing Array Elements

- Each element in an array is assigned a unique *subscript* or index
- Each array element can be accessed through its subscript
- Subscripts start at 0 and end at *n*-1 where *n* is the number of elements in the array



Accessing Array Elements

- Given the following program:

```
int tests[5];
tests[0] = 85;
tests[3] = 90;
```

tests[0]	tests[1]	tests[2]	tests[3]	tests[4]
85	?	?	90	?

```
tests[5] = 75;      // not valid
```


Accessing Array Elements

- Array elements can be used just like regular variables:

```
tests[0] = 79;
cout << tests[0];
cin >> tests[1];
tests[4] = tests[0] + tests[1];
```

- Arrays must be accessed via individual elements (can't read whole array with one statement):

```
cout << tests;       // not legal
```


Program 7-1

```
1  // This program asks for the number of hours worked
2  // by six employees. It stores the values in an array.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     const int NUM_EMPLOYEES = 6;
9     int hours[NUM_EMPLOYEES];
10
11    // Get the hours worked by each employee.
12    cout << "Enter the hours worked by "
13         << NUM_EMPLOYEES << " employees: ";
14    cin >> hours[0];
15    cin >> hours[1];
16    cin >> hours[2];
17    cin >> hours[3];
18    cin >> hours[4];
19    cin >> hours[5];
20
```

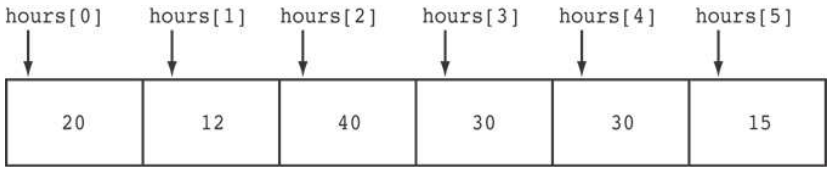
(Program Continues)

```
21    // Display the values in the array.
22    cout << "The hours you entered are:";
23    cout << " " << hours[0];
24    cout << " " << hours[1];
25    cout << " " << hours[2];
26    cout << " " << hours[3];
27    cout << " " << hours[4];
28    cout << " " << hours[5] << endl;
29    return 0;
30 }
```

Program Output with Example Input Shown in Bold

```
Enter the hours worked by 6 employees: 20 12 40 30 30 15 [Enter]
The hours you entered are: 20 12 40 30 30 15
```

Here are the contents of the `hours` array, with the values entered by the user in the example output:



Accessing Array Contents

- The size declarator of an array definition must be a constant or literal
- But the subscript of an array can also use integer variable or expression

```
int i = 5;
cout << tests[i] << endl;
cin >> tests[i+1];
```

- This makes it much easier to access array elements with a loop

Using a Loop to Step Through an Array

- Example – The following code defines an array, `numbers`, and initializes each element to 0:

```
const int ARRAY_SIZE = 5;
int numbers[ARRAY_SIZE];

for (int count = 0; count < ARRAY_SIZE; count++)
     numbers[count] = 0;
```


A Closer Look At the Loop

The variable `count` starts at 0, which is the first valid subscript value.

The loop ends when the variable `count` reaches 5, which is the first invalid subscript value.

```
for (count = 0; count < ARRAY_SIZE; count++)
   numbers[count] = 0;
```

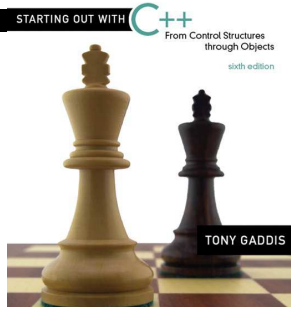
The variable `count` is incremented after each iteration.

Using a Loop to Step Through an Array

```
const int NUM_EMPLOYEES = 6;     // # of employees
int hours[NUM_EMPLOYEES];        // hours worked for
                                    each employee
int count;                       // loop counter

// Input hours worked for each employee
for (count=0; count<NUM_EMPLOYEES; count++)
{
    cout << "Enter hours worked by employee "
         << count+1 << ": ";
    cin >> hours[count];
}
```


Default Initialization

- Global array → all elements initialized to `0` by default
- Local array → all elements *uninitialized* by default



7.3

No Bounds Checking in C++

No Bounds Checking in C++

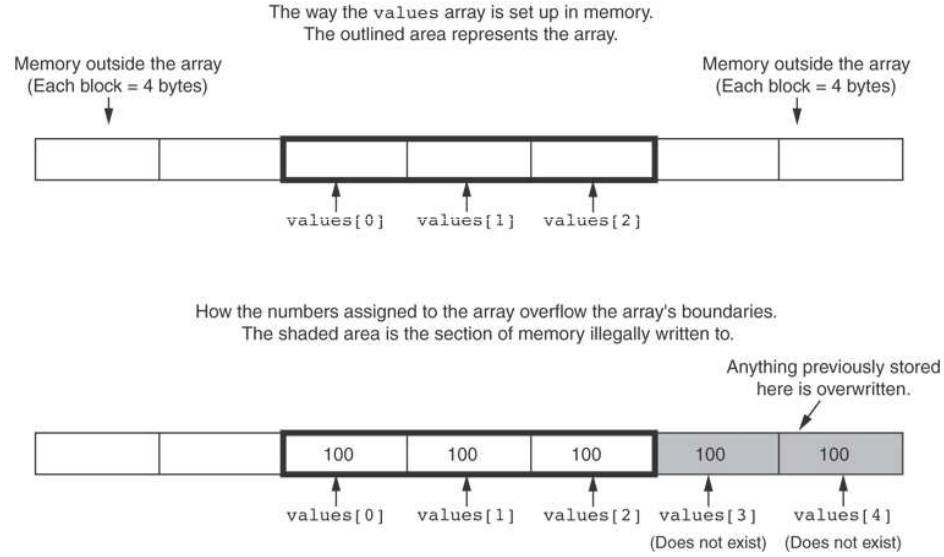
- When you use a value as an array subscript, C++ does not check it to make sure it is a *valid* subscript.
- In other words, you can use subscripts that are beyond the bounds of the array.

Code From Program 7-5

- The following code defines a three-element array, and then writes five values to it!

```
9     const int SIZE = 3;  // Constant for the array size
10    int values[SIZE];    // An array of 3 integers
11    int count;           // Loop counter variable
12
13    // Attempt to store five numbers in the three-element array.
14    cout << "I will store 5 numbers in a 3 element array!\n";
15    for (count = 0; count < 5; count++)
16       values[count] = 100;
```


What the Code Does



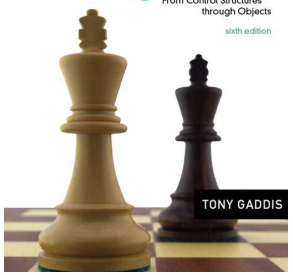
No Bounds Checking in C++

- Be careful not to use invalid subscripts.
- Doing so can corrupt other memory locations, crash program, or lock up computer, and cause elusive bugs.

Off-By-One Errors

- An off-by-one error happens when you use array subscripts that are off by one.
- This can happen when you start subscripts at 1 rather than 0:

```
// This code has an off-by-one error.
const int SIZE = 100;
int numbers[SIZE];
for (int count = 1; count <= SIZE; count++)
   numbers[count] = 0;
```



7.4

Array Initialization

Array Initialization

- Arrays can be initialized with an initialization list:

```
const int SIZE = 5;
int tests[SIZE] = {79,82,91,77,84};
```

- The values are stored in the array in the order in which they appear in the list.
- The initialization list cannot exceed the array size.

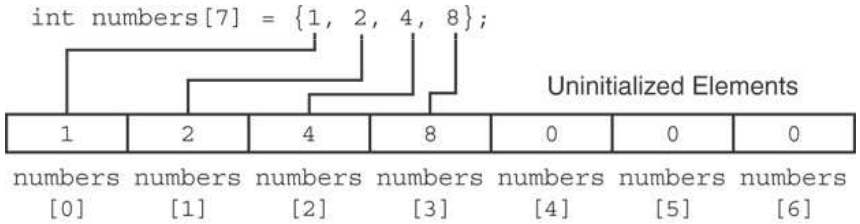
```
7     const int MONTHS = 12;
8     int days[MONTHS] = { 31, 28, 31, 30,
9                          31, 30, 31, 31,
10                         30, 31, 30, 31};
11
12    for (int count = 0; count < MONTHS; count++)
13    {
14       cout << "Month " << (count + 1) << " has ";
15       cout << days[count] << " days.\n";
16    }
```

Program Output

```
Month 1 has 31 days.
Month 2 has 28 days.
Month 3 has 31 days.
Month 4 has 30 days.
Month 5 has 31 days.
Month 6 has 30 days.
Month 7 has 31 days.
Month 8 has 31 days.
Month 9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
```


Partial Array Initialization

- If array is initialized with fewer initial values than the size declarator, the remaining elements will be set to `0`:



- Can't skip elements in initialization list

```
int numbers[7] = {1,  ,4,  ,10};    // NOT legal
```

Implicit Array Sizing

- Can determine array size by the size of the initialization list:

```
int quizzes[]={22,17,15,20};
```

22	17	15	20

- Must use either array size declarator or initialization list at array definition

```
int quizzes[];        // not legal
```

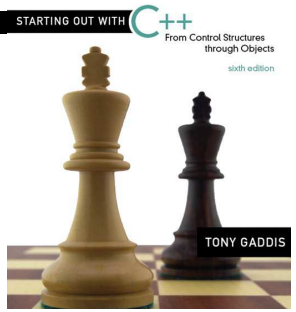

Initializing With a String

- Can use a character array to store a string
- Character array can be initialized by enclosing string in " ":

```
const int SIZE = 6;
char fName[SIZE] = "Henry";
```

- Must leave room for '\0' at end of array
- If initializing character-by-character, must add in '\0' explicitly:

```
char fName[SIZE] =
  {'H', 'e', 'n', 'r', 'y', '\0'};
```

7.5

Processing Array Contents

Processing Array Contents

- Individual array elements are processed like any other type of variable

```
pay = hours[count] * rate;
if ( cost[20] < cost[10] )
  ……
```

- When using ++, -- operators, don't confuse the element with the subscript:

```
tests[i]++;      // add 1 to tests[i]
tests[i++];      // increment i, no
                 // effect on any element
                 // of tests
```

Array Assignment

To copy one array to another,

- Don't try to assign one array to the other:

```
const int SIZE = 4;
int tests[SIZE] = {70, 75, 80, 85};
int newTests[SIZE];
newTests = tests;    // Won't work
```

- Instead, assign element-by-element:

```
for (i=0; i<SIZE; i++)
   newTests[i] = tests[i];
```


Printing the Contents of an Array

- You can display the contents of a *character* array by sending its name to cout:

```
char fName[] = "Henry";
cout << fName << endl;
```

But, this ONLY works with character arrays!

Printing the Contents of an Array

- For other types of arrays, you must print element-by-element:

```
const int SIZE = 4;
int tests[SIZE] = {70, 75, 80, 85};
for (i = 0; i < SIZE; i++)
    cout << tests[i] << endl;
```


Summing and Averaging Array Elements

- Use a simple loop to add together array elements:

```
int tnum;
double average, sum = 0;
for(tnum = 0; tnum < SIZE; tnum++)
      sum += tests[tnum];
```

- Once summed, can compute average:

```
average = sum / SIZE;
```

Finding the Highest Value in an Array

```
int count;
int highest;
highest = numbers[0];
for (count = 1; count < SIZE; count++)
{
   if (numbers[count] > highest)
      highest = numbers[count];
}
```

When this code is finished, the `highest` variable will contain the highest value in the `numbers` array.

Finding the Lowest Value in an Array

```
int count;
int lowest;
lowest = numbers[0];
for (count = 1; count < SIZE; count++)
{
   if (numbers[count] < lowest)
      lowest = numbers[count];
}
```

When this code is finished, the `lowest` variable will contain the lowest value in the `numbers` array.

Partially-Filled Arrays

- If it is unknown how much data an array will be holding:
  - Make the array large enough to hold the largest expected number of elements.
  - Use a counter variable to keep track of the number of items stored in the array.

Partially-Filled Arrays

```
const int SIZE=100;
int studentID[SIZE], count=0, number;

cout << "Enter an ID or -1 to quit: ";
cin >> number;

while ( number != -1 && count < SIZE )
{
    studentID[count++] = number;
    cout << "Enter an ID or -1 to quit: ";
    cin >> number;
}

cout << count << " IDs were entered. They are\n";
for ( int i=0; i<count; i++)
    cout << studentID[i] << endl;
```

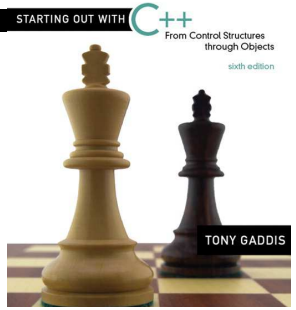
Comparing Arrays

- The following is an incorrect way to compare the two arrays.

```
const int SIZE = 5;
int firstArray[SIZE] = { 5, 10, 15, 20, 25 };
int secondArray[SIZE] = { 5, 10, 15, 20, 25 };

if ( firstArray == secondArray )
   cout << "The arrays are equal.\n";
else
   cout << "The arrays are not equal.\n";
```


You Must Compare Arrays Element-by-Element

```
const  int SIZE = 5;
int    firstArray[SIZE] = { 5, 10, 15, 20, 25 };
int    secondArray[SIZE] = { 5, 10, 15, 20, 25 };
bool   arraysEqual = true;        // Flag variable

// Compare the two arrays.
for ( int index=0; index<SIZE; index++ )
{
   if ( firstArray[index] != secondArray[index] )
   {
      arraysEqual = false;
      break;
   }
}
if (arraysEqual)
   cout << "The arrays are equal.\n";
else
   cout << "The arrays are not equal.\n";
```

7.7

Arrays as Function Arguments

Arrays as Function Arguments

- Arrays can be used to pass blocks of data to functions
- To define a function that takes an array parameter, use empty `[]` for array argument:

```
void showScores(int []);
                  // function prototype
void showScores(int nums[])
                  // function header
```

- To pass an array to a function, just use the array name:

```
showScores(tests);
```

Arrays as Function Arguments

- When passing an array to a function, it is common to pass array size so that function knows how many elements to process:

```
showScores(tests, ARRAY_SIZE);
```

- Array size must also be reflected in the function prototype and header:

```
void showScores(int [], int);
                  // function prototype
void showScores(int nums[], int size)
                  // function header
```


Program 7-14

```
1  // This program demonstrates an array being passed to a function.
2  #include <iostream>
3  using namespace std;
4
5  void showValues(int [], int); // Function prototype
6
7  int main()
8  {
9     const int ARRAY_SIZE = 8;
10    int numbers[ARRAY_SIZE] = {5, 10, 15, 20, 25, 30, 35, 40};
11
12    showValues(numbers, ARRAY_SIZE);
13    return 0;
14 }
15
```

(Program Continues)

Program 7-14 *(Continued)*

```
16 //************************************************
17 // Definition of function showValue.             *
18 // This function accepts an array of integers and  *
19 // the array's size as its arguments. The contents *
20 // of the array are displayed.                     *
21 //************************************************
22
23 void showValues(int nums[], int size)
24 {
25    for (int index = 0; index < size; index++)
26       cout << nums[index] << " ";
27    cout << endl;
28 }
```

Program Output

```
5 10 15 20 25 30 35 40
```


Modifying Arrays in Functions

- Array names in functions are like reference variables – changes made to array in a function are reflected in actual array in calling function
- Need to exercise caution that array is not inadvertently changed by a function

```
// This program uses a function to double the value
// of each element of an array.

#include <iostream>
using namespace std;

// Function prototypes
void doubleArray(int [], int);
void showValues(int [], int);

int main()
{
    const int SIZE = 7;
    int data[SIZE] = {1, 2, 3, 4, 5, 6, 7};

    // Display the initial array values
    cout << "The array's initial values are: " << endl;

    // Double the values in the array
    doubleArray(data, SIZE);
```



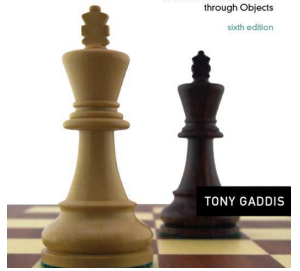
```
    // Display the array values after the function call
    cout << "After calling doubleArray the values are:\n";
    showValues(data, SIZE);

    return 0;
}

// This function doubles the value of each element in the
// array passed nums. The value of size is the number of
// elements in the array.
void doubleArray(int nums[], int size)
{
    for (int index=0; index<size; index++)
        nums[index] = nums[index]*2;
}
```



```
// This function displays the value of each element in the
// array passed nums. The value of size is the number of
// elements in the array.
void showValues(int nums[], int size)
{
    for (int index=0; index<size; index++)
        cout << nums[index] << " ";
    cout << endl;
}
```

Program output:

```
The array's initial values are:
1 2 3 4 5 6 7
After calling doubleArray the values are:
2 4 6 8 10 12 14
```

7.8

Two-Dimensional Arrays

Two-Dimensional Arrays

- 1D array can only hold one set of data
- 2D array can store multiple sets of data
  - Ex. Use a 1D array to store grades for one student. Use a 2D array to store grades for all students
- Like multiple 1D arrays of the same type
- Like a table in a spreadsheet
- Use two size declarators in definition:

```
const int ROWS = 4, COLS = 3;
int exams[ROWS][COLS];
```

- First declarator is number of rows; second is number of columns

Two-Dimensional Array Representation

```
const int ROWS = 4, COLS = 3;
int exams[ROWS][COLS];
```

	Column 0	Column 1	Column 2
Row 0	exams[0][0]	exams[0][1]	exams[0][2]
Row 1	exams[1][0]	exams[1][1]	exams[1][2]
Row 2	exams[2][0]	exams[2][1]	exams[2][2]
Row 3	exams[3][0]	exams[3][1]	exams[3][2]

- Use two subscripts to access element:
  `exams[2][2] = 86;`

Use nested loops to cycle through each element of a two-dimensional array

Program 7-18

```
1  // This program demonstrates a two-dimensional array.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  int main()
7  {
8     const int NUM_DIVS = 3;           // Number of divisions
9     const int NUM_QTRS = 4;           // Number of quarters
10    double sales[NUM_DIVS][NUM_QTRS]; // Array with 3 rows and 4 columns.
11    double totalSales = 0;            // To hold the total sales.
12    int div, qtr;                     // Loop counters.
13
14    cout << "This program will calculate the total sales of\n";
15    cout << "all the company's divisions.\n";
16    cout << "Enter the following sales information:\n\n";
17
```

(program continues)

Program 7-18 *(continued)*

```
18    // Nested loops to fill the array with quarterly
19    // sales figures for each division.
20    for (div = 0; div < NUM_DIVS; div++)
21    {
22       for (qtr = 0; qtr < NUM_QTRS; qtr++)
23       {
24          cout << "Division " << (div + 1);
25          cout << ", Quarter " << (qtr + 1) << ": $";
26          cin >> sales[div][qtr];
27       }
28       cout << endl; // Print blank line.
29    }
30
31    // Nested loops used to add all the elements.
32    for (div = 0; div < NUM_DIVS; div++)
33    {
34       for (qtr = 0; qtr < NUM_QTRS; qtr++)
35          totalSales += sales[div][qtr];
36    }
37
38    cout << fixed << showpoint << setprecision(2);
39    cout << "The total sales for the company are: $";
40    cout << totalSales << endl;
41    return 0;
42 }
```

Program Output with Example Input Shown in Bold

```
This program will calculate the total sales of
all the company's divisions.
Enter the following sales data:

Division 1, Quarter 1: $31569.45 [Enter]
Division 1, Quarter 2: $29654.23 [Enter]
Division 1, Quarter 3: $32982.54 [Enter]
Division 1, Quarter 4: $39651.21 [Enter]

Division 2, Quarter 1: $56321.02 [Enter]
Division 2, Quarter 2: $54128.63 [Enter]
Division 2, Quarter 3: $41235.85 [Enter]
Division 2, Quarter 4: $54652.33 [Enter]

Division 3, Quarter 1: $29654.35 [Enter]
Division 3, Quarter 2: $28963.32 [Enter]
Division 3, Quarter 3: $25353.55 [Enter]
Division 3, Quarter 4: $32615.88 [Enter]

The total sales for the company are: $456782.34
```


2D Array Initialization

- Two-dimensional arrays are initialized row-by-row:

```
const int ROWS = 2, COLS = 2;
int exams[ROWS][COLS] = {{84, 78},{92, 97}};
```

Or:

```
int exams[ROWS][COLS] = {{84, 78},
                         {92, 97}};
```

	Col 0	Col 1
Row 0	84	78
Row 1	92	97

2D Array Initialization

- Can omit inner `{ }`. The following are the same.

```
int exams[ROWS][COLS] = {{84, 78},{92, 97}};
int exams[ROWS][COLS] = {84, 78, 92, 97};
```

- Extra braces provide ability to leave out some initial values in a row – array elements without initial values will be set to `0` or `NULL`

```
int exams[ROWS][COLS] = { {84}, {92, 97} };
(exams[0][1] is automatically set to 0)
```


Two-Dimensional Array as Parameter, Argument

- Use empty `[]` for row, size declarator for column in prototype & header, and an `int` for # of rows:

```
const int COLS = 2;

// Prototype
void getExams(int [][COLS], int);

// Header
void getExams(int exams[][COLS], int rows)
```

- Use array name as argument in function call:

```
getExams(exams, 2);
```

Example – The showArray Function from Program 7-19

```
30 //*****************************************************************
31 // Function Definition for showArray                              *
32 // The first argument is a two-dimensional int array with COLS    *
33 // columns. The second argument, rows, specifies the number of    *
34 // rows in the array. The function displays the array's contents. *
35 //*****************************************************************
36
37 void showArray(int array[][COLS], int rows)
38 {
39    for (int x = 0; x < rows; x++)
40    {
41       for (int y = 0; y < COLS; y++)
42       {
43          cout << setw(4) << array[x][y] << " ";
44       }
45       cout << endl;
46    }
47 }
```


How showArray is Called

```
15    int table1[TBL1_ROWS][COLS] = {{1, 2, 3, 4},
16                                   {5, 6, 7, 8},
17                                   {9, 10, 11, 12}};
18    int table2[TBL2_ROWS][COLS] = {{10, 20, 30, 40},
19                                   {50, 60, 70, 80},
20                                   {90, 100, 110, 120},
21                                   {130, 140, 150, 160}};
22
23    cout << "The contents of table1 are:\n";
24    showArray(table1, TBL1_ROWS);
25    cout << "The contents of table2 are:\n";
26    showArray(table2, TBL2_ROWS);
```


Summing All the Elements in a Two-Dimensional Array

- Given the following definitions:

```
const int NUM_ROWS = 5; // Number of rows
const int NUM_COLS = 5; // Number of columns
int total = 0;          // Accumulator
int numbers[NUM_ROWS][NUM_COLS] =
   { {2, 7, 9, 6, 4},
     {6, 1, 8, 9, 4},
     {4, 3, 7, 2, 9},
     {9, 9, 0, 3, 1},
     {6, 2, 7, 4, 1} };
```


Summing All the Elements in a Two-Dimensional Array

```
// Sum the array elements.
for (int row = 0; row < NUM_ROWS; row++)
{
   for (int col = 0; col < NUM_COLS; col++)
      total += numbers[row][col];
}

// Display the sum.
cout << "The total is " << total << endl;
```

Summing the Rows of a Two-Dimensional Array

- Given the following definitions:

```
const int NUM_STUDENTS = 3;
const int NUM_SCORES = 5;
double total;       // Accumulator
double average;     // To hold average scores
double scores[NUM_STUDENTS][NUM_SCORES] =
     {{88, 97, 79, 86, 94},
      {86, 91, 78, 79, 84},
      {82, 73, 77, 82, 89}};
```


Summing the Rows of a Two-Dimensional Array

```
// Get each student's average score.
for (int row = 0; row < NUM_STUDENTS; row++)
{
   // Set the accumulator.
   total = 0;
   // Sum a row.
   for (int col = 0; col < NUM_SCORES; col++)
      total += scores[row][col];
   // Get the average
   average = total / NUM_SCORES;
   // Display the average.
   cout << "Score average for student "
        << (row + 1) << " is " << average <<endl;
}
```


Summing the Columns of a Two-Dimensional Array

- Given the following definitions:

```
const int NUM_STUDENTS = 3;
const int NUM_SCORES = 5;
double total;       // Accumulator
double average;     // To hold average scores
double scores[NUM_STUDENTS][NUM_SCORES] =
     {{88, 97, 79, 86, 94},
      {86, 91, 78, 79, 84},
      {82, 73, 77, 82, 89}};
```


Summing the Columns of a Two-Dimensional Array

```
// Get the class average for each score.
for (int col = 0; col < NUM_SCORES; col++)
{
   // Reset the accumulator.
   total = 0;
   // Sum a column
   for (int row = 0; row < NUM_STUDENTS; row++)
      total += scores[row][col];
   // Get the average
   average = total / NUM_STUDENTS;
   // Display the class average.
   cout << "Class average for test " << (col + 1)
        << " is " << average << endl;
}
```