Arrays Hold Multiple Values

- **Array**: variable that can store multiple values of the same type
- Values are stored in adjacent memory locations
- Declared using `[ ]` operator: `int tests[5];`

Array - Memory Layout

- The definition: `int tests[5];`
  - 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.
  - The size of an array is (number of elements) * (size of each element)
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.

Accessing Array Elements

- Each element in an array is assigned a unique subscript.
- Subscripts start at 0

<table>
<thead>
<tr>
<th>Subscripts:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
</table>

- The last element's subscript is $n-1$ where $n$ is the number of elements in the array.

<table>
<thead>
<tr>
<th>Subscripts:</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
</table>

- Array elements can be used as regular variables:
  ```
  tests[0] = 79;
  cout << tests[0];
  cin >> tests[1];
  tests[4] = tests[0] + tests[1];
  ```
- Arrays must be accessed via individual elements:
  ```
  cout << tests; // not legal
  ```

Accessing Array Elements in Program 7-1

```cpp
// This program asks for the number of hours worked
// by six employees. It stores the values in an array.
#include <iostream>

int main()
{
  int hours[MAX_EMPLOYEES];
  cout << "Enter the hours worked by ";
  cin >> hours[MAX_EMPLOYEES];
  // Get the hours worked by each employee.
  for (int i = 0; i < MAX_EMPLOYEES; i++)
  {
    cout << "Enter the hours worked by ";
    cin >> hours[i];
    cin.ignore();
  }
  return 0;
}
```
Accessing Array Elements in Program 7-1

```cpp
// Display the values in the array.
cout << "Here are the hours you entered:");
for (int count = 0; count < ARRAY_SIZE; count++)
    cout << hours[count] << " ");
```

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>12</td>
<td>40</td>
<td>30</td>
<td>15</td>
</tr>
</tbody>
</table>

Accessing Array Contents

- Can access element with a constant or literal subscript:
  ```cpp
cout << tests[3] << endl;
```

- Can use integer expression as subscript:
  ```cpp
int i = 5;
cout << tests[i] << endl;
```

Using a Loop to Step Through an Array

**Example** – The following code defines an array, `numbers`, and assigns 99 to each element:

```cpp
const int ARRAY_SIZE = 5;
int numbers[ARRAY_SIZE];
for (int count = 0; count < ARRAY_SIZE; count++)
    numbers[count] = 99;
```

A Closer Look At the Loop

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

```cpp
for (count = 0; count < ARRAY_SIZE; count++)
    numbers[count] = 99;
```

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

The variable `count` is incremented after each iteration.

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!

```cpp
const int SIZE = 3; // Constant for the array size
int values[SIZE]; // An array of 3 integers
int count; // Loop counter variable

// Attempt to store five numbers in the three-element array.
cout << "I will store 5 numbers in a 3 element array!n";
for (count = 0; count < 5; count++)
    values[count] = 100;
```

What the Code Does

- 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:

```cpp
// 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:
  ```cpp
cost int SIZE = 5;
int tests[SIZE] = {79, 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.

Partial Array Initialization

- If array is initialized with fewer initial values than the size declarator, the remaining elements will be set to 0:
  ```cpp
int numbers[8] = {1, 2, 4, 8};
```

Implicit Array Sizing

- Can determine array size by the size of the initialization list:
  ```cpp
int quizzes[]={12, 17, 15, 11};
```
- Must use either array size declarator or initialization list at array definition.

The Range-Based for Loop

- C++ 11 provides a specialized version of the for loop that, in many circumstances, simplifies array processing.
- The range-based for loop is a loop that iterates once for each element in an array.
- Each time the loop iterates, it copies an element from the array to a built-in variable, known as the range variable.
- The range-based for loop automatically knows the number of elements in an array.
- You do not have to use a counter variable.
- You do not have to worry about stepping outside the bounds of the array.
The Range-Based for Loop

- Here is the general format of the range-based for loop:

  ```c++
  for (dataType rangeVariable : array)
  
  dataType is the data type of the range variable.
  
  rangeVariable is the name of the range variable. This variable will receive the value of a different array element during each loop iteration.
  
  array is the name of an array on which you wish the loop to operate.
  
  statement is a statement that executes during a loop iteration. If you need to execute more than one statement in the loop, enclose the statements in a set of braces.
  ```

The range-based for loop in Program 7-10

```c++
// This program demonstrates the range-based for loop. 
#include <iostream>
using namespace std;

int main()
{
  // Define an array of integers.
  int numbers[] = { 10, 20, 30, 40, 50 };
  // Display the values in the array.
  for (int val : numbers)
  {
    cout << val << endl;
  }
  return 0;
}
```

Modifying an Array with a Range-Based for Loop

- As the range-based for loop executes, its range variable contains only a copy of an array element.
- You cannot use a range-based for loop to modify the contents of an array unless you declare the range variable as a reference.
- To declare the range variable as a reference variable, simply write an ampersand (&) in front of its name in the loop header.
- Program 7-12 demonstrates

Modifying an Array with a Range-Based for Loop in Program 7-12

```c++
const int SIZE = 5;
int numbers[5];

// Get values for the array.
for (auto &val : numbers)
{
  cout << "Enter an integer value: ";
  cin >> val;
}

// Display the values in the array.
for (int val : numbers)
  cout << val << endl;
```

Modifying an Array with a Range-Based for Loop

- You can use the auto keyword with a reference range variable. For example, the code in lines 12 through 16 in Program 7-12 could have been written like this:

  ```c++
  for (auto &val : numbers)
  {
    cout << "Enter an integer value: ";
    cin >> val;
  }
  ```

The Range-Based for Loop versus the Regular for Loop

- The range-based for loop can be used in any situation where you need to step through the elements of an array, and you do not need to use the element subscripts.
- If you need the element subscript for some purpose, use the regular for loop.
7.6  Processing Array Contents

- Array elements can be treated as ordinary variables of the same type as the array.
- 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 tests
  ```

Array Assignment

- To copy one array to another,
  - Don’t try to assign one array to the other:
    ```
    newTests = tests; // Won’t work
    ```
  - Instead, assign element-by-element:
    ```
    for (i = 0; i < ARRAY_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!

- For other types of arrays, you must print element-by-element:
  ```
  for (i = 0; i < ARRAY_SIZE; i++)
  cout << tests[i] << endl;
  ```

Printing the Contents of an Array

- In C++ 11 you can use the range-based `for` loop to display an array’s contents, as shown here:
  ```
  for (int val : numbers)
  cout << val << endl;
  ```
Summing and Averaging Array Elements

- Use a simple loop to add together array elements:
  ```c
  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

```c
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

```c
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.

Comparing Arrays

- To compare two arrays, you must compare element-by-element:
  ```c
  const int SIZE = 5;
  int firstArray[SIZE] = { 5, 10, 15, 20, 25 };
  int secondArray[SIZE] = { 5, 10, 15, 20, 25 };
  bool arrayEqual = true; // Flag variable
  int count = 0; // Loop counter variable
  // Compare the two arrays, while (arrayEqual as count < SIZE)
  // If (firstArray[count] != secondArray[count])
  // arrayEqual = false;
  // count++;
  // If arrayEqual
  count = "The arrays are equal.");
  else
  count = "The arrays are not equal.");
  ```
7.7

Using Parallel Arrays

Parallel Arrays
- Two or more arrays that contain related data
- A subscript is used to relate arrays: elements at same subscript are related
- Arrays may be of different types

Parallel Array Example

```c++
const int SIZE = 5; // Array size
int id[SIZE];       // student ID
double average[SIZE]; // course average
char grade[SIZE];   // course grade

for(int i = 0; i < SIZE; i++)
{
    cout << "Student ID: " << id[i]
        << " average: " << average[i]
        << " grade: " << grade[i]
        << endl;
}
```

Parallel Arrays in Program 7-15

```c++
// This program uses two parallel arrays: one for hours worked and one for pay rate.
// The arrays are of different types: int for hours and double for pay rate.

const int NUM_employees = 5;
int hours[5];    // array for hours worked
double payrate[5]; // array for pay rate per hour

// Display employee's total pay
for(int i = 0; i < NUM_employees; i++)
{
    double total_pay = hours[i] * payrate[i];
    cout << "Employee \#" << i + 1 << " total pay: " << total_pay << endl;
}
```

Parallel Arrays in Program 7-15

```c++
Program 7-15

// Display employee's total pay
for(int i = 0; i < NUM_employees; i++)
{
    double total_pay = hours[i] * payrate[i];
    cout << "Employee \#" << i + 1 << " total pay: " << total_pay << endl;
}
```

Parallel Arrays in Program 7-15

```c++
Program 7-15 (continued)

// Display employee's total pay
for(int i = 0; i < NUM_employees; i++)
{
    double total_pay = hours[i] * payrate[i];
    cout << "Employee \#" << i + 1 << " total pay: " << total_pay << endl;
}
```
Arrays as Function Arguments

7.8

Arrays as Function Arguments

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

`showScores(tests);`

To define a function that takes an array parameter, use empty `[]` for array argument:

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

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

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

Array size must also be reflected in prototype, header:

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

Passing an Array as Function Arguments

Passing an Array to a Function in Program 7-17

```cpp
Program 7-17
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    showValues(numbers, ARRAY_SIZE);
12    return 0;
13 }
14
15 // Program Output
16 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
7.9

Two-Dimensional Arrays

Can define one array for multiple sets of data
Like a table in a spreadsheet
Use two size declarators in definition:

```c++
const int ROWS = 4, COLS = 3;
extems[ROWS][COLS];
```

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

Two-Dimensional Array Representation

```c++
const int ROWS = 4, COLS = 3;
extems[ROWS][COLS];
```

Use two subscripts to access element:

```c++
exems[2][2] = 86;
```

A Two-dimensional Array in Program 7-21

Program 7-21

```c++
// This program demonstrates a two-dimensional array.
#include <iostream>
using namespace std;
int main()
{
int exams[4][3];

// Initial values
for (int i = 0; i < 4; i++)
for (int j = 0; j < 3; j++)
exams[i][j] = 86;

for (int i = 0; i < 4; i++)
for (int j = 0; j < 3; j++)
cout << exams[i][j] << " \"

return 0;
```

A Two-dimensional Array in Program 7-21

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:

<table>
<thead>
<tr>
<th>Division</th>
<th>Quarter 1</th>
<th>Quarter 2</th>
<th>Quarter 3</th>
<th>Quarter 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Division 1</td>
<td>29064.43</td>
<td>29064.43</td>
<td>29064.43</td>
<td>29064.43</td>
</tr>
<tr>
<td>Division 2</td>
<td>29064.43</td>
<td>29064.43</td>
<td>29064.43</td>
<td>29064.43</td>
</tr>
<tr>
<td>Division 3</td>
<td>29064.43</td>
<td>29064.43</td>
<td>29064.43</td>
<td>29064.43</td>
</tr>
<tr>
<td>Division 4</td>
<td>29064.43</td>
<td>29064.43</td>
<td>29064.43</td>
<td>29064.43</td>
</tr>
</tbody>
</table>

The total sales for the company are $45768.12

8/23/2014
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}};
  ```
- Can omit inner `{ }`, some initial values in a row – array elements without initial values will be set to 0 or NULL.

Two-Dimensional Array as Parameter, Argument

- Use array name as argument in function call:
  ```
  getExams(exams, 2);
  ```
- Use empty `{ }` for row, size declarator for column in prototype, header:
  ```
  const int COLS = 2;
  // Prototype
  void getExams(int [ ] [COLS], int);
  // Header
  void getExams(int exams [ ] [COLS], int rows)
  ```

Example – The showArray Function from Program 7-22

```
// Function definition for showArray

// The first argument is a two-dimensional int array with COLS // columns. The second argument, rows, specifies the number of // rows in the array. The function displays the array's contents.

void showArray(int array[][COLS], int rows)
{
  for (int row = 0; row < rows; row++)
  {
    for (int col = 0; col < COLS; col++)
    {
      cout << array[row][col] << " ";
    }
    cout << endl;
  }
}
```

How showArray is Called

```
int table1[TELL_ROWS][COLS] = {{1, 2, 3, 4},
                                 {5, 6, 7, 8},
                                 {9, 10, 11, 12}};
int table2[TELL_ROWS][COLS] = {{15, 20, 30, 40},
                                 {55, 60, 70, 80},
                                 {90, 100, 110, 120},
                                 {130, 140, 150, 160}};

cout << "The contents of table1 are:\n";
showArray(table1, TELL_ROWS);
cout << "The contents of table2 are:\n";
showArray(table2, TELL_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, 6, 5, 7, 1},
       {6, 2, 7, 4, 1}};
  ```

```
// 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.
```
Summing the Rows of a Two-Dimensional Array

Given the following definitions:

```c++
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}};
```

// 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:

```c++
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}};
```

// 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_SCORES;
    // Display the class average.
    cout << "Class average for test " << (col + 1) << " is " << average << endl;
}

Arrays with Three or More Dimensions

- Can define arrays with any number of dimensions:
  ```c++
  short rectSolid[2][3][5];
  double timeGrid[3][4][3][4];
  ```
- When used as parameter, specify all but 1st dimension in prototype, heading:
  ```c++
  void getRectSolid(short [][3][5]);
  ```
7.12

Introduction to the STL vector

A data type defined in the Standard Template Library (covered more in Chapter 16)

Can hold values of any type:

```cpp
vector<int> scores;
```

Automatically adds space as more is needed – no need to determine size at definition

Can use [] to access elements

---

Declaring Vectors

- You must `#include<vector>`
- Declare a vector to hold int element:
  ```cpp
  vector<int> scores;
  ```
- Declare a vector with initial size 30:
  ```cpp
  vector<int> scores(30);
  ```
- Declare a vector and initialize all elements to 0:
  ```cpp
  vector<int> scores(30, 0);
  ```
- Declare a vector initialized to size and contents of another vector:
  ```cpp
  vector<int> finals(scores);
  ```

---

Adding Elements to a Vector

- If you are using C++ 11, you can initialize a vector with a list of values:
  ```cpp
  vector<int> numbers { 10, 20, 30, 40);
  ```
- Use `push_back` member function to add element to a full array or to an array that had no defined size:
  ```cpp
  scores.push_back(75);
  ```
- Use `size` member function to determine size of a vector:
  ```cpp
  howbig = scores.size();
  ```

---

Removing Vector Elements

- Use `pop_back` member function to remove last element from vector:
  ```cpp
  scores.pop_back();
  ```
- To remove all contents of vector, use `clear` member function:
  ```cpp
  scores.clear();
  ```
- To determine if vector is empty, use `empty` member function:
  ```cpp
  while (!scores.empty()) ...
<table>
<thead>
<tr>
<th>Member Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>at(elt)</td>
<td>Returns the value of the element at position <code>elt</code> in the vector</td>
<td><code>cout &lt;&lt; vec1.at(i);</code></td>
</tr>
<tr>
<td>capacity()</td>
<td>Returns the maximum number of elements a vector can store without allocating more memory</td>
<td><code>maxelts = vec1.capacity();</code></td>
</tr>
<tr>
<td>reverse()</td>
<td>Reverse the order of the elements in a vector</td>
<td><code>vec1.reverse();</code></td>
</tr>
<tr>
<td>resize(elt, val)</td>
<td>Add elements to a vector, optionally initializes them</td>
<td><code>vec1.resize(5,0);</code></td>
</tr>
<tr>
<td>swap(vec2)</td>
<td>Exchange the contents of two vectors</td>
<td><code>vec1.swap(vec2);</code></td>
</tr>
</tbody>
</table>