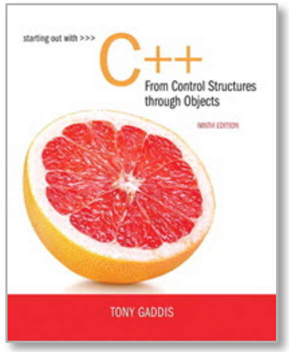


7.3

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.

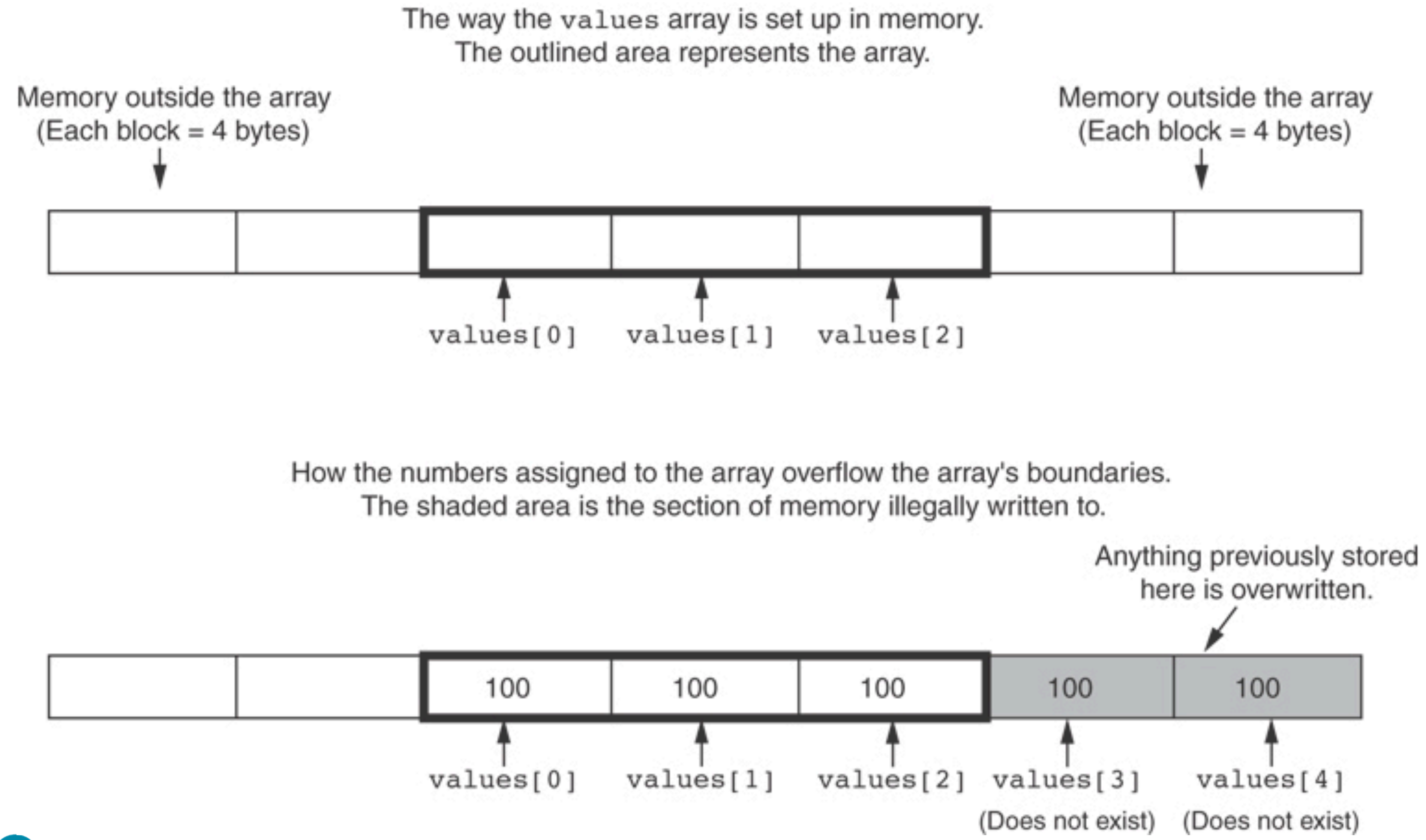
Code From Program 7-9

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

```
    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 3-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


The way the values array is set up in memory.
The outlined area represents the array.
Memory outside the array
(Each block = 4 bytes)
Memory outside the array
(Each block = 4 bytes)
values[0]
values[1]
values[2]
How the numbers assigned to the array overflow the array's boundaries.
The shaded area is the section of memory illegally written to.
Anything previously stored
here is overwritten.
100
100
100
100
100
values[0]
values[1]
values[2]
values[3]
(Does not exist)
values[4]
(Does not exist)

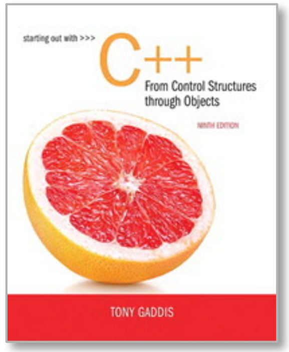
No Bounds Checking in C++

- 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

The Range-Based `for` Loop

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:

```
for (dataType rangeVariable : array)
        statement;
```

- ***`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

```
// 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

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

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

// Display the values in the array.
cout << "Here are the values you entered:\n";
for (int val : numbers)
   cout << val << endl;
```

Modifying an Array with a Range-Based `for` Loop

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

```
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.