Chapter 7: Arrays

Hanoi University of Science and Technology

December 17, 2025

Slide contents are mainly based on the following documents:

- Tony Gaddis, Starting out with C++, Six Edition, Pearson, 2009
- Tony Gaddis, Power Points Slides:
  `https://www.umsl.edu/~antognolij/gaddis.html`

Arrays Hold Multiple Values

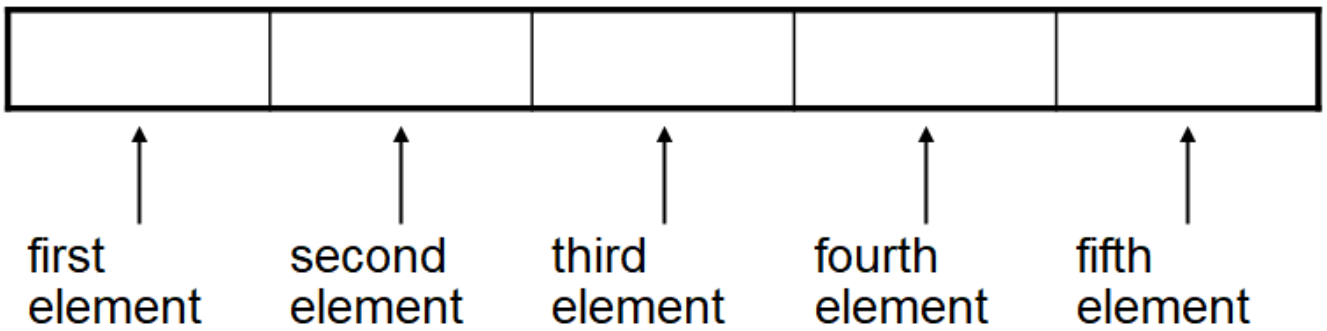
- Array: variable that can store multiple values of the same type
- Values are stored in adjacent memory locations
- Declared using [ ] operator:

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

- 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)

Array Terminology

- The size of an 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`

Size Declarators

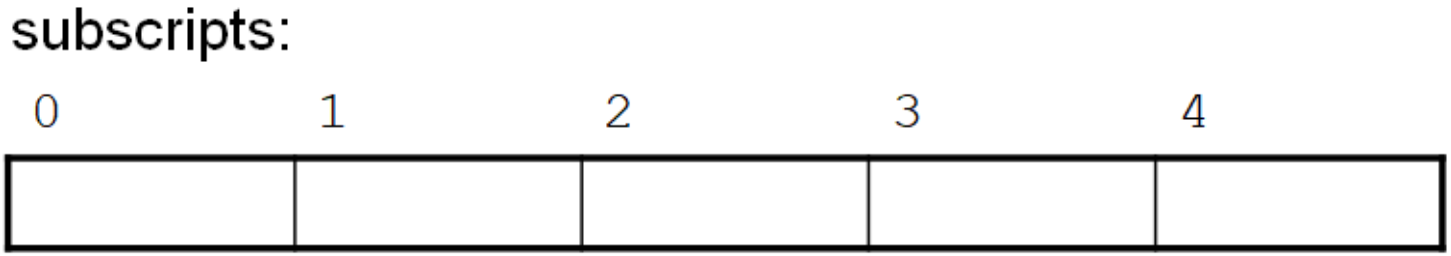
- Named constants are commonly used as size declarators.

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



Accessing Array Elements

- 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

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
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
```

Accessing Array Contents

- Can access element with a constant or literal subscript:

```
cout << tests[3] << endl;
```

- Can use integer expression as subscript:

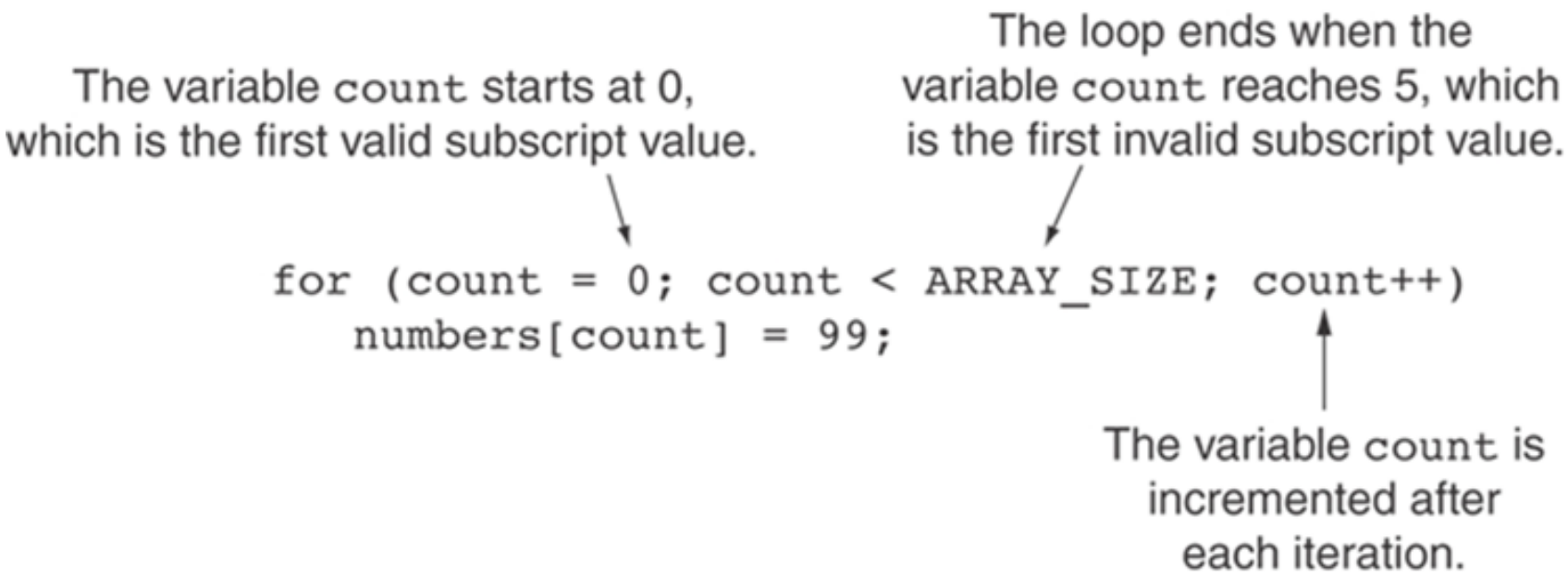
```
int i = 5;
cout << tests[i] << endl;
```

Using a Loop to Step Through an Array

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

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

Accessing Array Elements



Default Initialization

- Global array: all elements initialized to 0 by default
- Local array: all elements uninitialized by default

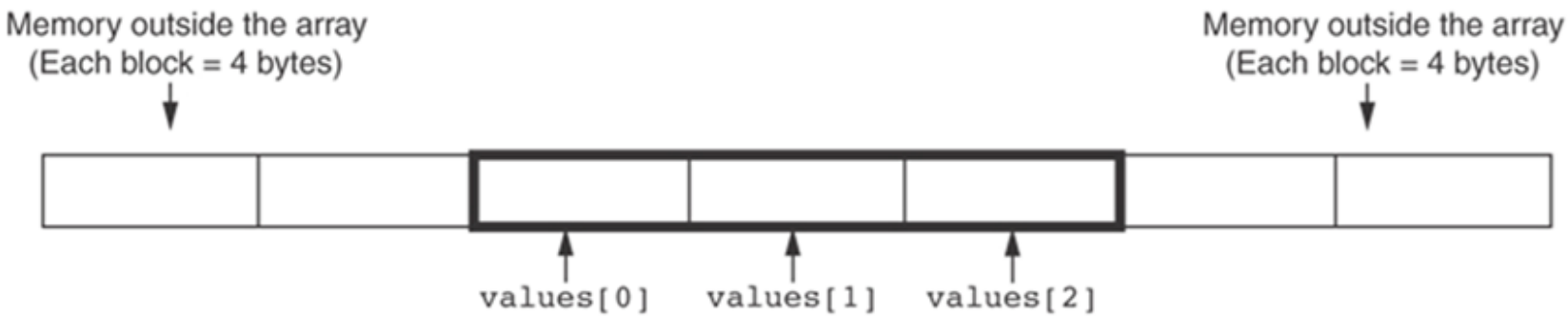
No Bounds Checking in C++

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

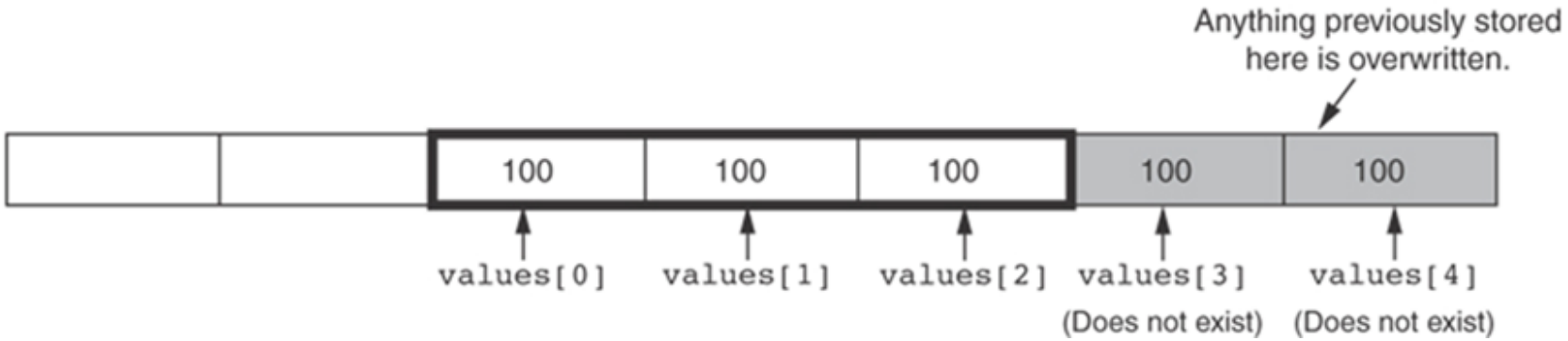
```
 9     const int SIZE = 3;   // Constant for the array size
10     int values[SIZE];     // An array of 3 integers
11     int count;            // Loop counter variable
12
13     // Attempt to store five numbers in the three-element array.
14     cout << "I will store 5 numbers in a 3 element array!\n";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

What the Code Does

The way the `values` array is set up in memory.
The outlined area represents the array.



How the numbers assigned to the array overflow the array's boundaries.
The shaded area is the section of memory illegally written to.



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;
```

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.

Array Initialization

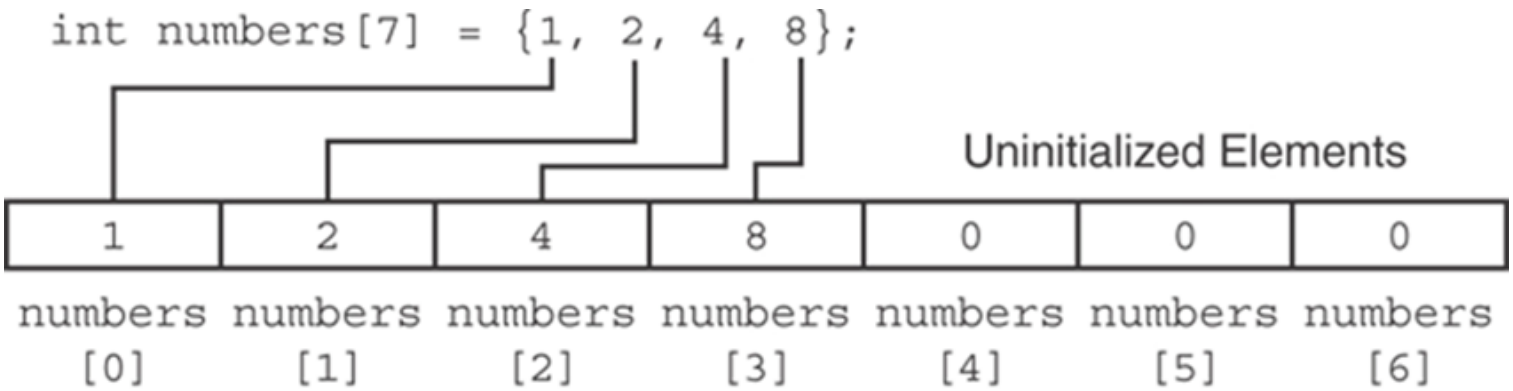
```
7     const int MONTHS = 12;
8     int days[MONTHS] = { 31, 28, 31, 30,
9                          31, 30, 31, 31,
10                         30, 31, 30, 31};
11
12    for (int count = 0; count < MONTHS; count++)
13    {
14       cout << "Month " << (count + 1) << " has ";
15       cout << days[count]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

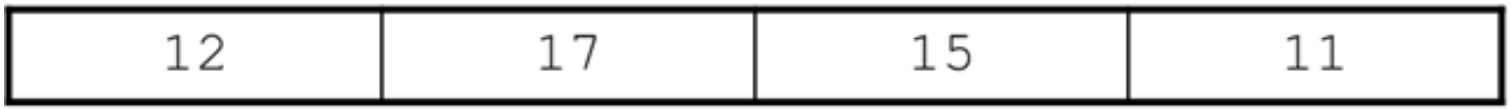
Partial Array Initialization

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



Implicit Array Sizing

- Can determine array size by the size of the initialization list:
  `int quizzes[] = {12,17,15,11};`



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

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;
```

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];
}
```

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];
}
```

Comparing Arrays

- To compare two arrays, you must compare 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
int count = 0; // Loop counter variable
// Compare the two arrays.
while (arraysEqual && count < SIZE)
{
  if (firstArray[count] != secondArray[count])
    arraysEqual = false;
  count++;
}
if (arraysEqual)
  cout << "The arrays are equal.\n";
else   cout << "The arrays are not equal.\n";
```

Exercises

- Challenging problems: 1 - 4.
- Input a sequence of N numbers. List the negative numbers in the sequence.
- Input a sequence of N integers. Count the number of even numbers and calculate the sum of the odd numbers.
- Input a sequence of N numbers and a number K. Count how many elements in the sequence are less than K.
- Input a sequence of N numbers. Reverse the elements in the sequence.
- Input a sequence of N numbers. Find the value(s) that appear most frequently in the sequence.
- Input a sequence of N numbers. Sort these numbers in ascending order.