# Array

# Array

- Array: variable that can store a collection of data of the same type
  - Examples: A list of names, A list of temperatures
- Why do we need arrays?
  - Imagine keeping track of 5 test scores, or 100, or 1000 in memory
    - How would you name all the variables?
    - How would you process each of the variables?

# Declaring an Array

- An array, named test, containing five variables of type int can be declared as

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

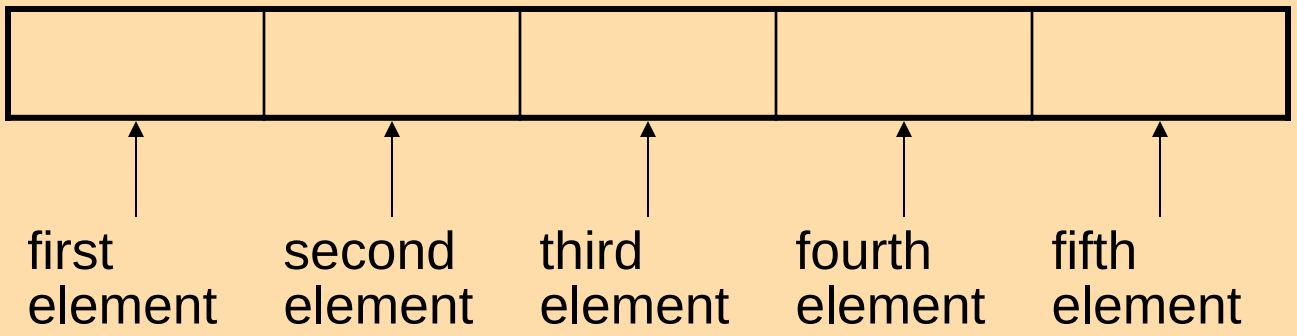
- The value in brackets is called
  - A subscript
  - An index

# Array - Memory Layout

- The definition:

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

allocates the following memory:



# 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`

  `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.

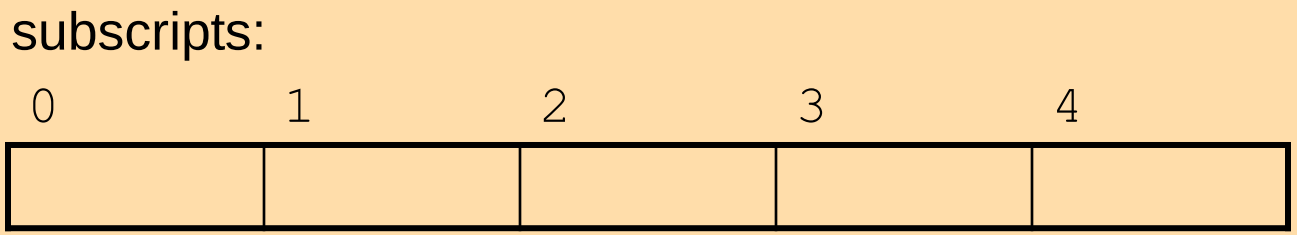
```
const int SIZE = 5;
int tests[SIZE];
```

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

# ACCESSING ARRAY

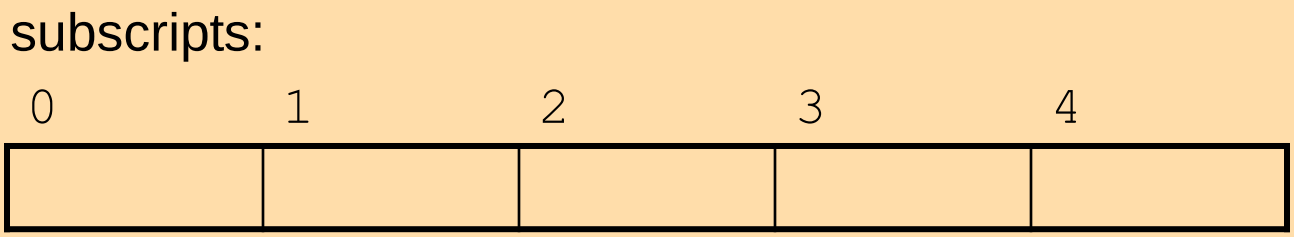
# Accessing Array Elements

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



# Accessing Array Elements

- 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];
```

- However, arrays must be accessed via individual elements:

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

# Accessing Array Elements - example

**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 six employees.
12    cout << "Enter the hours worked by six employees: ";
13    cin >> hours[0];
14    cin >> hours[1];
15    cin >> hours[2];
16    cin >> hours[3];
17    cin >> hours[4];
18    cin >> hours[5];
19
```

*(Program Continues)*

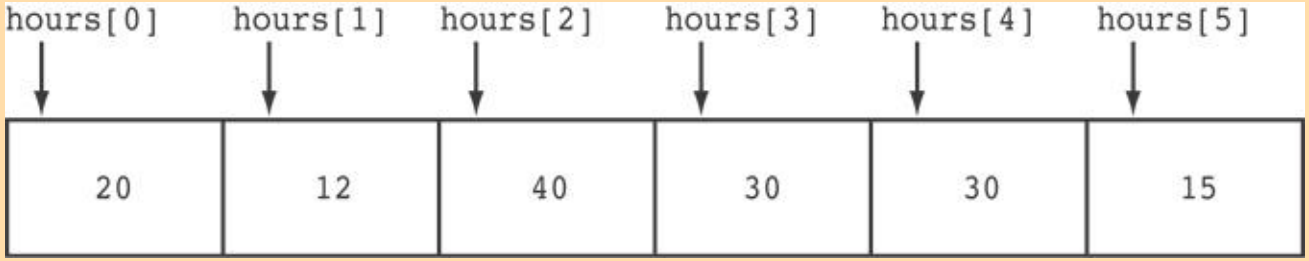
# Accessing Array Elements - example

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

**Program Output with Example Input**

```
Enter the hours worked by six 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

- 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;
```

Must use array with loop

# ARRAY AND LOOP

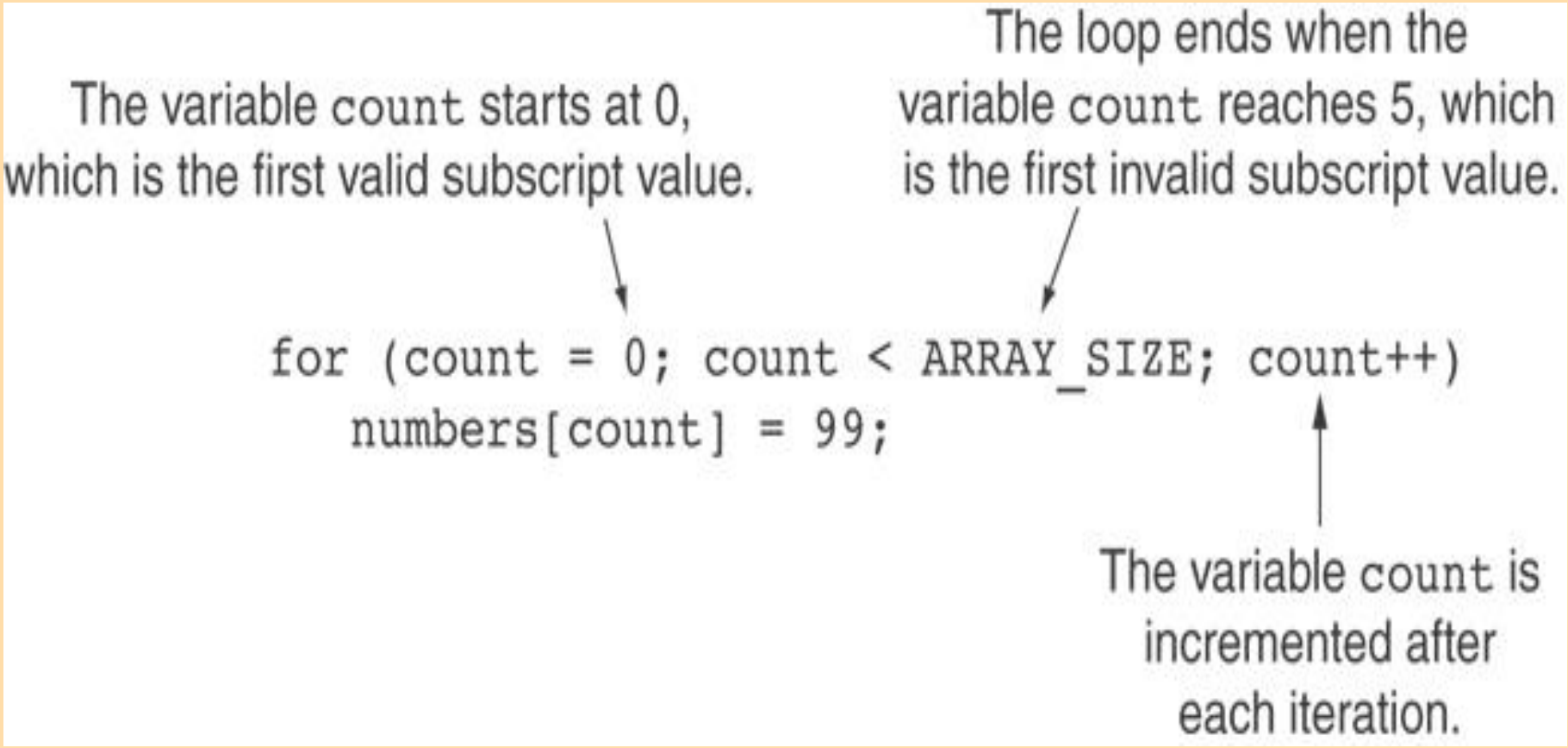
# Using a Loop to Step Through an Array

- Example – 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;
```

# A Closer Look At the Loop



# Default Initialization

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

- Local array → all elements *uninitialized* by default

# In-class Exercise

- Do Lab 12, Exercise 1, no. 1 (pg. 172)
- Do Lab 12, Exercise 1, No. 2 (pg. 172)

Be careful of array bound: invalid subscripts => corrupt memory; cause bugs

# ARRAY AND BOUND CHECKING

# 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.

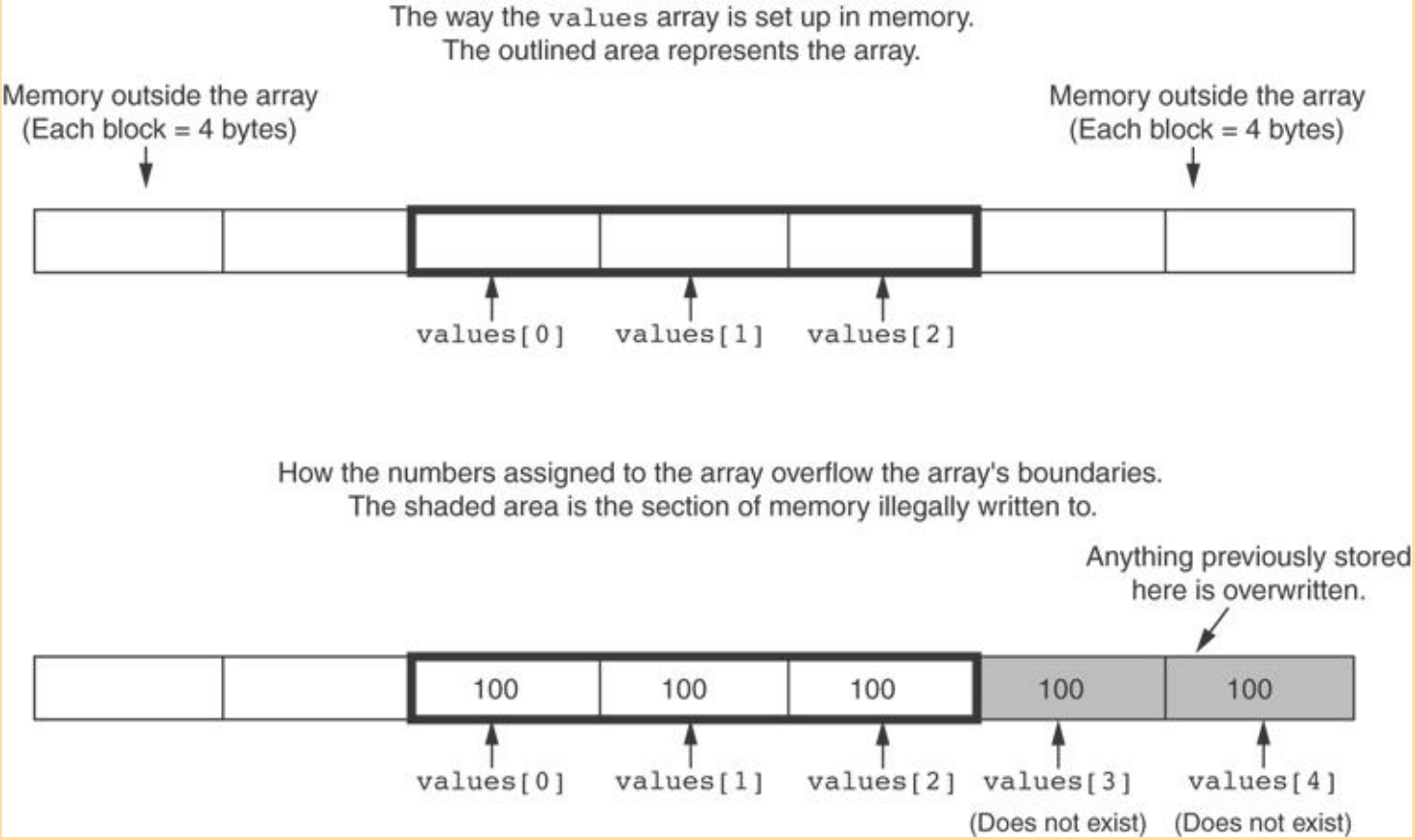
- In other words, you can use subscripts that are beyond the bounds of the array.

# Example

- 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.

# 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.

# Example

```
 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.
```

# Array Initialization

- Valid

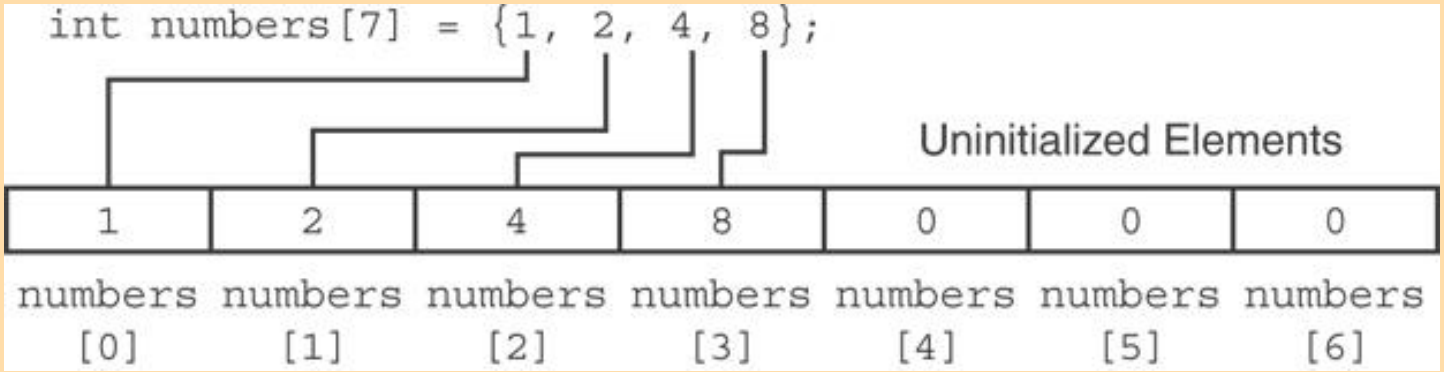
```
int tests[3] = { 3, 5, 11 };
```

- Invalid

```
int tests[3];
tests= { 3, 5, 11 };
```

# Partial Array Initialization

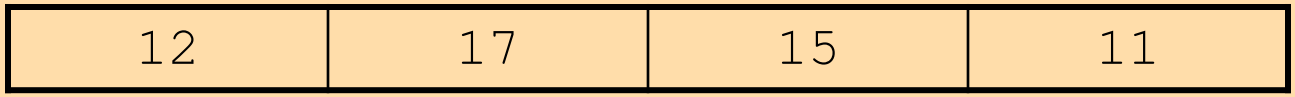
- If array is initialized with fewer initial values than the size declarator, the remaining elements will be set to `0` :



# 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

# Initializing With 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'};
```

# In-Class Exercise

- Are each of the following valid or invalid array definitions? (If a definition is invalid, explain why)

```
int numbers[10] = {0, 0, 1, 0, 0, 1, 0, 0, 1, 1};
int matrix[5] = {1, 2, 3, 4, 5, 6, 7};
double radix[10] = {3.2, 4.7};
int table[7] = {2, , , 27, , 45, 39};
char codes [] = {'A', 'X', '1', '2', 's'};
int blanks[];
char name[6] = "Joanne";
```

- Do Lab 12, Exercise 1, No. 3 (pg. 174)

Arrays and operators; arrays assignment

# PROCESSING ARRAY CONTENTS

# 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];
```