# Chapter 7 Part 1

Arrays

## 7.1

Arrays Hold Multiple Values

## 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:

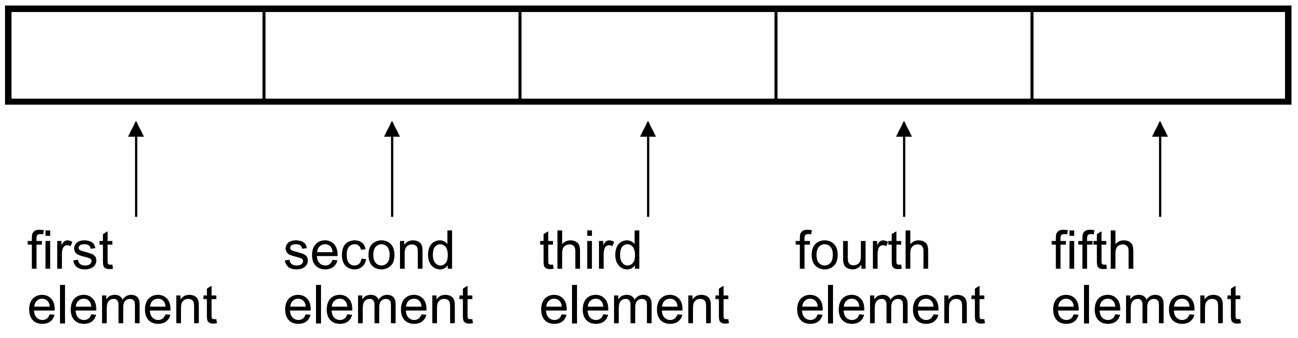
```
int tests[5];
```

## 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 <u>name</u> of the array
- `5`, in `[5]`, is the <u>size declarator</u>. It shows the number of elements in the array.
- The <u>size</u> of an array in memory is (number of elements) * (size of each element)

## Array Terminology

- The <u>size</u> 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];
```

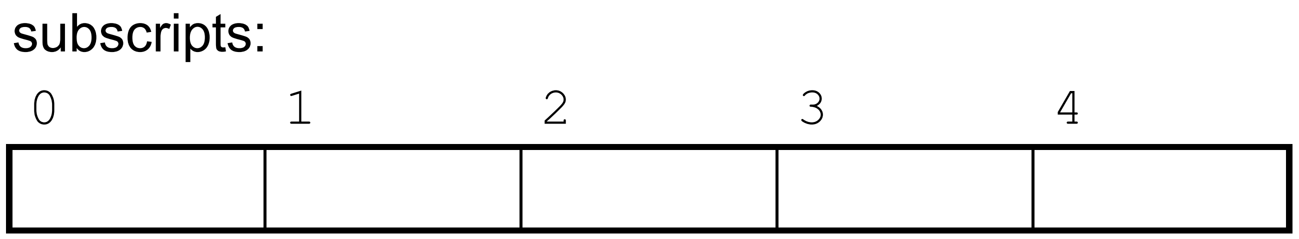
- This eases program maintenance when the size of the array needs to be changed.

# 7.2

Accessing Array Elements

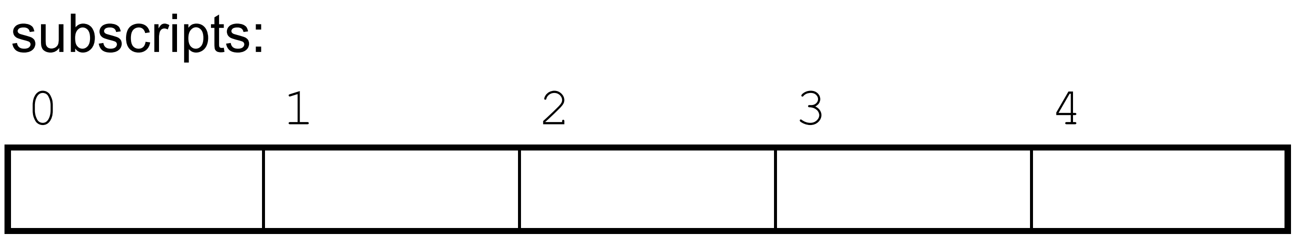
## Accessing Array Elements

- 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];
```

- Arrays must be accessed via individual elements:

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

## Program 7-1

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

int main()
{
   const int NUM_EMPLOYEES = 6;
   int hours[NUM_EMPLOYEES];

   // Get the hours worked by six employees.
   cout << "Enter the hours worked by six employees: ";
   cin >> hours[0];
   cin >> hours[1];
   cin >> hours[2];
   cin >> hours[3];
   cin >> hours[4];
   cin >> hours[5];

```

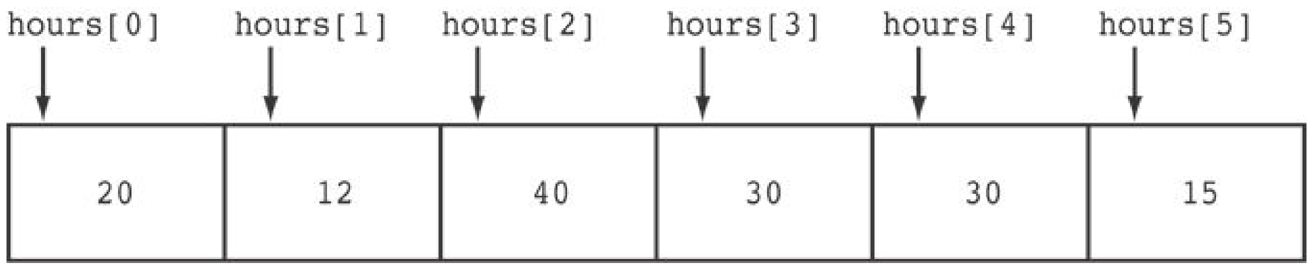
*(Program Continues)*

```
    // Display the values in the array.
    cout << "The hours you entered are:";
    cout << " " << hours[0];
    cout << " " << hours[1];
    cout << " " << hours[2];
    cout << " " << hours[3];
    cout << " " << hours[4];
    cout << " " << hours[5] << endl;
    return 0;
}
```

**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:

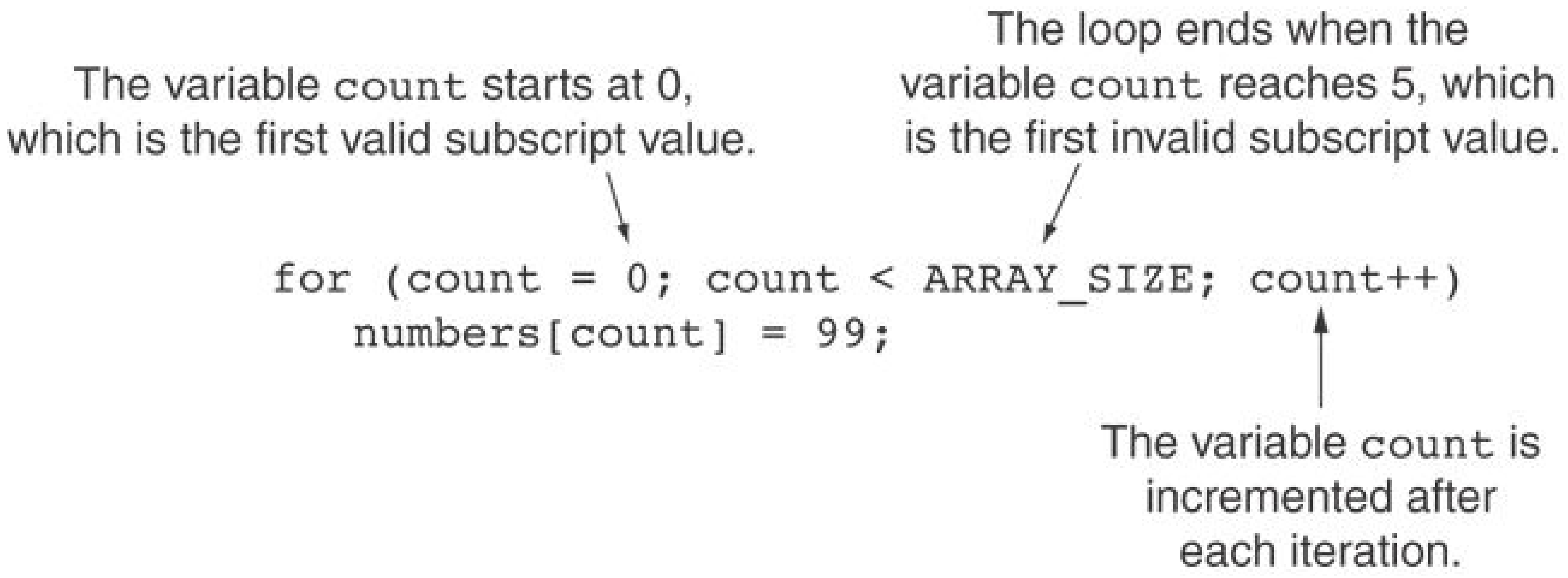
```
int i = 5;
cout << tests[i] << endl;
```

## 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



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] = 99;
```

The variable count is incremented after each iteration.

## Default Initialization

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

- Local array → all elements ***un****initialized* 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.

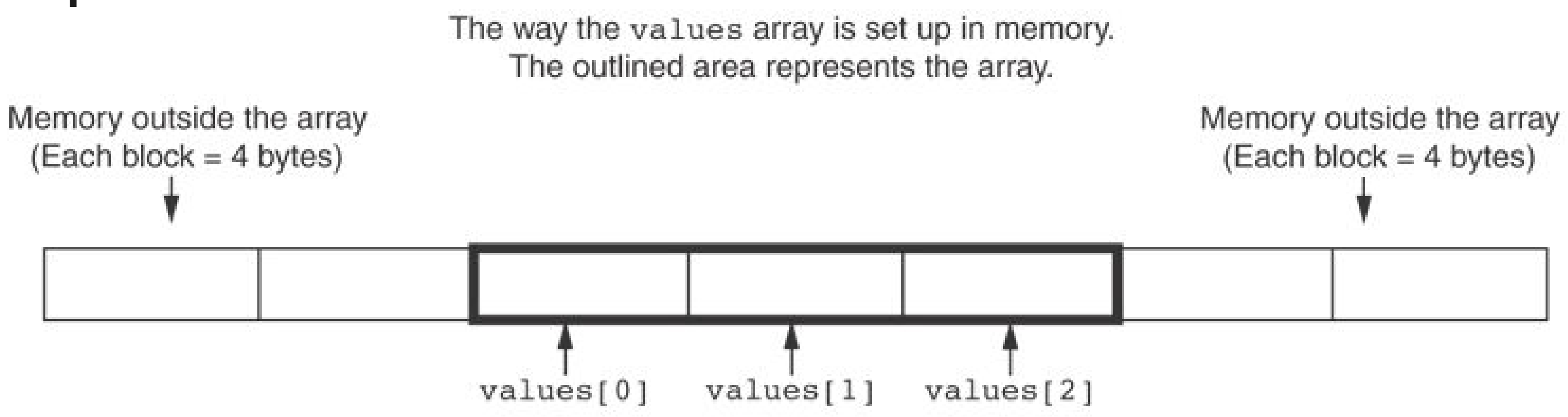
- In other words, you can use subscripts that are beyond the bounds of the array.

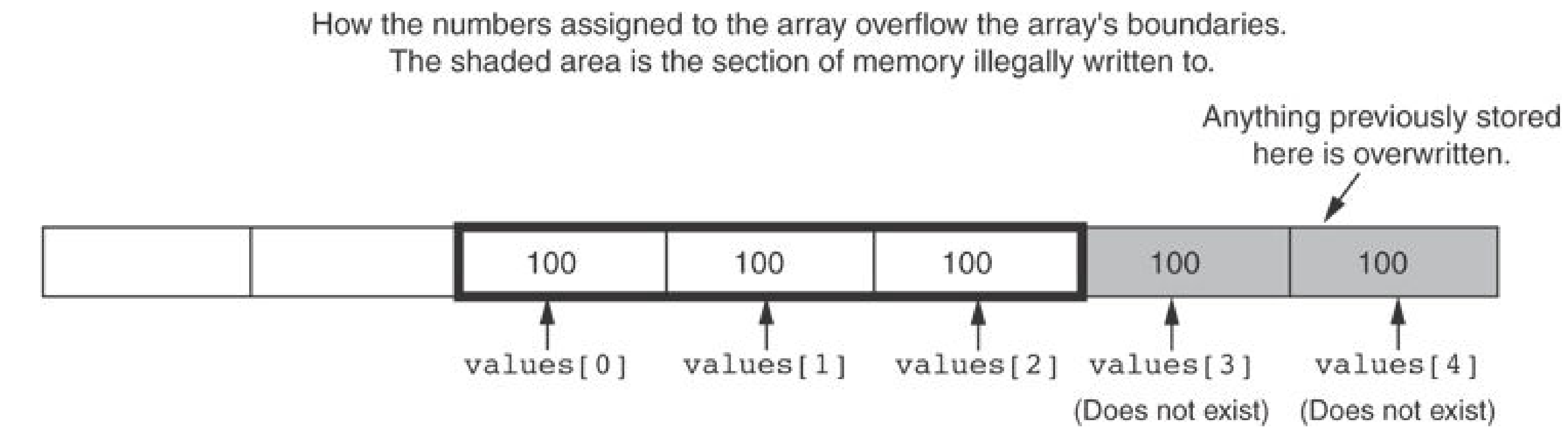
## Code From Program 7-5

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

```
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




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

Array Initialization

## Array Initialization

- Arrays can be initialized with an <u>initialization list</u>:

```
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.

## Code From Program 7-6

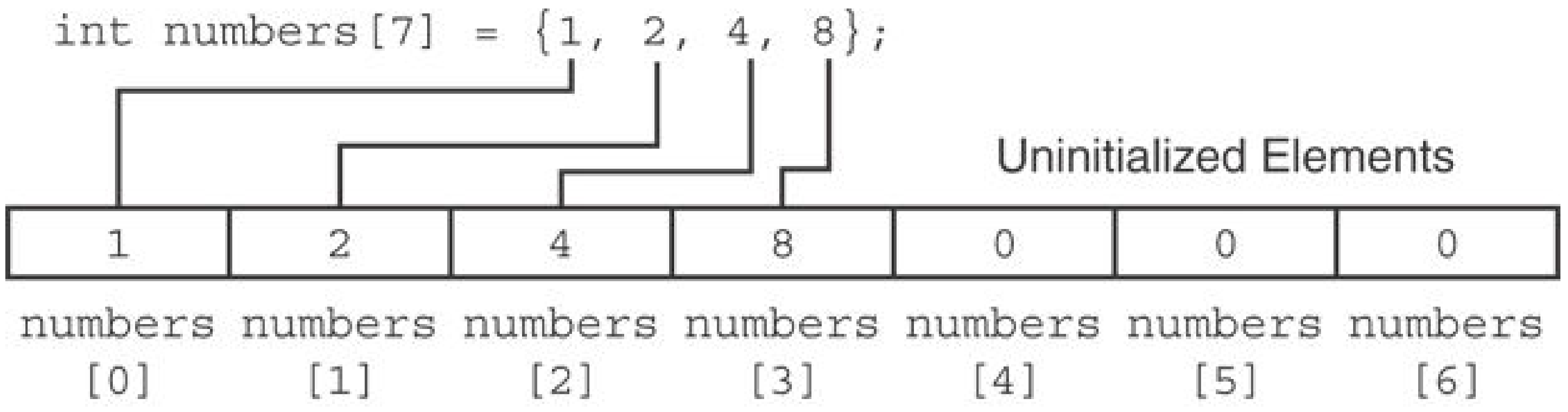
```
const int MONTHS = 12;
int days[MONTHS] = { 31, 28, 31, 30,
                     31, 30, 31, 31,
                     30, 31, 30, 31};

for (int count = 0; count < MONTHS; count++)
{
   cout << "Month " << (count + 1) << " has ";
   cout << days[count] << " days.\n";
}
```

**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};
```

| | | | |
|---|---|---|---|
| 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'};
```

## 7.5

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];
```

## 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 <u>character arrays!</u>

## Printing the Contents of an Array

- 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];
}
```

When this code is finished, the `highest` variable will contains 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 contains 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:

# Comparing Arrays

```
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";
```

# 7.6

Using Parallel Arrays

## 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

```
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;
}
```

**Program 7-12**

```
// This program stores, in an array, the hours worked by 5
// employees who all make the same hourly wage.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   const int NUM_EMPLOYEES = 5;
   int hours[NUM_EMPLOYEES];          // Holds hours worked
   double payRate[NUM_EMPLOYEES];    // Holds pay rates

   // Input the hours worked.
   cout << "Enter the hours worked by " << NUM_EMPLOYEES;
   cout << " employees and their\n";
   cout << "hourly pay rates.\n";
   for (int index = 0; index < NUM_EMPLOYEES; index++)
   {
      cout << "Hours worked by employee #" << (index+1) << ": ";
      cin >> hours[index];
      cout << "Hourly pay rate for employee #" << (index+1) << ": ";
      cin >> payRate[index];
   }

```

*(Program Continues)*

# Program 7-12 *(Continued)*

```
    // Display each employee's gross pay.
    cout << "Here is the gross pay for each employee:\n";
    cout << fixed << showpoint << setprecision(2);
    for (index = 0; index < NUM_EMPLOYEES; index++)
    {
       double grossPay = hours[index] * payRate[index];
       cout << "Employee #" << (index + 1);
       cout << ": $" << grossPay << endl;
    }
    return 0;
}
```

**Program Output with Example Input Shown in Bold**

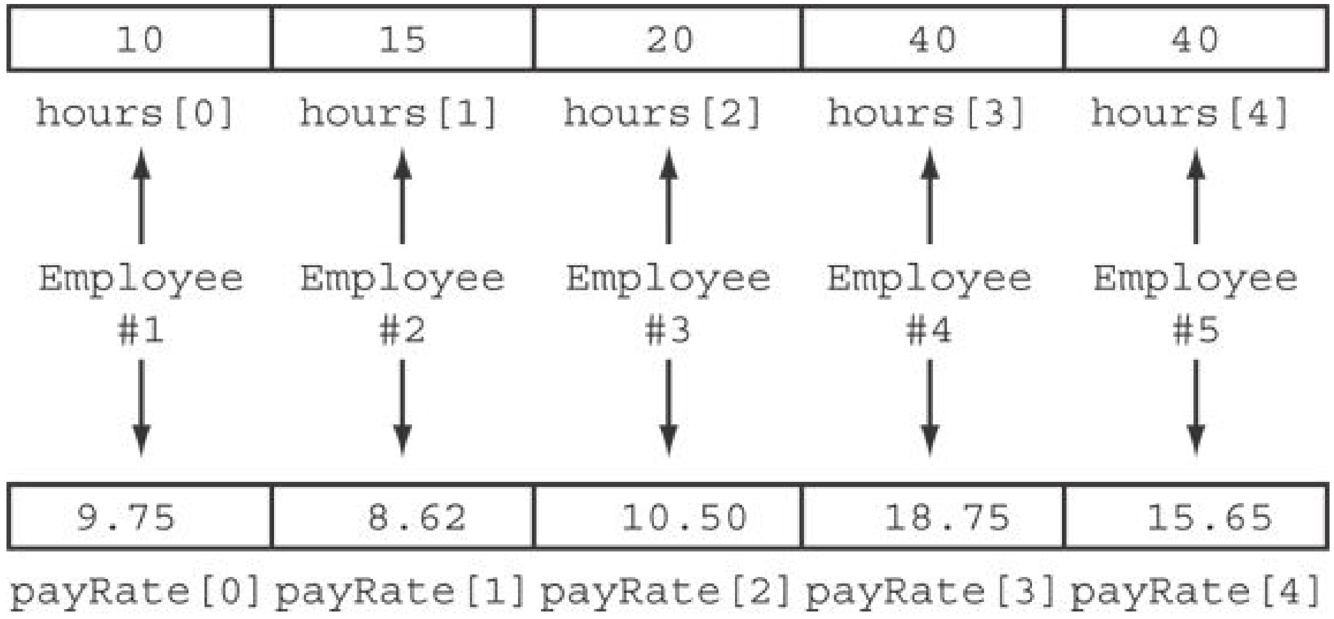
```
Enter the hours worked by 5 employees and their
hourly pay rates.
Hours worked by employee #1: 10 [Enter]
Hourly pay rate for employee #1: 9.75 [Enter]
Hours worked by employee #2: 15 [Enter]
Hourly pay rate for employee #2: 8.62 [Enter]
Hours worked by employee #3: 20 [Enter]
Hourly pay rate for employee #3: 10.50 [Enter]
Hours worked by employee #4: 40 [Enter]
Hourly pay rate for employee #4: 18.75 [Enter]
Hours worked by employee #5: 40 [Enter]
Hourly pay rate for employee #5: 15.65 [Enter]
```

*(program output continues)*

**Program 7-12** *(continued)*

```
Here is the gross pay for each employee:
Employee #1: $97.50
Employee #2: $129.30
Employee #3: $210.00
Employee #4: $750.00
Employee #5: $626.00
```

The `hours` and `payRate` arrays are related through their subscripts:



## 7.7

Arrays as Function Arguments

## 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:

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

## 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 prototype, header:

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

## Passing a C-String to the Function

- When passing a c-string to the function, you do not have to pass the size of the string.
- The end of a c-string is easy to find: it is marked by '\0'.

```
void processString(char [ ] str)
{
        char i=0;
        while( str[i] ! = '\0' )
        {  // do some processing
          i++;
        }
}
```

**Program 7-14**

```
// This program demonstrates an array being passed to a function.
#include <iostream>
using namespace std;

void showValues(int [], int); // Function prototype

int main()
{
   const int ARRAY_SIZE = 8;
   int numbers[ARRAY_SIZE] = {5, 10, 15, 20, 25, 30, 35, 40};

   showValues(numbers, ARRAY_SIZE);
   return 0;
}

```

*(Program Continues)*

## Program 7-14 *(Continued)*

```
//*****************************************************
// Definition of function showValue.                  *
// This function accepts an array of integers and     *
// the array's size as its arguments. The contents    *
// of the array are displayed.                        *
//*****************************************************

void showValues(int nums[], int size)
{
   for (int index = 0; index < size; index++)
      cout << nums[index] << " ";
   cout << endl;
}
```

**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