

## 7.2 Arrays

An array is a group of variables (called **elements** or **components**) containing values that all have the *same* type. Arrays are *objects*, so they're considered *reference types*. As you'll soon see, what we typically think of as an array is actually a *reference* to an array object in memory. The *elements* of an array can be either *primitive types* or *reference types* (including arrays, as we'll see in [Section 7.11](#)). To refer to a particular element in an array, we specify the *name* of the reference to the array and the *position number* of the element in the array. The position number of the element is called the element's **index** or **subscript**.

### Logical Array Representation

[Figure 7.1](#) shows a logical representation of an integer array called `c`. This array contains 12 *elements*. A program refers to any one of these elements with an **array-access expression** that includes the *name* of the array followed by the *index* of the particular element in **square brackets** (`[ ]`). The first element in every array has **index zero** and is sometimes called the **zeroth element**. Thus, the elements of array `c` are `c[0]`, `c[1]`, `c[2]` and so on. The highest index in array `c` is 11,

which is 1 less than 12—the number of elements in the array. Array names follow the same conventions as other variable names.

An index must be a *nonnegative integer* that's less than the array's size. A program can use an expression as an index. For example, if we assume that variable `a` is 5 and variable `b` is 6, then the statement

```
c[a + b] += 2;
```

adds 2 to array element `c[11]`. An indexed array name is an *array-access expression*, which can be used on the left side of an assignment to place a new value into an array element.



## Common Programming Error 7.1

*An index must be an `int` value or a value of a type that can be promoted to `int`—namely, `byte`, `short` or `char`, but not `long`; otherwise, a compilation error occurs.*

Name of array (c) →	c[ 0 ]	-45
	c[ 1 ]	6
	c[ 2 ]	0
	c[ 3 ]	72
	c[ 4 ]	1543
	c[ 5 ]	-89
	c[ 6 ]	0
	c[ 7 ]	62
	c[ 8 ]	-3
	c[ 9 ]	1
	c[ 10 ]	6453
Index (or subscript) of the element in array c →	c[ 11 ]	78

Fig. 7.1

A 12-element array.

#### Description

Let's examine array `c` in Fig. 7.1 more closely. The **name** of the array is `c`. Every array object knows its own length and stores it in a **length instance variable**. The expression `c.length` returns array `c`'s length. Even though the **length** instance variable of an array is **public**, it cannot be changed because it's a **final** variable. This array's 12 elements are referred to as `c[0]`, `c[1]`, `c[2]`, ..., `c[11]`. The value of `c[0]` is -45, the value of `c[1]` is 6, the value of `c[2]` is 0, the value of `c[7]` is 62 and the value of `c[11]` is 78. To calculate the sum of the values contained in

the first three elements of array `c` and store the result in variable `sum`, we would write

```
sum = c[0] + c[1] + c[2];
```

To divide the value of `c[6]` by 2 and assign the result to the variable `x`, we would write

```
x = c[6] / 2;
```