Ordering Information for Deitel C Products:

C How to Program, 3/e

- View *C How to Program's* **Table of Contents**
- Read the **Preface**
- Download the **Code Examples**

To view all the Deitel products and services available, visit the Deitel Kiosk on InformIT at **www.informIT.com/deitel**.

To follow the Deitel publishing program, sign-up now for the *DEITEL™ BUZZ ON-LINE* e-mail newsletter at **www.deitel.com/newsletter/subscribeinformIT.html.**

To learn more about our **C and C++ programming classes** or any other Deitel instructor-led corporate training courses that can be delivered at your location, visit **www.deitel.com/training** or contact our Director of Corporate Training Programs at (978) 461-5880 or e-mail: **christi.kelsey@deitel.com**.

Note from the Authors: This DEITEL™ article is an excerpt from Chapter 7, Section 7.7 of *C How to Program, 3/e* and describes pointer expressions and pointer arithmetic. Readers should be familiar with basic C programming concepts and pointers. The code example included in this articles teaches programming through the Deitel™ signature *LIVE-CODE™ Approach*, which presents all concepts in the context of complete working programs followed by the screen shots of the actual inputs and outputs.

7.7 Pointer Expressions and Pointer Arithmetic

Pointers are valid operands in arithmetic expressions, assignment expressions, and comparison expressions. However, not all the operators normally used in these expressions are valid in conjunction with pointer variables. This section describes the operators that can have pointers as operands, and how these operators are used.

A limited set of operations may be performed on pointers. A pointer may be incremented (**++**) or decremented (**--**), an integer may be added to a pointer (**+** or **+=**), an integer may be subtracted from a pointer (**-** or **-=**), or one pointer may be subtracted from another.

Assume that array **int v[10]** has been declared and its first element is at location **3000** in memory. Assume pointer **vPtr** has been initialized to point to **v[0]**, i.e., the value of **vPtr** is **3000**. Figure 7.18 diagrams this situation for a machine with 4-byte integers. Note that **vPtr** can be initialized to point to array **v** with either of the statements

```
vPtr = v;
vPtr = &v[ 0 ];
```

In conventional arithmetic, the addition **3000 + 2** yields the value **3002**. This is normally not the case with pointer arithmetic. When an integer is added to or subtracted from a pointer, the pointer is not simply incremented or decremented by that integer, but by that integer times the size of the object to which the pointer refers. The number of bytes depend on the object's data type. For example, the statement

```
vPtr += 2;
```

would produce **3008** (**3000 + 2 * 4**) assuming an integer is stored in 4 bytes of memory. In the array **v**, **vPtr** would now point to **v[2]** (Fig. 7.19). If an integer is stored in 2 bytes of memory, then the preceding calculation would result in memory location **3004** (**3000 + 2 * 2**). If the array were of a different data type, the preceding statement would increment the pointer by twice the number of bytes that it takes to store an object of that data type. When performing pointer arithmetic on a character array, the results will be consistent with regular arithmetic because each character is one byte long.

Portability Tip 7.1

Most computers today have 2-byte or 4-byte integers. Some of the newer machines use 8-byte integers. Because the results of pointer arithmetic depend on the size of the objects a pointer points to, pointer arithmetic is machine dependent.

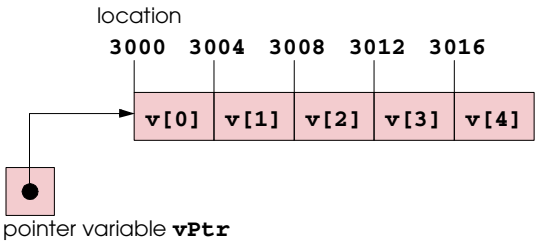

Fig. 7.18 The array **v** and a pointer variable **vPtr** that points to **v**.

If **vPtr** had been incremented to **3016**, which points to **v[4]**, the statement

```
vPtr -= 4;
```

would set **vPtr** back to **3000**—the beginning of the array. If a pointer is being incremented or decremented by one, the increment (**++**) and decrement (**--**) operators can be used. Either of the statements

```
++vPtr;
vPtr++;
```

increment the pointer to point to the next location in the array. Either of the statements

```
--vPtr;
vPtr--;
```

decrement the pointer to point to the previous element of the array.

Pointer variables may be subtracted from one another. For example, if **vPtr** contains the location **3000**, and **v2Ptr** contains the address **3008**, the statement

```
x = v2Ptr - vPtr;
```

would assign to **x** the number of array elements from **vPtr** to **v2Ptr**, in this case, **2**. Pointer arithmetic is meaningless unless performed on an array. We can not assume that two variables of the same type are stored contiguously in memory unless they are adjacent elements of an array.

Common Programming Error 7.1

Using pointer arithmetic on a pointer that does not refer to an array of values.

Common Programming Error 7.2

Subtracting or comparing two pointers that do not refer to the same array.

Common Programming Error 7.3

Running off either end of an array when using pointer arithmetic.

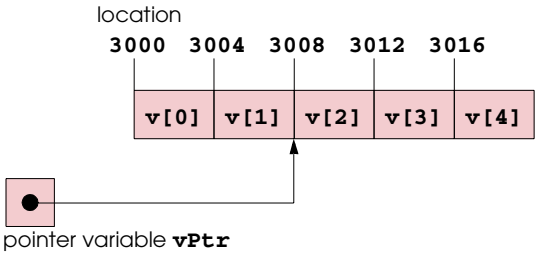

Fig. 7.19 The pointer **vPtr** after pointer arithmetic.

A pointer can be assigned to another pointer if both pointers are of the same type. Otherwise, a cast operator must be used to convert the pointer on the right of the assignment to the pointer type on the left of the assignment. The exception to this rule is the pointer to **`void`** (i.e., **`void *`**) which is a generic pointer that can represent any pointer type. All pointer types can be assigned a pointer to **`void`**, and a pointer to **`void`** can be assigned a pointer of any type. In both cases, a cast operation is not required.

A pointer to **`void`** cannot be dereferenced. For example, the compiler knows that a pointer to **`int`** refers to four bytes of memory on a machine with 4-byte integers, but a pointer to **`void`** simply contains a memory location for an unknown data type—the precise number of bytes to which the pointer refers is not known by the compiler. The compiler must know the data type to determine the number of bytes to be dereferenced for a particular pointer.

Common Programming Error 7.4

Assigning a pointer of one type to a pointer of another type if neither is of type ***`void *`*** *causes a syntax error.*

Common Programming Error 7.5

Dereferencing a ***`void *`*** *pointer.*

Pointers can be compared using equality and relational operators, but such comparisons are meaningless unless the pointers point to members of the same array. Pointer comparisons compare the addresses stored in the pointers. A comparison of two pointers pointing to the same array could show, for example, that one pointer points to a higher-numbered element of the array than the other pointer does. A common use of pointer comparison is determining whether a pointer is **`NULL`**.