Ordering Information:

Perl How to Program
The Complete Perl Training Course

- View the complete **Table of Contents**
- 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 **Perl programming courses** or any other Deitel instructor-led corporate training courses that can be delivered at your location, visit **www.deitel.com/training**, contact our Director of Corporate Training Programs at (978) 461-5880 or e-mail: **christi.kelsey@deitel.com**.

Note from the Authors: This article is an excerpt from Chapter 4, Sections 4.12 and 4.13 of *Perl How to Program* and introduces the hash data structure, used for storing data in key-value pairs. We provide readers with two examples of creating and manipulating hashes. This article is intended for beginning programmers and readers should be familiar with basic Perl syntax and arrays. The code examples included in this article show readers programming examples using 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.

4.12 Introduction to Hashes

The second multivalued (nonscalar) data type that is fundamental to the Perl language is the *hash*, or *associative array*. A hash is an unordered collection of *key-value pairs*. Rather than accessing a hash element with a subscript like an array, elements in a hash are accessed using a string known as a *key*. Each *key* must be unique. Hashes are sometimes known as associative arrays, because they define associations between keys and their values.

Common Programming Error 4.12

The keys in a hash must be unique. Using the same key more than once in a hash causes the original value for that key to be replaced with a new value; this situation could be a logic error, or it could be a normal update operation.

The special symbol for a hash is **%**. Hashes are implementations of a data structure known as a *hash table*. Because each value has an associated key, hashes take up more space than arrays. However, the internal structure of a hash provides fast lookup capabilities through which a value can often be located in one operation.

Performance Tip 4.4

Hashes are particularly useful in situations where extremely fast retrieval of values is required.

Performance Tip 4.5

Hashes require more memory than arrays. Often, there is a delicate balance between the time it takes to locate values and the amount of memory required to maintain the data. These issues are normally categorized as space-time trade-offs.

4.13 Creating and Manipulating a Hash

Like arrays, hashes can be created in two ways: either by assigning a list to the hash variable or by assigning values to single elements. Figure 4.15 shows the creation of a hash and the accessing of its elements; watch for the **=>** operator, which we have not covered yet.

```
1  #!/usr/bin/perl
2  # Fig. 4.15: fig04_15.pl
3  # Creating and accessing hash elements
4
5  # create a hash and output its values
6  %hash = ( width => '300',
7            height => '150' );
8  print "\$hash{ 'width' } = $hash{ 'width' }\n";
9  print "\$hash{ 'height' } = $hash{ 'height' }\n\n";
10
11 # assigning to a new hash element
12 $hash{ 'color' } = 'blue';
13 print "\$hash{ 'width' } = $hash{ 'width' }\n";
14 print "\$hash{ 'height' } = $hash{ 'height' }\n";
15 print "\$hash{ 'color' } = $hash{ 'color' }\n\n";
16
```

Fig. 4.15 Creating and using hashes.

```
17  # display a hash with print
18  print "%hash\n";       # no interpolation, unlike with arrays
19  print %hash, "\n";     # difficult to read, no spaces
```

```
$hash{ 'width' } = 300
$hash{ 'height' } = 150

$hash{ 'width' } = 300
$hash{ 'height' } = 150
$hash{ 'color' } = blue

%hash
height150width300colorblue
```

Fig. 4.15 Creating and using hashes.

Lines 6–7 define a hash by assigning a list to a hash variable (**%hash**). As with arrays, a hash variable can have any valid identifier as a name, including the name **hash**, used here. Here, we introduce the *"corresponds to" operator*, **=>**. This operator is similar to the comma operator, except that it interprets its left-side operand as a string, so no quotes need to be placed around the string. The first value in this list (**width**) is the key of the first element to be created in the hash, and the second value in the list (**'300'**) is the value that corresponds to that key. The list elements are grouped this way into key-value pairs. So, the value **height** is the key for the second element in the hash, and the element's corresponding value is **'150'**. Lines 8–9 display the individual hash elements. Note that you can access a key's corresponding value by preceding the hash name with a **$** and enclosing the key in curly braces (**{}**). This expression "looks up" the value that corresponds to the key and returns that value.

Line 12

```
$hash{ 'color' } = 'blue';
```

adds a new element to the existing hash. Assigning a value to a new key in a hash automatically creates a new element in that hash. Individual hash elements are accessed in a similar manner to arrays, except that the subscript is now the key surrounded by curly braces, as shown in line 12. The technique of adding new elements to a hash can be used to create a new hash as well. Note that if a particular key already exists in the hash, assigning a value to that key replaces the old value with the new one. Lines 13–15 display the updated contents of **%hash**.

Lines 18–19

```
print "%hash\n";       # no interpolation, as with arrays
print %hash, "\n";     # difficult to read, no spaces
```

attempt to output **%hash** using techniques we demonstrated for arrays earlier in this chapter. Unlike arrays, hashes are not interpolated when enclosed in double quotes. So, line 18 simply displays the string **%hash**. Outputting the hash with **print** (line 19) concatenates all the key-value pairs and outputs them as one long string. Note that the key-value pairs do not appear in a way that indicates the order in which they were added to the hash.

Just as arrays have array slices, hashes have *hash slices* (Fig. 4.16). Providing multiple keys in curly braces (**{}**) returns a list of the corresponding values for those keys. The resulting list is manipulated as an array of values rather than a key-value pairs, so the **@** symbol is used for hash slices as well as array slices.

Common Programming Error 4.13

*Using parentheses, **()**, rather than braces, **{}**, when creating a hash slice is a syntax error.*

Lines 5–14 create hash **%romanNumerals**, where the keys are English words representing the numbers from 1–10 and the values are the Roman numerals representing these numbers. Note that the keys are not placed in quotes because they appear to the left of the "corresponds to" operator. Lines 16–17 display the results of a hash slice containing the values for the keys **'three'**, **'five'** and **'eight'**.

```
1  #!/usr/bin/perl
2  # Fig. 4.16: fig04_16.pl
3  # Demonstrating hash slices.
4
5  %romanNumerals = ( one   => 'I',
6                     two   => 'II',
7                     three => 'III',
8                     four  => 'IV',
9                     five  => 'V',
10                    six   => 'VI',
11                    seven => 'VII',
12                    eight => 'VIII',
13                    nine  => 'IX',
14                    ten   => 'X' );
15
16 print "The Roman numerals for three, five and eight are: ",
17       "@romanNumerals{ 'three', 'five', 'eight' }\n";
```

```
The Roman numerals for three, five and eight are: III V VIII
```

Fig. 4.16 Demonstrating hash slices.