The journal of Apple technology.

Volume Number: 18 (2002)
Issue Number: 9
Column Tag: Section 7

A Bit More Perl

control flow, data structure, resources...

by Rich Morin

As promised last month, this column will go a bit deeper into Perl, looking at its data structures and control flow operators, closing out with a discussion of Perl resources.

Perl only has a few data types, but their versatility allows them to serve in a large number of roles. In several years of Perl programming, I've never found myself reaching for a data type (or anything else, really :-) that Perl didn't have.

Scalars

The "scalar" is Perl's basic data element. A scalar can be an integer, a floating-point number, a text string (usually ASCII, but other encodings are allowed), or a "reference" to some other type of entity. With a bit of work, scalars can be mapped onto bit strings or even used to represent sets of alternative quantum states. As Perl's slogan says: There's More Than One Way To Do It.

Because strings are "first-class citizens" in Perl, they are used in most situations where a C programmer might use an array of characters. In fact, if I see an array of characters being used in a Perl program, my first assumption is that a C programmer has been unable or unwilling to learn about Perl's string manipulation facilities! Here are some examples of Perl scalars:

```
% cat x0
#!/usr/bin/env perl
$num_1 = 123;
$num_2 = 123.45;
$n1_r1 = \$num_1;
$n1_r2 = "num_1";
printf("num_1=%d, num_2=%f\n", $num_1, $num_2);
printf("n1_r1=%d, n1_r2=%f\n", $$n1_r1, $$n1_r2);
$str_1 = '123';
$str_2 = "The value of \$num_1 is $num_1.";
print "The value of \$str_1 is |$str_2|.\n";
% x0
num_1=123, num_2=123.450000
n1_r1=123, n1_r2=123.000000
The value of $str_1 is |The value of $num_1 is 123.|.
```

Note that Perl allows both "hard" and "symbolic" references. $n1_r1 (a hard reference) will run a bit faster than

$n1_r2 (a symbolic reference). The naming and behavior are taken from "hard" and "symbolic" links, as used in BSD file systems.

Aggregates

Perl has only two aggregate data structures: arrays and hashes. As indicated above, however, they are quite powerful. Perl arrays can be subscripted (from either end!), used as queues, stacks, or deques (double-ended queues), and more. If you're looking for a way to store an ordered collection, an array will probably serve your needs.

Perl's hashes handle unordered collections of data, storing each value under a unique (scalar) key. They are quite similar to the tables one finds in relational database systems; in fact, hashes can be "tied" to database table to provide persistent storage.

```
% cat x1
#!/usr/bin/env perl
@A = (1, 'deux', 'III', 4);
print "$A[0], $A[1], $A[-2], $A[-1]\n";
%H = (uno => 1, dos => 2);
printf("sum = %d\n", $H{uno} + $H{dos});
% x1
1, deux, III, 4
sum = 3
```

Although Perl's arrays and hashes can only contain scalars, multi-level data structures (as well as trees, graphs, etc.) can be formed by using references. For convenience, Perl supports abbreviated ways to use these complex forms. Here are some multi-level structures:

```
$AAA[1][2][3] = 1;      # 3-dimensional array
$HHH{a}{b}{c} = 2;      # 3-dimensional hash
$AH[0]{a}     = 3;      # array of hashes
$HA{b}[1]     = 4;      # hash of arrays
$AHA[2]{c}[3] = 5;      # array of ...
$HAHA{d}[4]{e}[5]++;    # hash of ...
```

Control Flow

Although Perl supports the traditional C-style "for" loop, it isn't used much in practice. Instead, Perl programmers tend to use list-based looping, as:

```
foreach $item (@A) { ...
foreach $key (sort(keys(%H))) { ...
```

Explicit references can be useful in dealing with multi-level data structures:

```
% cat x2
#!/usr/bin/env perl
$HH{a}{b} = 'ab';
$HH{c}{d} = 'cd';
foreach $k1 (sort(keys(%HH))) {
    $r1 = $HH{$k1};
    foreach $k2 (sort(keys(%{$r1}))) {
        $tmp = $r1->{$k2};
        print "\$HH{$k1}{$k2}=$tmp\n";
    }
}
% x2
$HH{a}{b}=ab
$HH{c}{d}=cd
```

Perl also provides the until and while looping operators, along with a plethora of ways to do conditional execution:

```
% cat x3
#!/usr/bin/env perl
$foo = 'bar';
if     ($foo eq 'bar') { print '0 '; };
unless ($foo ne 'bar') { print '1 '; };
print '2 ' if     ($foo eq 'bar');
print "3 " unless ($foo ne bar);
$foo eq 'bar' and print '4 ';
$foo ne 'bar' or  print '5 ';
print "\n";
% x3
0 1 2 3 4 5
```

Perl also provides GOTOs and loop modifiers, subroutines (anonymous and/or recursive, if need be), exception handling, and some even trickier facilities. It also has tightly integrated regular expressions, allowing (parts of) strings to be matched, extracted, and/or modified. In short, Perl is a powerful language, suited for everything from short one-offs to substantial applications.

Resources

Perl has a very active user community, providing a variety of forums for communication. Whether you prefer conferences, IRC channels, local user group meetings, mailing lists, newsletters, usenet newsgroups, or weblogs, Perl has it. To find these resources, start at www.perl.{org,com}, the primary sources of Perl information.

Perl has enormous amounts of online documentation, Typing man perl will yield a list of several dozen subsidiary man pages, along with some advice on how to approach them. Typing perldoc will lead you into a function index and an online FAQ. Spend some time scanning through these; it will pay off handsomely later on...

Finally, of course, there are literally dozens of books on Perl, ranging from introductory and overview texts to detailed coverage of specialized subtopics. The majority of Perl books are published by O'Reilly and Associates (www.oreilly.com), who also operates www.perl.com. O'Reilly's Perl books tend to be authoritative, diverse, readable, and practical; if you had to pick a single publisher of Perl books, you could stick to O'Reilly and survive quite nicely. And, if you could only buy one book on Perl, their Programming Perl would be the clear winner.

Fortunately, you don't have to restrict yourself to a single publisher, let alone a single book. My Perl collection, for instance, includes fine volumes by Addison-Wesley, Manning, and Wiley. So, look around a bit! That said, here is a "reading list" of Perl books for the typical MacTech reader. Be warned; the books get significantly more chewy as the list goes on:

- Elements of Programming with Perl - Johnson (Manning)
- Learning Perl - Schwartz & Christiansen (O'Reilly)

- Programming Perl - Wall, et al (O'Reilly)
- Effective Perl Programming - Hall (Addison-Wesley)
- Object-Oriented Perl - Conway (Manning)
- Mastering Algorithms with Perl - Orwant, et al (O'Reilly)
- Advanced Perl Programming - Srinivasan (O'Reilly)
- Mastering Regular Expressions - Friedl (O'Reilly)

Here are some reference books that you might want to add in:

- Perl in a Nutshell - Siever, et al (O'Reilly)
- Perl Cookbook - Christiansen & Torkington (O'Reilly)

If you're doing web programming in Perl, consider getting:

- CGI Programming with Perl - Guelich, et al (O'Reilly)
- Network Programming with Perl - Stein (Addison-Wesley)
- Official Guide to Programming with CGI.pm - Stein (Wiley)
- Perl & LWP - Burke (O'Reilly)
- Programming Web Graphics with Perl & GNU Software - Wallace (O'Reilly)
- Web Client Programming with Perl - Wong (O'Reilly)
- Writing Apache Modules with Perl and C - Stein & MacEachern (O'Reilly)

Finally, if you're using any Perl add-ons, you should consider books such as:

- Learning Perl/Tk - Walsh (O'Reilly)
- Mastering Perl/Tk - Lidie & Walsh (O'Reilly)
- Perl & XML - Ray & McIntosh (O'Reilly)
- Programming the Perl DBI - Descartes and Bunce (O'Reilly)

Rich Morin has been using computers since 1970, Unix since 1983, and Mac-based Unix since 1986 (when he helped Apple create A/UX 1.0). When he isn't writing this column, Rich runs Prime Time Freeware (www.ptf.com), a publisher of books and CD-ROMs for the Free and Open Source software community. Feel free to write to Rich at rdm@ptf.com.