

Re: Readline using foreach and while

Source: <http://coding.derkeiler.com/Archive/Perl/comp.lang.perl.misc/2008-03/msg01350.html>

- *From:* "szz" <szzRE@xxxxxxxxxxxxxxxxxx>
 - *Date:* Sat, 29 Mar 2008 08:47:34 -0700
-

Ben Morrow wrote:

Quoth "szz" <szzRE@xxxxxxxxxxxxxxxxxx>:

Ben Morrow wrote:

Quoth "szz" <szzRE@xxxxxxxxxxxxxxxxxx>:

Actually the behaviors of "for (@ary)" and "for (@ary, ())" do seem constant if you really think about it. The resulting list is what it iterates over (from the first element, to what ever *count* is... in the former case *count* come from the array, and since the condition is checked at the start of each iteration, if the array is added to, the count is incremented.

In the latter case, a new list is created from contents of @ary + an empty list, which gives you a new list, which contains the values of @ary, but is a new separate list, and thus is not effected by changes to @ary because it has it's own copy of @ary's values.

OK, now explain to me why

```
my @ary = qw/a b c/;
print map { /c/ and push @ary, 'd'; $_ } @ary;
```

doesn't work like that :).

Re: Readline using foreach and while

Actually it does. The difference is, map doesn't recheck the count every time around like for/foreach do. If you print the contents of @ary after the line with the map statement, it does indeed contain 'd' at the end. This behavior seems to correct, as one would likely expect that the list map returns when it is finished to be the same length as the one /passed/ into map at the start. If you pass a 3 element list, you should get back a 3 element list, should you not?

Absolutely not. my %h = map { \$_ => 1 } qw/a b c/; is quite a common idiom.

Well, you're still getting that many *sets* which is probably what I should of said, or have been clearer. In that example, you get 3 set of hash pairs resulting from the 3 element list. The point is you get count_of_passed_list amount of something from the map, in some form or another. How exactly it's returned is determined by the template inside the map.

<snip>

\$_ is aliased to the current array element just like in for. Again, the only difference I see if that map doesn't recheck the count of the passed list for each iteration.

No, you're misunderstanding the difference between a list and an array. Evaluating an array in list context returns a list of its elements *as they are now*;

I seem to understand it just fine. What we both said above seems to be true. Maybe we're just misunderstanding what the other is trying to say? :-)

under most circumstances, it returns a list of aliases to those elements, but any changes to the order of the elements in @ary are not propagated into the list. Consider

```
my @ary = qw/a b c/;
sub foo {
  my @keep = map "$_", @_; # kill the aliasing
```

Ok @keep now contains (a, b, c)...

Re: Readline using foreach and while

```
unshift @ary, 'h';
```

@ary, which comes from a scope outside this sub, now contains (h, a, b, c)

```
$_[$_] .= $keep[$_] for 0..$#_;
```

Keep in mind `$_[0]` *still* points to what used to be the first element of @ary. Remember, the aliasing isn't to the *array* but to its *elements*. This is because when you normally pass args to a sub (e.g., `do_something($x, $y);`), the aliasing is with `$x` and `$y` to `$_[0]` and `$_[1]`. Passing an array just like passing that many scalars as are elements in the array; each individual one gets aliased to the next sequential element of `@_` in the sub's scope.

```
}  
foo @ary;  
print for @ary;
```

Running this (with the last line as ``print "$_\n" for @ary;`` for clarity) prints:

```
h  
aa  
bb  
cc
```

Seems the changes propagated just fine to me. You pushed 'h' to the beginning of @ary, then you effectively iterated from `$ary[1]..$ary[$#ary]`.

Map is the same way in that regard; `$_` is an alias to an *element*

To me, this behavior is part of what separates map from for/foreach.

It separates for (LIST) from everything else that accepts a LIST. This is why I called it 'weird'.

I still don't see what you consider weird about it. What does it do that you don't expect it to do?

```
--  
szr
```

Re: Readline using foreach and while