

CSE2395/CSE3395


Topic 11

References

Programming Perl 3rd edition
chapters 8-9, pages 242-287

Programming Perl 2nd edition
chapter 4, pages 243-275

`perlref`, `perlrefut`, `perllo`, `perldsc`
`manpages`



1

Last time

Covered in Topic 10

- The World Wide Web model
 - HTTP and HTML
- The Common Gateway Interface (CGI) model
 - interactive documents on the World Wide Web
- Installing and running CGI programs
- The `CGI.pm` module
- Passing parameters to CGI programs
 - submitting forms
- Generating HTML

CSE2395/CSE3395

2

To be covered today

- References
 - Perl's answer to pointers
- Making references
- Using references
- Nested data structures
 - multi-dimensional arrays
 - emulating `C structs`

CSE2395/CSE3395

3

Documentation

References on references

- *Programming Perl* (camel book)
 - references shown throughout these notes
- *Advanced Perl Programming*
 - Sriram Srinivasan, O'Reilly & Associates
 - chapters 1-2, pages 1-37
- `manpages`
 - `perlref`
 - complete but terse explanation of references
 - `perlrefut`
 - easy tutorial of similar complexity as these notes
 - `perllo`
 - managing nested data structures (lists of lists)
 - `perldsc`
 - data structures cookbook, more detail than `perllo`

CSE2395/CSE3395


4

Scalars

Revision

- Scalars can be of three types
 - number
 - string
 - reference
- Only hard references described here
 - symbolic ("soft") references not covered

CSE2395/CSE3395

 *Camel3* pages 58-59; *Camel2* pages 38-39
`perlref` manpage


5

References

Definitions and behaviour

- A reference is a scalar value that refers to another scalar, array or hash
 - or filehandle or function or . . .
- References are like C pointers
 - contain address of referred-to value
 - know the type of thing they refer to (scalar, array, . . .)
- But . . .
 - references can never refer to uninitialized memory
 - can't do pointer arithmetic on a reference
 - can't cast a reference to refer to a different type

CSE2395/CSE3395

 *Camel3* pages 242-244; *Camel2* pages 244-245
`perlref` manpage

6

Uses of references

When you need references

- Complex data structures
 - arrays can contain only scalars, not other arrays
 - but a reference to an array is a scalar
 - arrays can contain references to arrays
 - this is how multi-dimensional arrays are done in Perl
 - the above also applies to hashes
- Passing values to and from functions
 - more efficient than passing large data structures
 - allows passing of parameters by reference
 - allows passing of multiple arrays or hashes
- Object-oriented programming
 - a Perl object is a (referenced) special type of value

CSE2395/CSE3395

7

Making references

Method 1

- If you have a variable you want to refer to, prepend a backslash (\) to make a reference to that variable
 - like the & (address) operator in C
 - `$scalar = 5; $sref = \ $scalar;`
 - `@array = (1, 2, 3); $aref = \@array;`
 - `%hash = (a => 1, b => 2); $href = \%hash;`



Camel3 pages 245; Camel2 pages 245-246
perlref, perlreftut manpages

CSE2395/CSE3395

8

Making references

Method 2

- To make a reference to an anonymous array, put the array elements in square brackets ([]) `[]`
 - Perl creates a new array with the given values, and returns a reference to it
 - `$aref = [1, 2, 3];`
 - same as `@array = (1, 2, 3); $aref = \@array;` except doesn't create intermediate variable `@array`.
 - *this doesn't work:* `$aref = \ (1, 2, 3);`



Camel3 pages 245-246; Camel2 page 246
perlref, perlreftut manpages

CSE2395/CSE3395

9

Making references

Method 2, continued

- To make a reference to an anonymous hash, put the hash elements in braces ({ })
 - Perl creates a new hash with the given key-value pairs, and returns a reference to it
 - `$href = {a => 1, b => 2};`
 - same as `%hash = (a => 1, b => 2); $href = \%hash;` except doesn't create intermediate variable `%hash`.



Camel3 pages 246-247; Camel2 pages 246-247
perlref, perlreftut manpages

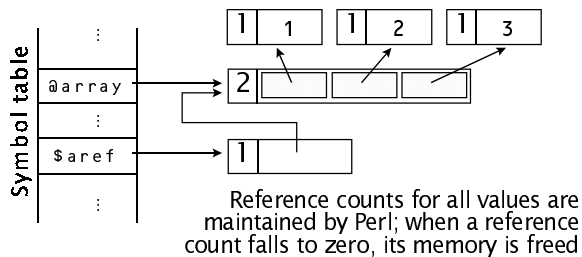
CSE2395/CSE3395

10

Internal implementation

Reference counts: reference to existing array

- `@array = (1, 2, 3); $aref = \@array;`



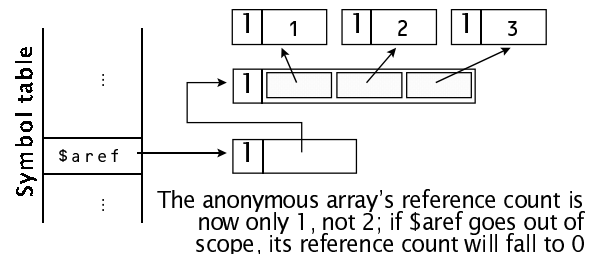
CSE2395/CSE3395

11

Internal implementation

Reference counts: reference to anonymous array

- `$aref = [1, 2, 3];`



CSE2395/CSE3395

12

Nested data structures

Two-dimensional arrays

- Elements of arrays (anonymous or named) are always scalars (including references)
 - arrays can contain references to arrays
 - `@array2d = ([11, 12, 13], [21, 22, 23], [31, 32, 33]);`
 - `$array2d_ref = [[11, 12, 13], [21, 22, 23], [31, 32, 33]];`
 - this is how Perl implements two-dimensional arrays
- Same applies to higher dimensions
- Same applies to hashes
 - can make arrays of (references to) hashes, etc.



Camel3 pages 245-246, 268-275; *Camel2* pages 257-264
`perl10l, perl1dsc, perlref, perlrefut` manpages

CSE2395/CSE3395

13

Example

Reading a two-dimensional matrix

```
@matrix = ();
# Read rows until end of file.
while (<<)
{
    # Split up the row by white space.
    @row = split /\s+/;
    # Make a reference to a list containing
    # the row.
    # This copies the values in @row into a new
    # anonymous array; doing $rowref = \@row
    # instead would result in a subtle bug.
    $rowref = [ @row ];
    # Add the row to the matrix.
    # Could avoid using intermediate variable with
    # push @matrix, [ @row ];
    push @matrix, $rowref;
}
```

CSE2395/CSE3395

14

Using references

Method 1

- To access a value referred to by a reference (dereferencing), prepend the appropriate type character (`$`, `@`, `%`)
 - like `*` (dereference) operator in C
 - `$$sref` (equals scalar `$`)
 - `@saref` (equals array `(1, 2, 3)`)
 - `%shref` (equals hash `(a => 1, b => 2)`)
- Using the wrong symbol causes a runtime error
 - `@$$sref` (error: `$sref` refers to scalar, not array)



Camel3 pages 251-252; *Camel2* pages 248-249
`perlref, perlrefut` manpages

CSE2395/CSE3395

15

Using references

Method 1, continued

- Can use braces for clarity and grouping
 - `#{ $sref }, @ { $saref }, % { $shref }`
- Can access elements of referenced array or hash using normal rules
 - `#{ $saref } [0]` (equals first value of array: 1)
 - `#{ $shref } { "a" }` (equals value in hash associated with key "a": 1)
 - `#{ $array2d [0] } [2]` (equals third element of array referred to by first element of `@array2d`: 13)
 - `#{ $ { $array2d_ref } [0] } [2]` (same, using `$array2d_ref`)



Camel3 pages 252-253; *Camel2* pages 249-250
`perlref, perlrefut` manpages

CSE2395/CSE3395

16

Using references

Method 2

- To access an element of a referenced array, place `->` between reference and `[index]`
 - `$aref->[0]` (equals first element of array: 1)
 - *this is wrong:* `$aref[0]`
 - `$array2d[0]->[2]` (equals third element of array referred to by first element of `@array2d`: 13)
- To access a value in a referenced hash, place `->` between reference and `{key}`
 - `$href->{"a"}` (equals value in hash associated with key "a": 1)



Camel3 pages 253-255; *Camel2* pages 250-251
`perlref, perlrefut` manpages

CSE2395/CSE3395

17

Using references

Method 2, continued

- This method can be repeated for nested data structures
 - `$array2d_ref->[0]->[2]` (equals element 2 of array referred to by element 0 of array referred to by `$array2d_ref`: 13)
- Between adjacent `[]` or `{}` *only*, can omit `->`
 - `$array2d_ref->[0][2]` (remaining `->` must stay)
 - `$array2d[0][2]` (same, using `@array2d`, looks like normal C two-dimensional array)



Camel3 pages 253-255; *Camel2* pages 250-251, 257-258
`perlref, perlrefut` manpages

CSE2395/CSE3395

18

Example

Applying a function to every element in a 2D array

```
# The 2D array we're working on.
@array2d = ( [11, 12, 13], [21, 22, 23],
            [31, 32, 33] );

# Count through every row of the array.
for ($rowc = 0; $rowc < @array2d; $rowc++)
{
    # Count through every column in this row.
    for ($colc = 0; $colc < @{$array2d[$rowc]};
        $colc++)
    {
        # Apply the function to this element and
        # replace the old value with the new.
        $array2d[$rowc][$colc] =
            function($array2d[$rowc][$colc])
    }
}
```

CSE2395/CSE3395

19

Example

Computing the difference between two arrays

```
# Call this function like:
# diff (\@first, \@second)
# It returns a list of all elements in @first
# that are not in @second.

sub diff
{
    my ($a, $b) = @_;
    my (@result, %seen);

    # Remember what elements are in @{$b}.
    foreach (@{$b}) { $seen{$_} = 1; }

    # Add each unseen element in @{$a} into @result.
    foreach (@{$a}) {
        push @result, $_ unless $seen{$_};
    }
    return @result;
}
```

CSE2395/CSE3395

20

Example

Properly reversing a hash

```
# Reversing a hash with the reverse function
# doesn't deal with duplicated values (see
# Topic 5). To fix this, we keep a reference
# to a *list* of matching keys in each value of
# the reversed hash.

%fruitcol = (banana => "yellow", cherry => "red",
            lemon => "yellow", lime => "green");

while (($fruit, $hue) = each %fruitcol)
{
    # Add this fruit to the array associated
    # with this hue.
    # This creates the anonymous array if necessary.
    push @{$colfruit{$hue}}, $fruit;
}

# %colfruit contains (red => ["cherry"], green =>
# ["lime"], yellow => ["banana", "lemon"])
```

CSE2395/CSE3395

21

Emulating C structures

Using hashes

- Perl doesn't have a special data structure for records (C structs)
- Typically use a (reference to anonymous) hash to represent a record
 - ▶ \$teacher = { name => "Debbie", height => 177, pets => ["Tiger", "Fudge"] };
- Can now make an array of records
 - ▶ @people = (\$teacher, \$baker, \$doctor);



Camel2 pages 277-279; Camel3 pages 264-265
perlref manpage

CSE2395/CSE3395

22

Dynamic typing

Determining the type of a referred value

- Can use `ref` function to determine type of object referred to by a reference
 - ▶ `ref $not_a_reference` (equals empty string)
 - ▶ `ref $sref` (equals string "SCALAR")
 - ▶ `ref $aref` (equals string "ARRAY")
 - ▶ `ref $href` (equals string "HASH")

CSE2395/CSE3395



Camel3 page 773; Camel2 pages 251-252
perlfunc, perlreftut manpages

23

Covered today

- References
 - ▶ Perl's answer to pointers
- Making references
 - ▶ \, [], and {}
- Using references
 - ▶ \${\$sref}, @{\$aref}, %{\$href}
 - ▶ \$aref->[], \$href->{}
- Nested data structures
 - ▶ multi-dimensional arrays
 - arrays of references to arrays
 - ▶ emulating C structs
 - anonymous hashes

CSE2395/CSE3395

24

Going further

More things related to today's topic

- Symbolic (“soft”) references
 - ▶ `$var = "Hello"; $ref="var"; print ${$ref};`
 - ▶ *Camel3* pages 263-264; *Camel2* pages 254-255
- Closures
 - ▶ anonymous functions which remember their scope, an alternative to object orientation
 - ▶ *Camel3* pages 259-263; *Camel2* pages 253-254; *Advanced Perl Programming* pages 56-64

25

Next time

To be covered in Topic 12

- Packages and modules
- Object-oriented Perl

CSE3395



Reading:

Programming Perl 3rd edition chapters 10-12, pages 288-292, 299-301, 308-346

Programming Perl 2nd edition chapter 5, pages 277-301, 315-325

`perlmod`, `perlobj`, `perltoot`, `perlbott` manpages

26

CSE2395/CSE3395 lecture notes copyright
© 2000-2001 Deborah Pickett.
Reproduction of this presentation for
nonprofit study use is permitted. All other
reproduction, including for other
educational courses, must be authorized in
writing by the author.

CSE2395/CSE3395

27