Day 8: Data Structures

Suggested reading:
http://perldoc.perl.org/perlreftut.html
or
perldoc perlreftut
Homework Review
So Far:

$  @  %
But What About … ?

Complex Data
‘Japan’ => (‘lang’ => ‘Japanese’, ‘pop’ => 127.56M )

Multidimensional Arrays
matrix[4][2] = 3.6354

Trees and Graphs
References
References

**Scalar** that refers to another scalar, list, hash, ...

```perl
my $a = 10;
my $b = \$a;
${$b} = 5;
print $a; # => 5
```
Making a Reference to a Variable

Prefix the variable with \ 

```perl
my $scalar_ref = \$region_string;
my $array_ref  = \@countries;
my $hash_ref   = \%country_code_map;
```

```
$array_ref

@countries

('Afghanistan', 'Albania', ..., 'Zimbabwe')
```
Making a Reference to Anonymous

Special syntax for unnamed arrays and hashes

```
my $cc_ref = [ 'ABW', 'AFG', ..., 'ZWE' ];
my $ch_ref = { 'ABW' => 'Aruba', 'AFG' => 'Afghanistan' };
```

$cc_ref

('ABW', 'AFG', 'AGO', 'AIA', ..., 'ZWE')
Using References

Use \texttt\{$reference$} in place of \texttt{name} of real thing

\begin{verbatim}

${$region_ref}  <=>  $region
@{$country_ref}  <=>  @country
${$country_ref}[1]  <=>  $country[1]
%{$code_ref}  <=>  %code
${$code_ref}"ABW"  <=>  $code{"ABW"}

print "Region: ${$region_ref}\n";
foreach my $country (@{$country_ref}) {
    print "${$code_ref}{$country}\n";
}
\end{verbatim}


Reference Shortcuts I

• No *need* to use — `{ref}` always works
• Use `ref->` to get one element of array or hash

```
$countries[2]
{$countries_ref}[2]
$countries_ref->[2]
$countries_ref[2] # BAD! (in Perl 5.8)
```

```
$country_codes{'ABW'}
{$country_code_ref}{'ABW'}
$country_code_ref->[{'ABW'}]
$country_code_ref->{'ABW'} # BAD! (in 5.8)
```
Reference Shortcuts II

Can omit \texttt{->} between indices

\begin{verbatim}
${${score_ref}{'Tim'}}[5]
$score_ref->{'Tim'}->[5]
$score_ref->{'Tim'}[5]
\end{verbatim}
Data Structures
No References

- array
- (linked) list
- stack
- queue
- associative array / hashtable / dictionary / map
- set
- others?
Array of Arrays
Matrix (equal-sized sub-arrays) or not

@countries

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ABW</td>
<td>Aruba</td>
<td>193</td>
<td>0.1</td>
</tr>
<tr>
<td>AFG</td>
<td>Afghanistan</td>
<td>652230</td>
<td>25.5</td>
</tr>
<tr>
<td>ZMB</td>
<td>Zambia</td>
<td>752612</td>
<td>13.0</td>
</tr>
<tr>
<td>ZWE</td>
<td>Zimbabwe</td>
<td>390757</td>
<td>13.0</td>
</tr>
</tbody>
</table>
Array of Arrays

Create as list of references to anonymous lists

```perl
my @countries = (
    ['ABW', 'Aruba', 193, 0.1 ],
    ['AFG', 'Afghanistan', 647500, 28.2 ],
    # ...
);

push @countries, [
    ['---', 'South Sudan', 619745, 8.2];
```
Array of Arrays

Access data via references

my @countries = (  
    [ 'ABW', 'Aruba', 193, 0.1 ],  
    [ 'AFG', 'Afghanistan', 647500, 28.2 ],  
    # ...
);

print "The size of Aruba is " .  
    $countries[0][2] .  
    " km^2.\n";

# or, $countries[0]->[2] if that helps.
Array of Arrays

Access data via references

```perl
my @countries = (
    [ 'ABW', 'Aruba', 193, 0.1 ],
    [ 'AFG', 'Afghanistan', 647500, 28.2 ],
    # ...;
);

my $population = 0;
foreach my $country (@countries) {
    $population += $country->[2];
}
print "$population million people!\n";
```
### General Structured Data

- No class or struct
- Nest lists and hashes as needed
- Use references to connect parts

<table>
<thead>
<tr>
<th>USA</th>
<th>CAN</th>
<th>JPN</th>
<th>IND</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Canada</td>
<td>area</td>
<td>9984670</td>
</tr>
<tr>
<td>lang</td>
<td></td>
<td>pop</td>
<td></td>
</tr>
<tr>
<td>en</td>
<td>fr</td>
<td>1960</td>
<td>17.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1961</td>
<td>18.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
General Structured Data

- Perl is crazy (but you knew that already)
- Auto-creates intermediate anonymous structures

```perl
$countries{'CAN'}{'name'} = 'Canada';
$countries{'IND'}{'lang'}[0] = 'hi';

$countries{'JPN'} = { 'name' => 'Japan' };  
$countries{'JPN'}{'lang'} = [];  
push @{$countries{'JPN'}{'lang'}}, 'ja';  
if (exists $countries{'JPN'}{'area'}) {
   $all_area += $countries{'JPN'}{'area'};
}
```
Trees and Graphs

- General solution may be complicated
- One approach: Hash the parent-child relations

```
$parent{ 'B' } = 'A';
$parent{ 'C' } = 'A';
$parent{ 'D' } = 'B';
$parent{ 'E' } = 'C';
```

or

```
$child{ 'A' } = [ 'B', 'C' ];
$child{ 'B' } = [ 'D' ];
$child{ 'C' } = [ 'E', 'F' ];
```
Subroutines and Data Structures
The Problem

Perl flattens arrays and hashes:

```perl
my @array_1 = (1, 2);
my @array_2 = (3, 4);
my %hash = ( 'foo' => 'bar' );

my_function(@array_1, @array_2, %hash);

sub my_function {
    print join(', ', @_) . "\n";
}

# 1, 2, 3, 4, foo, bar
```
The Solution

Use references!

my @array_1 = (1, 2);
my @array_2 = (3, 4);
my %hash = ('foo' => 'bar');

my_function(@array_1, @array_2, %hash);

sub my_function {
    my ($a1_ref, $a2_ref, $h_ref) = @_;
    my @array_1 = @{$a1_ref};
}
Almost Done!
Other Scripting Languages

- Nested data structures generally just work
- Typically: `object.member`
- Common data structures may have direct support
Homework

• Real country data, 1960–2002
• Read the assignment carefully! Lots of hints…
• Pick *just one* report
• Must use good data structure(s)
• If you are stuck on design, email me!