PDSC #0: General tips (abbreviated)

by Tom Christiansen
< tchrist@perl.com >

release 0.2
Sunday, 8 October 1995

The first thing you need to do is figure out how you want to access (such as via an assignment) just *one individual element* of your data structure just using lists and hashes. Use an **`@ARRAY`** if you're thinking of an array or linked list or stack or queue or deque, but use a **`%HASH`** if you're thinking of a record or a structure or a lookup table.

- **$coordinates[$row][$col] = "empty"**
  This is a simple two-dimensional array indexed by integers. Each first level numeric index itself produces a list (reference). See the List of Lists document.

- **$flight_time{"denver"}[3] = "12:34"**
  This is an (associative) array of lists. Each first level string index itself produces a list (reference). See the Hash of Lists document.

- **$student[$i]{"age"} = 15**
  This is an array of records that include named fields. Each first level numeric index itself produces a hash (reference). See the List of Hashes document.

- **$tv_shows{"the simpsons"}{"start time"} = "Monday 20:00"**
  This is an lookup table of records where you lookup the show by the name, and then you look up the record field by the field name. Each first level string index itself produces a hash (reference). See the Hash of Hashes document.

- **$tv{"the simpsons"}{members}[0]{name} = "homer"**
  This is an elaborate data structure involving a mix of records that contain fields that are sometimes themselves other arrays and records. See the More Elaborate Structures document.

- **$rec->{FH}->print(&{ $rec->{FUNC} } ($rec->{LIST}[0]))**
  This is a strange record that itself includes references to filehandles, functions, and other strings, lists, and hashes. We print to the filehandle referenced in **$rec**'s FH the result of calling the function in its FUNC field with an argument of the first element in the array which is its LIST field. See also the More Elaborate Structures document.

Note

Please see the expanded version of this document for a much more elaborate treatment of the following tips.

General Tips

Here are some further tips of general interest:

1. Always **use strict** and **-w**. The strict can be a pain, but it will save you from saying **$a[$i]** when you mean **$a->[$i]** and vice versa.

2. Things like **push()** require an @ sign, as in
   ```
   push @{ $a[3] }, @new_list
   ```
   You can't write
   ```
   push $a[3], @new_list
   ```

3. Things like **keys()** require a **%** sign, as in
   ```
   foreach $k (keys %{ $h{"key"} }) { ... }
   ```
   You can't write
   ```
   foreach $k (keys $h{"key"}) { ... }
   ```

4. Don't store pointers to existing data in a structure. Always create a new structure, eg. to build a 2D array indexed by line and by word number:

   ```
   while ( <> ) {
       @fields = split;
       push @a, [ @fields ];
   }
   ```

   This generally means never using the backslash to take a reference, but rather using the **[]** or **{}** constructors. This, for examples, is **wrong**!

   ```
   while ( <> ) {
       @fields = split;
       push @a, \@fields;
   }
   ```

   It's the same problem as this in C:

   ```
   char a[100], *p[10], *somefunc();
   for (i = 0; i < 10; i++) {
       strcpy(a, somefunc(i));
       p[i] = a;
   }
   ```

   An exception to this rule would be when you're writing a recursive data structure or are creating multiple key indices for the same set of records.

5. Never write **$$a[$i]** when you mean **${$a[$i]}** or **@$a[$i]** when you mean **@{$a[$i]}**. Those won't work at all.

6. Never write **$$a[$i]** even if you mean **$a->[$i]**. While it'll work to do that, it will needlessly confuse C programmers, who will think that subscripting binds tighter than the prefix dereference operator. This is right in C but wrong in perl where it's the other way around!

7. Remember that **$a[$i]** is the *i'th* elt of **@a**, but **$a->[$i]** is the *i'th* elt of the anon array pointed to by **$a**. Saying `use strict` will help here.

8. Never write
   ```
   @ { $a[$i] } = @list
   ```
   instead of
   ```
   $a[$i] = [ @list ]
   ```
   It'll work, but will confuse people.

9. Try to use pointer arrows and indirection bracketing whenever you feel the reader might be confused. Sometimes it'll clear things up in your mind as well. Here are the five kinds of prefix dereferencers with disambiguating braces:

   - **print ${ $a[$i] }**
     Dereference a SCALAR reference.
   - **pop @{ $a[$i] }**
     Dereference an ARRAY reference.
   - **@k = keys %{ $a[$i] }**
     Dereference a HASH reference.
   - **$retval = &{ $a[$i] }( )**
     Dereference a CODE reference.
   - ***{ $a[$i] } = \&func**
     Dereference a GLOB reference; this is deep magic best left to the wizards.

10. The new perl5db (ftp to perl.com in /pub/perl/ext/) will help print out complex data structures using the **x** and **X** commands.