or

```
        {
                local $" = "\n";
                print "@big_array";
        }
```

- Pass by reference

  Pass arrays and hashes by reference, not by value. For one thing, it's the only way to pass multiple lists or hashes (or both) in a single call/return. It also avoids creating a copy of all the contents. This requires some judgment, however, because any changes will be propagated back to the original data. If you really want to mangle (er, modify) a copy, you'll have to sacrifice the memory needed to make one.

- Tie large variables to disk.

  For "big" data stores (i.e. ones that exceed available memory) consider using one of the DB modules to store it on disk instead of in RAM. This will incur a penalty in access time, but that's probably better than causing your hard disk to thrash due to massive swapping.

19.1.18 Is it safe to return a reference to local or lexical data?

Yes. Perl's garbage collection system takes care of this so everything works out right.

```
sub makeone {
    my @a = ( 1 .. 10 );
    return \@a;
}

for ( 1 .. 10 ) {
    push @many, makeone();
}

print $many[4][5], "\n";

print "@many\n";
```

19.1.19 How can I free an array or hash so my program shrinks?

You usually can't. On most operating systems, memory allocated to a program can never be returned to the system. That's why long-running programs sometimes re-exec themselves. Some operating systems (notably, systems that use mmap(2) for allocating large chunks of memory) can reclaim memory that is no longer used, but on such systems, perl must be configured and compiled to use the OS's malloc, not perl's.

However, judicious use of my() on your variables will help make sure that they go out of scope so that Perl can free up that space for use in other parts of your program. A global variable, of course, never goes out of scope, so you can't get its space automatically reclaimed, although undef()ing and/or delete()ing it will achieve the same effect. In general, memory allocation and de-allocation isn't something you can or should be worrying about much in Perl, but even this capability (preallocation of data types) is in the works.