Sorting Techniques in Perl

Introduction

Sorting is a commonly needed operation in all kinds of programs. Luckily, for us perl programmers, perl provides a very simple yet extremely powerful mechanism to accomplish any sort you might think of. This article is about teaching the novice programmer how to sort lists of things, while showing to the more experienced folks certain techniques and ideas that could be new to them if they are migrating from a different language.

Moving to the meat of the matter straight away, we'll start from talking about comparison. Obviously, in order to put a list of things in order, you'll have to first define that order. Order is defined by how things compare to each other. If I give you two items from the list, can you tell me which one is bigger / better / nicer / sexier ... [insert you favorite adjective here] than the other? Or tell me that they are both of equal order? Well, that's just about it! If you give me a list of items and promise me that you can answer this question for any pair of them, I can make a sorted list of them. All I have to do is take all possible pairs and ask you "how do these two compare?" and arrange them accordingly to finally come up with a sorted list. Actually there are even smarter ways to do it, minimizing the amount of comparisons needed, but that is not an issue here, as we will see soon that perl performs that task for us, and we trust perl that it uses the least expensive method.

Now, the issue in question being comparison, I assume you must be familiar with all (or at least most) of perl's comparison operators. There's a list of them:

Numbers	Strings
<	lt
>	gt
<=	le
>=	gr
==	eq
<=>	cmp

Now the first five rows should be ok, they're just like math. But what are the <=> and cmp operators? Basically, the expression $a <=> $b (or $a cmp $b for strings) returns one of the values 1, 0, -1 if $a is, respectively, larger, equal or lower than $b. (see table below)

Relation of $a and $b	Value Returned by $a <=> $b
$a greater than $b	1
$a equal to $b	0
$a less than $b	-1

Does that ring a bell? Coming to think of it, the <=> and cmp operators actually provide the answer to the question we were investigating earlier when we talked about how to sort by using a comparative criterion. So, if we already have an operator that answers this question ("how do two items compare?"), all we need is a function that will take a list of items and perform the necessary comparisons to arrive at a sorted list.
And guess what? That's exactly what perl's sort operator does. So, if you have an unsorted list @not_sorted and want to create a sorted list @sorted, you just say:

```
@sorted = sort { $a <=> $b } @not_sorted   # numerical sort or
@sorted = sort { $a cmp $b } @not_sorted   # ASCII-betical sort
```

or better

```
 @sorted = sort { lc($a) cmp lc($b) } @not_sorted  # alphabetical
# sort
```

Looking at the sort function, we notice that it is exactly as we described it in words, earlier in this article. Perl needs just two things: the list of items to sort, and a subroutine that answers the question "how do these two compare?" for any pair of items in the list. Perl puts the two items it want to compare into the special variables $a and $b and your

function is responsible to give a return value that corresponds to the existing relationship of the two, as shown in the table shown earlier.

Here, in this simple example, we used perl's built-in comparison operators that work for numerical and alphabetical (not really... to be correct it is ASCII order) sorting. Of course, you can roll your own comparison function to create sorts for any kind of ordering you wish to have. Before you start coding your own functions, take a look to the following examples:

Get a list of hash keys sorted by value.

```
@sorted = sort { $hash{$a} cmp $hash{$b} } keys %hash;
```

Get a reverse sort of a list.

```
@sorted = sort { $b cmp $a } @list;
```

Which can also be done with

```
@sorted = reverse sort { $a cmp $b } @list;
```

Get an alphabetical sort of words, but make 'aardvark' always come last.

(Now, why you would want to do that is another question...)

```
@sorted = sort {
  if ($a eq 'aardvark') { return 1; }
  elsif ($b eq 'aardvark') { return -1; }
  else { return $a cmp $b; }
} @words;
```