NAME

sort - perl pragma to control sort() behaviour

SYNOPSIS

```
    use sort 'stable';          # guarantee stability
    use sort 'defaults';        # revert to default behavior
    no  sort 'stable';          # stability not important

    my $current;
    BEGIN {
        $current = sort::current();     # identify prevailing pragmata
    }
```

DESCRIPTION

With the "sort" pragma you can control the behaviour of the builtin "sort()" function.

A stable sort means that for records that compare equal, the original input ordering is preserved. Stability will matter only if elements that compare equal can be distinguished in some other way. That means that simple numerical and lexical sorts do not profit from stability, since equal elements are indistinguishable. However, with a comparison such as

```
   { substr($a, 0, 3) cmp substr($b, 0, 3) }
```

stability might matter because elements that compare equal on the first 3 characters may be distinguished based on subsequent characters.

Whether sorting is stable by default is an accident of implementation that can change (and has changed) between Perl versions. If stability is important, be sure to say so with a

```
  use sort 'stable';
```

The "no sort" pragma doesn't *forbid* what follows, it just leaves the choice open. Thus, after

```
  no sort 'stable';
```

sorting may happen to be stable anyway.

CAVEATS

As of Perl 5.10, this pragma is lexically scoped and takes effect at compile time. In earlier versions its effect was global and took effect at run-time; the documentation suggested using "eval()" to change the behaviour:

```
{ eval ’no sort "stable"’;      # stability not wanted
  print sort::current . "\n";
  @a = sort @b;
  eval ’use sort "defaults"’;   # clean up, for others
}
{ eval ’use sort qw(defaults stable)’;     # force stability
  print sort::current . "\n";
  @c = sort @d;
  eval ’use sort "defaults"’;   # clean up, for others
}
```

Such code no longer has the desired effect, for two reasons. Firstly, the use of "eval()" means that the sorting algorithm is not changed until runtime, by which time it’s too late to have any effect. Secondly, "sort::current" is also called at run-time, when in fact the compile-time value of "sort::current" is the one that matters.

So now this code would be written:

```
{ no sort "stable";      # stability not wanted
  my $current;
  BEGIN { $current = sort::current; }
  print "$current\n";
  @a = sort @b;
  # Pragmas go out of scope at the end of the block
}
{ use sort qw(defaults stable);     # force stability
  my $current;
  BEGIN { $current = sort::current; }
  print "$current\n";
  @c = sort @d;
}
```