91.3.20 Better pseudo-random number generator

In 5.005_0x and earlier, perl's rand() function used the C library rand(3) function. As of 5.005_52, Configure tests for drand48(), random(), and rand() (in that order) and picks the first one it finds.

These changes should result in better random numbers from rand().

91.3.21 Improved `qw//` operator

The `qw//` operator is now evaluated at compile time into a true list instead of being replaced with a run time call to `split()`. This removes the confusing misbehaviour of `qw//` in scalar context, which had inherited that behaviour from split().

Thus:

```
    $foo = ($bar) = qw(a b c); print "$foo|$bar\n";
```

now correctly prints "3|a", instead of "2|a".

91.3.22 Better worst-case behavior of hashes

Small changes in the hashing algorithm have been implemented in order to improve the distribution of lower order bits in the hashed value. This is expected to yield better performance on keys that are repeated sequences.

91.3.23 pack() format 'Z' supported

The new format type 'Z' is useful for packing and unpacking null-terminated strings. See pack in *perlfunc*.

91.3.24 pack() format modifier '!' supported

The new format type modifier '!' is useful for packing and unpacking native shorts, ints, and longs. See pack in *perlfunc*.

91.3.25 pack() and unpack() support counted strings

The template character '/' can be used to specify a counted string type to be packed or unpacked. See pack in *perlfunc*.

91.3.26 Comments in pack() templates

The '#' character in a template introduces a comment up to end of the line. This facilitates documentation of pack() templates.

91.3.27 Weak references

In previous versions of Perl, you couldn't cache objects so as to allow them to be deleted if the last reference from outside the cache is deleted. The reference in the cache would hold a reference count on the object and the objects would never be destroyed.

Another familiar problem is with circular references. When an object references itself, its reference count would never go down to zero, and it would not get destroyed until the program is about to exit.

Weak references solve this by allowing you to "weaken" any reference, that is, make it not count towards the reference count. When the last non-weak reference to an object is deleted, the object is destroyed and all the weak references to the object are automatically undef-ed.

To use this feature, you need the Devel::WeakRef package from CPAN, which contains additional documentation.

```
    NOTE: This is an experimental feature.  Details are subject to change.
```