```
sub GETC { print "Don't GETC, Get Perl"; return "a"; }
```

**DESTROY this**

As with the other types of ties, this method will be called when the tied handle is about to be destroyed. This is useful for debugging and possibly for cleaning up.

```
sub DESTROY {
    print "</shout>\n";
}
```

### 94.3.27 Malloc enhancements

If perl is compiled with the malloc included with the perl distribution (that is, if `perl -V:d_mymalloc` is 'define') then you can print memory statistics at runtime by running Perl thusly:

```
env PERL_DEBUG_MSTATS=2 perl your_script_here
```

The value of 2 means to print statistics after compilation and on exit; with a value of 1, the statistics are printed only on exit. (If you want the statistics at an arbitrary time, you'll need to install the optional module Devel::Peek.)

Three new compilation flags are recognized by malloc.c. (They have no effect if perl is compiled with system malloc().)

**-DPERL_EMERGENCY_SBRK**

If this macro is defined, running out of memory need not be a fatal error: a memory pool can allocated by assigning to the special variable `$^M`. See §??.

**-DPACK_MALLOC**

Perl memory allocation is by bucket with sizes close to powers of two. Because of these malloc overhead may be big, especially for data of size exactly a power of two. If `PACK_MALLOC` is defined, perl uses a slightly different algorithm for small allocations (up to 64 bytes long), which makes it possible to have overhead down to 1 byte for allocations which are powers of two (and appear quite often).

Expected memory savings (with 8-byte alignment in `alignbytes`) is about 20% for typical Perl usage. Expected slowdown due to additional malloc overhead is in fractions of a percent (hard to measure, because of the effect of saved memory on speed).

**-DTWO_POT_OPTIMIZE**

Similarly to `PACK_MALLOC`, this macro improves allocations of data with size close to a power of two; but this works for big allocations (starting with 16K by default). Such allocations are typical for big hashes and special-purpose scripts, especially image processing.

On recent systems, the fact that perl requires 2M from system for 1M allocation will not affect speed of execution, since the tail of such a chunk is not going to be touched (and thus will not require real memory). However, it may result in a premature out-of-memory error. So if you will be manipulating very large blocks with sizes close to powers of two, it would be wise to define this macro.

Expected saving of memory is 0-100% (100% in applications which require most memory in such 2**n chunks); expected slowdown is negligible.

### 94.3.28 Miscellaneous efficiency enhancements

Functions that have an empty prototype and that do nothing but return a fixed value are now inlined (e.g. `sub PI () { 3.14159 }`).

Each unique hash key is only allocated once, no matter how many hashes have an entry with that key. So even if you have 100 copies of the same hash, the hash keys never have to be reallocated.