NAME

carp - warn of errors (from perspective of caller)

cluck - warn of errors with stack backtrace (not exported by default)

croak - die of errors (from perspective of caller)

confess - die of errors with stack backtrace

shortmess - return the message that carp and croak produce

longmess - return the message that cluck and confess produce

SYNOPSIS

```
use Carp;
croak "We're outta here!";

use Carp qw(cluck);
cluck "This is how we got here!";

print FH Carp::shortmess("This will have caller's details added");
print FH Carp::longmess("This will have stack backtrace added");
```

DESCRIPTION

The Carp routines are useful in your own modules because they act like die() or warn(), but with a message which is more likely to be useful to a user of your module. In the case of cluck, confess, and longmess that context is a summary of every call in the call-stack. For a shorter message you can use carp, croak or shortmess which report the error as being from where your module was called. There is no guarantee that that is where the error was, but it is a good educated guess.

You can also alter the way the output and logic of `Carp` works, by changing some global variables in the `Carp` namespace. See the section on `GLOBAL VARIABLES` below.

Here is a more complete description of how shortmess works. What it does is search the call-stack for a function call stack where it hasn't been told that there shouldn't be an error. If every call is marked safe, it then gives up and gives a full stack backtrace instead. In other words it presumes that the first likely looking potential suspect is guilty. Its rules for telling whether a call shouldn't generate errors work as follows:

1. Any call from a package to itself is safe.
2. Packages claim that there won't be errors on calls to or from packages explicitly marked as safe by inclusion in @CARP_NOT, or (if that array is empty) @ISA. The ability to override what @ISA says is new in 5.8.
3. The trust in item 2 is transitive. If A trusts B, and B trusts C, then A trusts C. So if you do not override @ISA with @CARP_NOT, then this trust relationship is identical to, "inherits from".
4. Any call from an internal Perl module is safe. (Nothing keeps user modules from marking themselves as internal to Perl, but this practice is discouraged.)
5. Any call to Carp is safe. (This rule is what keeps it from reporting the error where you call carp/croak/shortmess.)

Forcing a Stack Trace

As a debugging aid, you can force Carp to treat a croak as a confess and a carp as a cluck across *all* modules. In other words, force a detailed stack trace to be given. This can be very helpful when trying to understand why, or from where, a warning or error is being generated.

This feature is enabled by 'importing' the non-existent symbol 'verbose'. You would typically enable it by saying

```
perl -MCarp=verbose script.pl
```

or by including the string `MCarp=verbose` in the PERL5OPT environment variable.

Alternately, you can set the global variable `$Carp::Verbose` to true. See the `GLOBAL VARIABLES` section below.

GLOBAL VARIABLES

$Carp::CarpLevel

This variable determines how many call frames are to be skipped when reporting where an error occurred on a call to one of `Carp`'s functions. For example:

```
$Carp::CarpLevel = 1;
sub bar     { .... or _error('Wrong input') }
sub _error  { Carp::carp(@_) }
```

This would make Carp report the error as coming from `bar`'s caller, rather than from `_error`'s caller, as it normally would.

Defaults to `0`.

$Carp::MaxEvalLen

This variable determines how many characters of a string-eval are to be shown in the output. Use a value of `0` to show all text.

Defaults to `0`.

$Carp::MaxArgLen

This variable determines how many characters of each argument to a function to print. Use a value of `0` to show the full length of the argument.

Defaults to `64`.

$Carp::MaxArgNums

This variable determines how many arguments to each function to show. Use a value of `0` to show all arguments to a function call.

Defaults to `8`.

$Carp::Verbose

This variable makes `Carp` use the `longmess` function at all times. This effectively means that all calls to `carp` become `cluck` and all calls to `croak` become `confess`.

Note, this is analogous to using `use Carp 'verbose'`.

Defaults to `0`.

BUGS

The Carp routines don't handle exception objects currently. If called with a first argument that is a reference, they simply call die() or warn(), as appropriate.