```
# Monitor:  an example of tie(); stores scalar, array, hash values as
# usual, but prints out a record of all reads and writes to each
# variable, useful in debugging to find out when a variable suddenly got
# a weird value

# ****************************************************************

# testmon.pl, to test the modules in Monitor

use Monitor::Scalar;
use Monitor::Array;
use Monitor::Hash;

$x;
@y;

tie $x,'Monitor::Scalar',\0,'x';
tie @y,'Monitor::Array',[0,0],'y';
tie %z,'Monitor::Hash',{},'z';

$x = 5;
print $x, "\n";  # prints 5
$y[0] = $x;
$y[1] = 12;
$y[2] = 13;
$z{'abc'} = 28;
@y = (1,2,8);  # causes execution error, due to lack of CLEAR()

# ****************************************************************

package Monitor::Scalar;  # file Scalar.pm in directory Monitor

# arguments:  package name, reference to initial value, name of tied
# variable (without $)
sub TIESCALAR {
   my ($pkg, $rval, $name) = @_;
   # $obj will be a reference to an anonymous array, consisting of the
   # name of the tied variable and the (dereferenced) value
   my $obj = [$name, $$rval];
   bless $obj, $pkg;
   return $obj;
}

# argument will just be the variable name
sub FETCH {
   my ($obj) = @_;
   my $val = $obj->[1];
   print STDERR 'Read    $', $obj->[0], " ... $val \n";
   return $val;
}

# arguments will be the variable name and the value to be written
sub STORE {
   # need parentheses when 'my' is applied to more than one variable
   my ($obj, $val) = @_;
   print STDERR 'Wrote   $', $obj->[0], " ... $val \n";
   $obj->[1] = $val;
}

1;

# ****************************************************************

package Monitor::Array;  # file Array.pm in directory Monitor

# arguments:  package name, reference to initial value of array, name of
# tied variable (without @)
sub TIEARRAY {
   my ($pkg, $rarray, $name) = @_;
   # $rarray is a reference to an array, so @$rarray is the array, so
   # [@$rarray] is a reference to an anonymous array having the same
   # elements as @$rarray (recall that, e.g., [2,5] is a reference to an
   # anonymous array consisting of 2 and 5)
   my $obj = [$name, [@$rarray]];
   bless $obj, $pkg;
   return $obj;
}

# argument will be the variable name and an array index
sub FETCH {
   my ($obj, $index) = @_;
   my $val = $obj->[1]->[$index];
   print STDERR 'Read    $', $obj->[0], "[$index] ... $val\n";
   return $val;
}

# argument will be the variable name, an array index and the value to be
# stored in that element of the array
sub STORE {
   my ($obj, $index, $val) = @_;
   print STDERR 'Wrote   $', $obj->[0], "[$index] ... $val\n";
   $obj->[1]->[$index] = $val;
}

# to allow full array operations, would also need CLEAR() (called by
# interpreter when array is "cleared," i.e. the array name is pointed to
# a different array than before), PUSH() (called when push() is called),
# etc.; see Perl documentation for full list

1;

# ****************************************************************

package Monitor::Hash;  # file Hash.pm in directory Monitor

# arguments:  package name, reference to initial value of hash, name of
# tied variable (without %)
sub TIEHASH {
   my ($pkg, $rhash, $name) = @_;
   # see comments in TIEARRAY() in Monitor::Array
   my $obj = [$name, {%$rhash}];
   return (bless $obj, $pkg);
}

# argument will be the variable name and a hash key
sub FETCH {
   my ($obj, $index) = @_;
   my $val = $obj->[1]->{$index};
   print STDERR 'Read    $', $obj->[0], "{$index} ... $val\n";
   return $val;
}

# argument will be the variable name, a hash key and its new value
sub STORE {
   my ($obj, $index, $val) = @_;
   print STDERR 'Wrote   $', $obj->[0], "{$index} ... $val\n";
   $obj->[1]->{$index} = $val;
   return $val;
}

1;
```