NAME

Algorithm::C3 - A module for merging hierarchies using the C3 algorithm

SYNOPSIS

```
use Algorithm::C3;

# merging a classic diamond
# inheritance graph like this:
#
# <A>
# / \
# <B> <C>
# \ /
# <D>

my @merged = Algorithm::C3::merge(
'D',
sub {
# extract the ISA array
# from the package
no strict 'refs';
@{$_[0] . '::ISA'};
}
);

print join ", " => @merged; # prints D, B, C, A
```

DESCRIPTION

This module implements the C3 algorithm. I have broken this out into it's own module because I found myself copying and pasting it way too often for various needs. Most of the uses I have for C3 revolve around class building and metamodels, but it could also be used for things like dependency resolution as well since it tends to do such a nice job of preserving local precedence orderings.

Below is a brief explanation of C3 taken from the Class::C3 module. For more detailed information, see the "SEE ALSO" section and the links there.

What is C3?

C3 is the name of an algorithm which aims to provide a sane method resolution order under multiple inheritance. It was first introduced in the language Dylan (see links in the "SEE ALSO" section), and then later adopted as the preferred MRO (Method Resolution Order) for the new-style classes in Python 2.3. Most recently it has been adopted as the 'canonical' MRO for Perl 6 classes, and the default MRO for Parrot objects as well.

How does C3 work.

C3 works by always preserving local precedence ordering. This essentially means that no class will appear before any of it's subclasses. Take the classic diamond inheritance pattern for instance:

```
<A>
/ \
<B> <C>
\ /
<D>
```

The standard Perl 5 MRO would be (D, B, A, C). The result being that **A** appears before **C**, even though **C** is the subclass of **A**. The C3 MRO algorithm however, produces the following MRO (D, B, C, A), which does not have this same issue.

This example is fairly trivial, for more complex examples and a deeper explanation, see the links

in the "SEE ALSO" section.

FUNCTION

merge ($root, $func_to_fetch_parent, $cache)

This takes a `$root` node, which can be anything really it is up to you. Then it takes a `$func_to_fetch_parent` which can be either a CODE reference (see SYNOPSIS above for an example), or a string containing a method name to be called on all the items being linearized. An example of how this might look is below:

```
{
package A;

sub supers {
no strict 'refs';
@{$_[0] . '::ISA'};
}

package C;
our @ISA = ('A');
package B;
our @ISA = ('A');
package D;
our @ISA = ('B', 'C');
}

print join ", " => Algorithm::C3::merge('D', 'supers');
```

The purpose of `$func_to_fetch_parent` is to provide a way for `merge` to extract the parents of `$root`. This is needed for C3 to be able to do it's work.

The `$cache` parameter is an entirely optional performance measure, and should not change behavior.

If supplied, it should be a hashref that merge can use as a private cache between runs to speed things up. Generally speaking, if you will be calling merge many times on related things, and the parent fetching function will return constant results given the same arguments during all of these calls, you can and should reuse the same shared cache hash for all of the calls. Example:

```
sub do_some_merging {
my %merge_cache;
my @foo_mro = Algorithm::C3::Merge('Foo', \&get_supers, \%merge_cache);
my @bar_mro = Algorithm::C3::Merge('Bar', \&get_supers, \%merge_cache);
my @baz_mro = Algorithm::C3::Merge('Baz', \&get_supers, \%merge_cache);
my @quux_mro = Algorithm::C3::Merge('Quux', \&get_supers, \%merge_cache);
# ...
}
```

CODE COVERAGE

I use **Devel::Cover** to test the code coverage of my tests, below is the **Devel::Cover** report on this module's test suite.

```
---------------------- ------ ------ ------ ------ ------ ------ ------
File stmt bran cond sub pod time total
---------------------- ------ ------ ------ ------ ------ ------ ------
Algorithm/C3.pm 100.0 100.0 100.0 100.0 100.0 100.0 100.0
---------------------- ------ ------ ------ ------ ------ ------ ------
Total 100.0 100.0 100.0 100.0 100.0 100.0 100.0
---------------------- ------ ------ ------ ------ ------ ------ ------
```

SEE ALSO

The original Dylan paper

http://www.webcom.com/haahr/dylan/linearization-oopsla96.html> -P < 4

The prototype Perl 6 Object Model uses C3

http://svn.openfoundry.org/pugs/perl5/Perl6-MetaModel/> -P < 4

Parrot now uses C3

http://aspn.activestate.com/ASPN/Mail/Message/perl6-internals/2746631> -P < 4
http://use.perl.org/~autrijus/journal/25768> -P < 4

Python 2.3 MRO related links

http://www.python.org/2.3/mro.html> -P < 4
http://www.python.org/2.2.2/descrintro.html#mro> -P < 4

C3 for TinyCLOS

http://www.call-with-current-continuation.org/eggs/c3.html> -P < 4

AUTHORS

Stevan Little, <stevan@iinteractive.com>

Brandon L. Black, <blblack@gmail.com>

COPYRIGHT AND LICENSE