```
# perl4 prints: $b 2 $a $b
# perl5 prints: 1 2 $a $b
```

- Regular Expression

  m//g now attaches its state to the searched string rather than the regular expression. (Once the scope of a block is left for the sub, the state of the searched string is lost)

```
$_ = "abababab";
while(m/ab/g){
    &doit("blah");
}
sub doit{local($_) = shift; print "Got $_ "}

# perl4 prints: Got blah Got blah Got blah Got blah
# perl5 prints: infinite loop blah...
```

- Regular Expression

  Currently, if you use the m//o qualifier on a regular expression within an anonymous sub, *all* closures generated from that anonymous sub will use the regular expression as it was compiled when it was used the very first time in any such closure. For instance, if you say

```
sub build_match {
    my($left,$right) = @_;
    return sub { $_[0] =~ /$left stuff $right/o; };
}
$good = build_match('foo','bar');
$bad = build_match('baz','blarch');
print $good->('foo stuff bar') ? "ok\n" : "not ok\n";
print $bad->('baz stuff blarch') ? "ok\n" : "not ok\n";
print $bad->('foo stuff bar') ? "not ok\n" : "ok\n";
```

  For most builds of Perl5, this will print: ok not ok not ok

  build_match() will always return a sub which matches the contents of $left and $right as they were the *first* time that build_match() was called, not as they are in the current call.

- Regular Expression

  If no parentheses are used in a match, Perl4 sets $+ to the whole match, just like $&. Perl5 does not.

```
"abcdef" =~ /b.*e/;
print "\$+ = $+\n";

# perl4 prints: bcde
# perl5 prints:
```

- Regular Expression

  substitution now returns the null string if it fails

```
$string = "test";
$value = ($string =~ s/foo//);
print $value, "\n";

# perl4 prints: 0
# perl5 prints:
```