POSIX syntax [= = is reserved for future] **extensions in regex; marked by <– HERE in m/%s/**

(F) Within regular expression character classes ([]) the syntax beginning with "[=" and ending with "=]" is reserved for future extensions. If you need to represent those character sequences inside a regular expression character class, just quote the square brackets with the backslash: "\[=" and "=\]". The <– HERE shows in the regular expression about where the problem was discovered. See *perlre*.

Possible attempt to put comments in qw() list

(W qw) qw() lists contain items separated by whitespace; as with literal strings, comment characters are not ignored, but are instead treated as literal data. (You may have used different delimiters than the parentheses shown here; braces are also frequently used.)

You probably wrote something like this:

```
@list = qw(
    a # a comment
    b # another comment
);
```

when you should have written this:

```
@list = qw(
    a
    b
);
```

If you really want comments, build your list the old-fashioned way, with quotes and commas:

```
@list = (
    'a',    # a comment
    'b',    # another comment
);
```

Possible attempt to separate words with commas

(W qw) qw() lists contain items separated by whitespace; therefore commas aren't needed to separate the items. (You may have used different delimiters than the parentheses shown here; braces are also frequently used.)

You probably wrote something like this:

```
qw! a, b, c !;
```

which puts literal commas into some of the list items. Write it without commas if you don't want them to appear in your data:

```
qw! a b c !;
```

Possible memory corruption: %s overflowed 3rd argument

(F) An ioctl() or fcntl() returned more than Perl was bargaining for. Perl guesses a reasonable buffer size, but puts a sentinel byte at the end of the buffer just in case. This sentinel byte got clobbered, and Perl assumes that memory is now corrupted. See ioctl in *perlfunc*.

Possible precedence problem on bitwise %c operator

(W precedence) Your program uses a bitwise logical operator in conjunction with a numeric comparison operator, like this :

```
if ($x & $y == 0) { ... }
```