```
# extract hours, minutes, seconds
($hours, $minutes, $second) = ($time =~ /(\d\d):(\d\d):(\d\d)/);
```

If the groupings in a regexp are nested, $1 gets the group with the leftmost opening parenthesis, $2 the next opening parenthesis, etc. For example, here is a complex regexp and the matching variables indicated below it:

```
/(ab(cd|ef)((gi)|j))/;
 1  2      34
```

so that if the regexp matched, e.g., $2 would contain 'cd' or 'ef'. For convenience, perl sets $+ to the string held by the highest numbered $1, $2, ... that got assigned (and, somewhat related, $^N to the value of the $1, $2, ... most-recently assigned; i.e. the $1, $2, ... associated with the rightmost closing parenthesis used in the match).

Closely associated with the matching variables $1, $2, ... are the **backreferences** \1, \2, ... Backreferences are simply matching variables that can be used *inside* a regexp. This is a really nice feature - what matches later in a regexp can depend on what matched earlier in the regexp. Suppose we wanted to look for doubled words in text, like 'the the'. The following regexp finds all 3-letter doubles with a space in between:

```
/(\w\w\w)\s\1/;
```

The grouping assigns a value to \1, so that the same 3 letter sequence is used for both parts. Here are some words with repeated parts:

```
% simple_grep '^(\w\w\w\w|\w\w\w|\w\w|\w)\1$' /usr/dict/words
beriberi
booboo
coco
mama
murmur
papa
```

The regexp has a single grouping which considers 4-letter combinations, then 3-letter combinations, etc. and uses \1 to look for a repeat. Although $1 and \1 represent the same thing, care should be taken to use matched variables $1, $2, ... only outside a regexp and backreferences \1, \2, ... only inside a regexp; not doing so may lead to surprising and/or undefined results.

In addition to what was matched, Perl 5.6.0 also provides the positions of what was matched with the @- and @+ arrays. $-[0] is the position of the start of the entire match and $+[0] is the position of the end. Similarly, $-[n] is the position of the start of the $n match and $+[n] is the position of the end. If $n is undefined, so are $-[n] and $+[n]. Then this code

```
$x = "Mmm...donut, thought Homer";
$x =~ /^(Mmm|Yech)\.\.\.(donut|peas)/; # matches
foreach $expr (1..$#-) {
    print "Match $expr: '${$expr}' at position ($-[$expr],$+[$expr])\n";
}
```

prints

```
Match 1: 'Mmm' at position (0,3)
Match 2: 'donut' at position (6,11)
```

Even if there are no groupings in a regexp, it is still possible to find out what exactly matched in a string. If you use them, perl will set $` to the part of the string before the match, will set $& to the part of the string that matched, and will set $' to the part of the string after the match. An example: