Регулярные выражения

Извлечение соответствий

Специальные переменные

Когда операция сопоставления находит в строке соответствие указанному регулярному выражению, она присваивает результаты своей работы нескольким специальным переменным:

- в переменную $` помещается часть строки до найденного соответствия;
- в переменную $& помещается часть строки, соответствующая образцу;
- в переменную $' помещается часть строки после найденного соответствия;
- в переменную $+ помещается последнее найденное совпадение для последнего шаблона в скобках.
- `@- Массив начальных индексов (смещений) совпадений в целевом тексте`
- `@+ Массив конечных индексов совпадений в целевом тексте`

  Первый элемент массива @- и @+ относится ко всему совпадению; иначе говоря, первый элемент массива @- (доступный как $-[0]) определяет смещение от начала целевого текста, с которого начинается совпадение.

  Первый элемент массива @+ (доступный как $+[0]) определяет смещение конца всего совпадения.

  Остальные элементы массивов содержат начальное и конечное смещение для каждой из сохраненных подгрупп. Так, пара $-[1] и $+[1] определяет смещения для подвыражения $1, пара $-[2] и $+[2] – для подвыражения $2 и т. д.

Если поиск окончился неудачей, то этим переменным новые значения не присваиваются.

Пример. Что сохранится в этих переменных после поиска такого соответствия:

```
$htm= "<A HREF='http://regexp.ru/'>Регулярные выражения</A>";
$htm =~ m|HREF=["']([\S+?)["']>|; # поиск URL сайта
```

При успешном совпадении с шаблоном в специальные переменные будут помещены такие значения:

```
$` = '<A '
$& = 'HREF='http://regexp.ru/'>'
$' = 'Регулярные выражения</A>'
$+ = 'http://regexp.ru/'
@- = (3,9)
@+ = (28,26)
```

Значениями этих переменных можно пользоваться при успешном сопоставлении с образцом, например:

```
print $& if $text =~ m/$pattern/; # выведет соответствие
```

Пример.

```
$str="The values x=1 yX=2 x=3";

$str=~/x/;
print $`."\n";
print $&."\n";
print $'."\n";
```

Захват значений по части шаблона

В регулярном выражении можно указать, что при успешном сопоставлении строки с шаблоном найденные соответствия нужно сохранить для дальнейшей обработки. С этой целью запоминаемые части шаблона нужно заключить в круглые скобки. Это также называется ***захватом значений***.

- Найденные совпадения для всех заключенных в скобки частей шаблона будут доступны через специальные переменные с именами $1, $2 и так далее.
- переменная $0 в список не входит – в ней хранится копия имени сценария, и эта переменная не имеет отношения к регулярным выражениям)

Составим ***регулярное выражение для поиска и сохранения в служебных переменных информации о сайте в том же тексте***:

```
$pattern = q|HREF=["'](\S+?)["']>([^<]+?)</A>|; # шаблон

$htm =~ m/$pattern/; # поиск соответствия в $htm

# в $1 = 'http://regexp.ru/'

# в $2 = 'Регулярные выражения'
```

- Сохраненные совпадения доступны и во время обработки регулярного выражения (т.е. ВНУТРИ регулярного выражения), но через переменные с именами \1, \2 и так далее. Эти переменные называются обратными ссылками (backreference) на найденные соответствия.

Пример. ***Найти два одинаковых слова, стоящих в тексте друг за другом через пробелы*** (возможно, по ошибке):

```
my $string = "Уже скоро скоро наступит весна!";

my $pattern = '(\S+)\s+\1'; # (\S+) сохранит значение 'скоро' в \1

$string =~ m/$pattern/; # соответствие: 'скоро скоро'
```

Также может найти, например, ‘the theme’, ‘and deep’.

Пример.

```
sub obr {
$_[0]=$_[0]*2.5;
}

print "Enter the length (i.e. 12.3 cm or 20.51 in)\n";

$str=<STDIN>;

chomp($str);
```

```
if($str=~/^([-+]?[0-9]+(\.[0-9]*)?) +(cm|in)$/) {
$ch=$1;
$ed=$3;
print $ch."  ".$ed."\n";

    if($ed eq "in")
    {
    $str=obr($str);

    $str.=" cm";
    }
    else
    {
       if($ed eq "cm")
       {
       $str=$str*2/5;
       $str.=" in";
       }
    }
 print $str."\n";
 }

 else
 {
 print $str." -- is not number\n";
 }
```

Отметим, что в используемом регулярном выражении есть пара круглых скобок, расположенных внутри другой пары, причем эта пара скобок предназначена для группировки указания квантификатору. Однако эта часть строки будет скопирована в переменную $2. Поэтому мы и использовали для определения единицы измерения $3. С целью избежать такого побочного эффекта используем группировку без сохранения (?:). То есть регулярное выражение приобретает следующий вид:

```
/^([-+]?[0-9]+(\.[0-9]*)?) +(cm|in)$/
```

(ЗДЕСЬ ОПЕЧАТКА. Во вложенной скобке -- (?:\.[0-9]*)))

Списочный контекст (без /g)

- Операция сопоставления, употребленная в списочном контексте, возвращает список найденных соответствий, для которых было предусмотрено сохранение значений.

Поэтому удобно сохранять найденные значения в массиве или в списке скалярных переменных.

Например, ***извлечем из текстовой строки последовательность цифр, похожую на время***:

```
my $text = 'Начало в 12:25:00.'; # строка с данными
my $pattern = '(\d\d):(\d\d):(\d\d)'; # образец для поиска
my @time = $text =~ m/$pattern/; # сохраним результат в массиве
my ($hh, $mm, $ss) = $text =~ m/$pattern/; # или в списке
```

каждый элемент списка соответствует паре круглых скобок. Три числа будут присвоены трем переменным, а также переменным $1, $2, $3.

- Если текст совпадения должен быть присвоен одной скалярной переменной, надо выполнить преобразование к списочному контексту, иначе вместо совпадения переменной будет присвоен логический признак успеха:

1)

```
my ($w) = $text =~ m/$pattern/; # в $w текст совпадения
```

2)

```
my $s = $text =~ m/$pattern/; # будет присвоен логический признак
```

Модификаторы поиска ======= /g

Оператор поиска сопоставления имеет формат:

```
$str =~ m/regexp/modifier;
```

Перечислим модификаторы для операции сопоставления:

/i - игнорировать регистр символов при поиске (case-Insensitive);

/g - искать в тексте все соответствия образцу (Global);

/s - рассматривать текст как одну строку (Single-line) -- режим совпадения метасимвола точки со всеми символами; обычно "точка" не совпадает с \n ;

/m - расширенный режим привязки к границам строк (Multi-line) с учетом \n ;

/o - один раз откомпилировать регулярное выражение (Once);

/x - использовать расширенный синтаксис регулярных выражений (eXtended) -- Режим свободного форматирования.

========== /g ==========

списковый контекст с модификатором /g

До сих пор операция сопоставления прекращала работу и возвращала результат, когда находилось первое соответствие строки указанному шаблону. Если для операции сопоставления указать модификатор /g (global), то она будет искать в строке все соответствия образцу, организуя неявный цикл обработки регулярного выражения.

- Эта полезная конструкция возвращает список всего текста, совпавшего с сохраняющими круглыми скобками (при отсутствии круглых скобок – текста, совпавшего со всем выражением), причем не только для одного совпадения, как в списковом контексте без модификатора /g, но и для всех совпадений в строке.

Например, так можно найти все числа в строке с помощью одного шаблона:

```
my @numbers = 'Не 12.5, а 25!' =~ /(\d+)/g; # глобальный поиск
print "@numbers";
# в @numbers будет (12, 5, 25)
```

Замена строк

Кроме поиска, регулярные выражения часто применяются для замены найденных совпадений на новые значения. Для этого существует операция замены (substitution) s/// (или оператор подстановки), которая

- пытается найти в строковой переменной соответствие образцу, а если находит, то заменяет найденную подстроку на указанное значение.

Операция замены выглядит так:

```
$variable =~ s/образец/замена/модификаторы;
# в переменной $variable отыскивается строка 'образец',
# и если найдена, то она заменяется на 'замена'
```

Все, что говорилось до этого про операцию сопоставления, применимо для левой части операции замены, в которой указывается образец поиска.

- Левая и правая части операции замены интерполируются, поэтому там могут использоваться escape-последовательности и переменные.

```
$pattern = 'шило'; # образец
$replacement = 'мыло'; # замена
$text =~ s/$pattern/$replacement/; # поменять 'шило' на 'мыло'
```

- В правой части операции замены могут использоваться обратные ссылки на найденные значения.

Пример. П***оменять местами два крайних слова в тройке слов, разделенных пробельными символами***:

```
$text = 'мать любит дочь';
$text =~ s/(\S+)\s+(\S+)\s+(\S+)/\3 \2 \1/;    # в $text будет 'дочь любит мать'
```

- Для операции замены s/// можно применять все модификаторы, упомянутые для операции сопоставления m//.

Например, модификатор /g указывает, что должны быть заменены все найденные в тексте соответствия.

```
$our_computers =~ s/Windows/Linux/g;
```

модификатор /e

У операции замены есть дополнительный модификатор /e (expression evaluation), при включении которого заменяющая часть вычисляется как выражение. При этом в заменяющей части можно использовать ссылки через специальные переменные $1, ... на захваченные при помощи круглых скобок соответствия. Это можно применять для более "интеллектуальной" замены найденных соответствий.

Пример, можно ***перевести температуру из шкалы Цельсия в шкалу Фаренгейта***:

```
$text = 'Бумага воспламеняется при 233C.';
$text =~ s/(\d+\.?\d*)C\b/int($1*1.8+32).'F'/e; #\1 нельзя
# в $text будет: 'Бумага воспламеняется при 451F.'
```

Пример

```
%h=('1'=>"one","2"=>"two","3"=>"three");
$i=0;

$str="The numbers  1 2 3  are replaced by strings";

print $str."\n";

$str=~s/([0-9]{1})/$h{$1}/g;

print $str;
```

Пример

```
s/([0-9]+)/sprintf("%x",$1)/ge
```

преобразует десятичные целые в 16-ричные.

Пример

Предположим, необходимо сократить десятичные дроби вида 12.30876 22100 до 12.308, но 1.230211155643 необходимо вывести в виде 1.23. Вот решение:

```
$num=~s/(\.\d\d[1-9]?)\d*/$1/;
```

Иногда необходимо, чтобы новая, модифицированная строка не портила старой. Вместо очевидного решения

```
$tm=$str;
$tmp=~s/.../.../;
```

можно объединить

```
($tmp=$str)=~s/.../.../;
```

Оператор замены можно применять и к массиву строк. Допустим так, как показано в примере, в котором все числовые данные окружаются HTML-тегом выделения текста полужирным шрифтом:

```
$str1="The first number is 12\n";

$str2="The second number is 23\n";

$str3="The third number is -12\n";

@m=($str1,$str2,$str3);

foreach(@m)
{
s/([-+]?[0-9]+)/<b>$1<\/b>/;
}
print "@m";
```

Заметим, что здесь по умолчанию используется переменная $_, которая является переменной цикла и строкой, к которой применяется оператор замены. С целью сохранения первоначального массива можно использовать одновременное присваивание и оператор замены, как и в скалярном случае:

```
for(@new=@m)
{
s/([-+]?[0-9]+)/<b>$1<\/b>/;
}
$"="\n";
print "@m";
print "\n";
print "@new";
```