Университет ИТМО, кафедра ВТ

Домашнее задание по регулярным выражениям в Perl

Работу выполнил

студент группы P3200

Рогов Я. С.

Преподаватель

Николаев В.В.

Санкт-Петербург, 2016

Примечание: отчёт для заданий 1-2 дан в виде [команда] – [ответ], [что ищет (и на что меняет)]

Задание 1: Пусть переменная var имеет значение abc123. Определите, что возвращают следующие выражения - true или false

1. $var =~ /./ **- true**, любой символ

2. $var =~ /[A-Z]*^/ **- true**, любое (>=0) количество заглавных букв перед началом строки

3. $var =~ /(\d)2(\1)/ **- false**, две одинаковых цифры, разделённых "2".

4. $var =~ /abc$/ - **false**, "abc" в конце строки

5. $var =~ /1234?/ - **true,** "123" или "1234"

Задание 2: Пусть $var имеет значение abc123abc. Какое значение примет $var после следующих подстановок

1. $var =~ s/abc/def/ - **def123,** заменяет первую подстроку "abc" на "def"

2. $var =~ s/(?<=\d)[a-z]+/X/g; - **abc123** ищет все подстроки вида "<=<цифра>< 1 или более букв из диапазона a-z>" - и заменяет последнюю их часть (состоящую из букв) на "X". (?expr) ищет совпадение, но не "захватывает" его.

3. $var =~ s/B/W/i; - **aWc123**, игнорируя регистр, ищет первое вхождение "b", и заменяет на "W"

4. $var =~ s/(.)\d.*\1/d/; - **abc123**, заменяет первую подстроку вида "<любой символ 1><цифра>< >=0 любых символов><любой символ 1>" на "d"

5. $var =~ s/(\d+)/$1*2/e; - **abc246**, ищет последовательность из цифр (>=1 символов), и выполняет выражение (модификатор **e**), следующее за регулярным выражением, в данном случае – получить подстроку первой группы, интерпретировать как число, умножить на два - и заменяет всю найденную подстроку на полученную строку.

Задание 3: Чему соответствуют следующие шаблоны

1. /a|bc*/ - подстрока "a" или подстрока "b" и любое количество "c" за ней

2. /[\d]{1,3}/ - подстрока, являющуюся последовательность от 1 до 3 цифр: "3", "42", "256"

3. /\bc[aou]t\b/ - "слову", равному "cat", "cot" или "cut". Слово – последовательность алфавитных,

4. нумерических символов, и нижнего подчёркивания.

5. /(xy+z)\.\1/ - подстроке вида "(x<1 или более символов y>z).(предыдущее выражение в скобках)", т.е. "xyyz.xyyz", но не "xyyz.xyyyyyyyyz"

6. /^$/ - пустая строка

Задание 4: Написать шаблоны, соответствующие примерам

1. Не менее пяти маленьких латинских букв, либо буква "a", если она стоит в начале строки

/[a-z]{5,}|^a/

2. Цифра 1 или слово "one" (в любом регистре):

/1|one/i

3. Число, возможно, дробное (с десятичной точкой):

/\d+(.\d+)?/

4. Любая буква, за которой следует гласная, повторяется еще раз (пример: "pop", "fifth", "daddy")

/([a-z])[euioa]\1/

5. Хотя бы один "+":

/\++/

Задание 5: Разобрать команды

Разберём наиболее встречаемые в этих командах ключи:
-e <строка>: передать программу на Perl как строку.
-l: автоматическое терминирование выводимых строк символом \n.
-n: оборачивает переданное выражение в цикл, повторяемый для каждой строки с потока стандартного ввода/файла
-p: аналогично -n, только печатает содержимое $_ в конце каждой итерации.

1.
```
perl -lne 'print if /((?<q>w)|a)(?(<q>)e|r)/'
                      ((     )|a)(?        )
                        ?<q>w      (<q>)e|r
```

В данном примере используется именование групп, в данном случае – под именем "q". В данном регулярном выражении (/((?<q>w)|a)(?(<q>)e|r)/):

1. Происходит поиск подстроки "w" . Если найдена, то помечаем не захватываемую группу с этим символом именем "q".

2. Иначе пытаемся найти символ "a". Результат пунктов 1 или 2 оборачивается в группу

3.Проверяется условие нахождения группы с именем "q". В случае нахождения, ищется подстрока "e", иначе – "r".

Т.о., все возможные варианты: "we" и "ar".

2.
```
perl -lpe '$p = qr/(\((?:[^()]++|(?-1))*+\))/; $_/=2 unless /x $p \+ y $p/x'
                    (\((?:             )*+\))
                          [^()]++|(?-1)
```

Разберём первую команду. В данном выражении строка, переданная как аргумент функции qr оборачивается как регулярное выражение и присваевается переменной $p. Разберём регулярной выражение:
1. Ищется подстрока "("
2. После неё ищется подстрока, включающая в себя любые символы, кроме скобок "(" и ")" и захватывает максимально длинную подходящую строку без возврата (backtracking).
    Иначе ищется подстрока, соответствующая прошлому группе, т.е. в данном случае – самой длинной группе в данном выражении
3. Найденное в п.1 оборачивается в незахватываемую группу. Происходит поиск таких подряд идущих выражений 0 или более раз без возврата.
4. Ищется подстрока ")"
Т.о. регулярное выражение в $p ищет выражение с возможными вложенными скобками. Например, "(a)", "(a(b(c)))", "(a(b+c)(d*(e-f)))"

Далее происходит деление входной строки, если НЕ выполняется условие, описываемое шаблоном: "x <регулярное выражение $p> + y <регулярное выражение $p>" и игнорированием пробельных символов и разрешением комментариев (модификтор **x**)

Т.о. происходит деление входной строки как числа, если она не соответствует выражению "x(expr1)+y(expr2)", где expr? - конструкция из как минимум одних скобок с возможными вложенными скобками.

3. `perl -lne 'continue if /[^a-z]/i;$v="aiueo";print if /^[^$v]?([$v][^$v])*[$v]?$/i'`

Развернём программу:

```
if ($_ =~ /[^a-z]/i){
      continue;
}
$v="aiueo";
if ($_ =~ /^[^$v]?([$v][^$v])*[$v]?$/i){
      print;
}
```

В первом условии регулярное выражение (`/[^a-z]/i`) ищет любые не алфавитные символы. Если находит, пропускает итерацию, т.е. пропускает строку.

Далее происходит объявление переменной v, vowels, т.е. гласные буквы.

После этого во втором условии происходит сравнение строки с шаблоном (`/^[^$v]?([$v][^$v])*[$v]?$/i`), который можно расшифровать как:

`"<0 или 1 согласный><0 или более последовательностей <гласный><согласный>><0 или 1 гласный>",`

например "regex", "adam", "adore", но не "knife", "oracle"

Т.о. из потока ввода фильтруются английские слова, соответствующие шаблону выше.

4. `perl -lne '@c=();for(split""){if(y/([{/)]}/){push@c,$_;next}if(/[])}]/){push@c,$_,last if($_ ne pop@c);next}}print"F" if@c'`

Развернём программу:

```
@c=();
for (split""){
      if (y/([{/)]}/){
            push @c, $_;
            next
      }
      if (/[])}]/){
            push @c, $_, last
            if ($_ ne pop @c);
            next
      }
}
print"F" if @c
```

Разберём её по строчкам:

1. Объявляем пустой массив
2. Разделяем входную строку на отдельные символы и происходит цикл для каждого из них, перед этим присваивая их значение переменной $_
3. Заменяет открывающую скобку на соответствующую ей закрывающую. Если замена произошла:
    3.1 Добавить в конец массива @c полученную подстроку, а именно – символ
    3.2 Перейти к следующей итерации
4. Проверяет, является ли символ закрывающей скобкой. Если да:

4.1 Вызов функции last – выход из цикла - при выполнении условия: если найденная закрывающая скобка НЕ соответствует последней найденной открывающей.

4.1.1 Иначе в массив добавляется эта закрывающая скобка.

4.1.2. И происходит переход к следующей итерации

5. Печатает "F", если массив не пустой.

Т.о. программа проверяет корректность выражения в плане количества пар соответствующих скобок: количество и положение закрывающих скобок соответствует количеству и положениям открытых скобок. Печатает "F", если выражение неверно. Т.о.: печатает F при "([])]", "{{{(}", но не печатает при "([{}])", "([[[[]]]])", "{()}"

5. `perl -lpe '/^[^][{}()]*(([[{[]((?:[^][{}()]*+|(?1)))*(??{$0=$2;$0=~y|([{|)]}|;"\\$0"}))/||($_=$.)'`

Разберём регулярное выражение:

```
/^[^][{}()]*(([[{[]((?:[^][{}()]*+|(?1)))*(??{$0=$2;$0=~y|([{|)]}|;"\\$0"}))/
/^[       ]*(       (                    )*(??{                            }))
   ^][{}()   (     ) (?:                )      $0=$2;$0=~y|([{|)]}|;"\\$0"
              [   ]     [       ]*+|(?1)
               ({[       ^][{}()
```

1. Начало строки
2. Любое количество любых символов, кроме скобок (открывающих и закрывающих)
3. Группа (1):
    1. Группа (2): любая открывающая скобка
    2. Любое количество групп (3):
        1. Незахватываемая группа:
            1. Любое количество любых символов, кроме скобок. Без возврата
            ИЛИ выражение группы (1)
    4. Группа (3): генерируемое регулярное выражение в фигурных скобках:
        1. Присваивание значения $2, найденной подстроки группы 2, переменной $0.
        2. Применение регулярного выражения `y|([{|)]}| <=> tr/([{/)]}/`, т.е. замена открывающей скобки на соответствующую ей закрывающую.
        3. Т.к. любая скобка является метасимволом, то её нужно экранировать: "\\$0"

Т.о. оно соответствует правильным (с ненарушенным порядком/количеством скобок) выражениям с вложенными скобками).

Последняя часть программы `"||($_=$.)"` отвечает за присваивание переменной $_ номера строки – переменной $. - если регулярное выражение не было найдено.

Т.о. программа выводит найденное выражение с корректными вложенными скобками, при его отсутствии выводит номер строки.

6. `perl '-es!!),-#(-.?{<>-8#=..#<-*}>;*7-86)!;y!#()-?{}!\x20/`-v;<!;s++$_+ee'`
Перепишем эту команду в развёрнутой форме:

```
perl -le '
$v = "),-#(-.?{<>-8#=..#<-*}>;*7-86)";
s//$v/;
tr|#()-?{}|\x20/`-v;<|;
s//$_/ee'
```

Разберём её построчно:

1. Т.к. передаваемые программе ключи, по сущности, строки, то ключ -e можно передавать и внутри строки с самой командой.
2. Присваивает переменной $v строку с, казалось бы, бессмысленной строкой.
3. Над переменной $_ выполняется регулярное выражение по замене пустой строки на $v. Т.к. $_ и есть пустая строка, то это идентично $_=$v

4. Основная "магия" программы: выше присвоенная "бессмысленная строка" переводится по некоторым правилам:

Что	#	(	)-?	{	}
На что заменяется	\x20	/	`-v	;	<
Комментарий	# заменяется на пробел - " "	-	Диапазон символов с ")" по "?" заменяется на соотв. символы из диапазона "`" - "v"	-	-

В данном случае та строка переводится в "`cd /dev;sudo tee sda<urandom`"

5. Операция замены с модификатором **ee** - интерпретировать правую часть выражения как строку, и выполнить на ней функцию **eval().** Т.к. по краям строки стоят символы "`" - выполнения команды и подстановки вывода – то команда "cd /dev;sudo tee sda<urandom " выполниться. Только, при наличии/получении соответствующих прав – если sudo была вызвана некоторое время назад, либо пользователь ввёл пароль.

Т.о. данная обфусцированная команда выполняет затирание блочного устройства sda псевдослучайными числами.

7. `perl -pe '11..exit'`

В данном примере используется оператор "**.." (**flip-flop) – по сути, обычный переключатель с двумя состояниями. Когда левая часть выражения возвращает true, он переключается в состояние true. Когда правая часть выражения возвращает true, он переключается в false.

Также, в данном примере используется неявное сравнение с $. - переменной, хранящей номер строки. Т.о. данное выражение можно переписать как "$.==11..exit". Тогда смысл этого выражения – читать строки до 11-й, переключить flip-flop и вызвать правую часть выражения. А т.к. эта правая часть – функция exit, то программа просто выходит. Тогда, по смыслу это выражение соответствует выражению "exit if 11==$.".

Т.о., программа читает строки до 11-й строки и прекращает работу.

8. `perl -pe 'print+(<>)[~9..-1]'`

Здесь используется унарный оператор "+", который не выполняет никаких действий над аргументом, а используется только для предотвращения интерпретации выражения в скобках как массива.

"<>" - оператор чтения из файла/потока. В данном контексте он интерпретируется в контексте массива, т.е. считывает весь файл поток в массив со строками как его элементами.

[n1..n2] – оператор "среза" массива, который берёт элементы массива с n1 ПО n2 и возвращает массив с этими элементами. Если n2 был больше размера даннного массива, то

остатки заполняются нулями. Если n1 и n2 отрицательные, то отсчёт идёт с конца. -1 – последний символ массива.

~ - унарный оператор побитового отрицания. При интерпретации полученного числа как отрицательного, можно расценивать равенство ~a = -a-1

Т.о. в одной итерации цикла (из-за опции **-p**) произойдёт чтение из стандартного потока ввода до EOF, получение последних 10 строк (-10..-1), их вывод, а также вывод содержимого $_ (опция же **-p**), что будет соответствовать первой строке из потока ввода.

9. `perl -pe '$\=$_.$\}{'`

Опция -p, по своей сути, разворачивает переданное выражение в подобную программу:

```
while (<>) {
            # переданное выражение
            print;}
```

Поэтому программа будет иметь следующий вид:

```
while (<>) {$\=$_.$\}
{print;}
```

Т.о. программа считывает строку со стандартного потока ввода, присваивает переменной $\ значение конкатенации этой строки и строки в $\. Когда строки закончились, происходит вывод значения переменной $_, но т.к. последняя строка точно была пустой, то произойдёт вывод пустой строки. Однако, т.к. мы переопределили значение переменной **$\, строки, добавляемой к каждому выводу print,** то у нас произойдёт вывод введённых нами строк в обратном порядке.

10. `perl -CSD -Mutf8 -0pe 's@^\* (.*?)[\r\n]+@<h3>$1</h3>@g' *`

Здесь использованы дополнительные ключи:

-C[число/спис.] – работа с указанными потоками будет выполняться в кодировке UTF-8. Использованный здесь список – **SD**, где **S = IOE**, т.е. стандартные потоки ввода, вывода и ошибок, и **D = io,** т.е. потоки ввода/вывода для открытых через **open()** файлов.
-M[-]модуль – подключает/отключает модуль, в зависимости от опции **"-"**. По сути, исполняет команды **"use модуль;"** или **"no модуль;"** соответственно.
Здесь используется модуль "utf8", т.е. поддержка utf-8 в исходном коде.
-0[octal/hexadecimal] – указывает разделитель строк для потока ввода в виде чисел в 8-й/16-й системах счислений соответственно. Если число не указано, как в нашем случае, разделителем считается NUL-символ с кодом 0.

Разберём регулярное выражение:

`'s@^\* (.*?)[\r\n]+@<h3>$1</h3>@g'`

Как разделитель в регулярном выражении здесь используется символ "@". Шаблон соответствует строке: "<начало строки>*<пробел><группа 1: любое количество любых символов><символ возврата каретки \r или переноса строки \n>". А т.к. разделителем строк у нас является символ NUL, то данное "начало строки" будет соответствовать началу потока ввода/файла. После чего, он заменяет эту строку на строку вида "<h3><найденное совпадение в группе 1></h3>".

Т.о. данная команда прочтёт все файлы рабочего каталога, и если в начале есть строка, описанная выше, он заменит её, и напечатает содержимое всего файла.

11.
```
perl -le '
      $LOVE=               AMOUR.
    true.cards.         ecstacy.crush
  .hon.promise.de     .votion.partners.
 tender.truelovers. treasure.affection.
devotion.care.woo.baby.ardor.romancing.
enthusiasm.fealty.fondness.turtledoves.
lovers.sentiment.worship.sweetling.pure
.attachment.flowers.roses.promise.poem;
 $LOVE=~ s/AMOUR/adore/g; @a=split(//,
  $LOVE); $o.= chr (ord($a[1])+6). chr
   (ord($a[3])+3). $a[16]. $a[5]. chr
    (32). $a[0]. $a[(26+2)]. $a[27].
      $a[5].$a[25]. $a[8].$a[3].chr
        (32).$a[29]. $a[8].$a[3].
          $a[62].chr(32).$a[62].
           $a[2].$a[38].$a[4].
               $a[3].".";
                 print
                  $o'
```
Развернём программу, опустив операторы конкатенации "." в первой команде(результат будет одинаковым):

```
$LOVE="AMOURtruecardsecstacycrushhonpromisedevotionpartnerstendertrueloverstreas
ureaffectiondevotioncarewoobabyardorromancingenthusiasmfealtyfondnessturtledoves
loverssentimentworshipsweetlingpureattachmentflowersrosespromisepoem";

$LOVE=~ s/AMOUR/adore/g;

@a=split(//, $LOVE);

$o .= chr(ord($a[1])+6).chr(ord($a[3])+3).$a[16].$a[5].chr(32).$a[0].$a[(26+2)].
$a[27].$a[5].$a[25].$a[8].$a[3].chr(32).$a[29].$a[8].$a[3].$a[62].chr(32).
$a[62].$a[2].$a[38].$a[4].$a[3].".";

print $o
```

Разберём каждую строчку:

1. Объявляется переменная $LOVE и ей задаётся значение – длинная строка

2. Происходит замена "AMOUR" на "adore" во всей строке в $LOVE, а именно – только в начале.

3. Происходит разделение строки на массив строк, используя шаблон // как разделитель. Т.к. он соответствует пустой строке, то в переменной @a будет лежать последовательный набор символов строки $LOVE. Т.е. "a", "d", "o" и т.д.

4. Происходит "сборка" строки из отдельных символов @a и кодов других символов, используя команды chr – получить символ из его кода, и ord – получить код по символу. Построим таблицу соответствия подвыражениям в данной строке:

Выражение	Используемый символ @a	Символ	Итоговое значение выражения
chr(ord($a[1])+6)	$a[1]	d	j
chr(ord($a[3])+3)	$a[3]	r	u
$a[16]	$a[16]	s	s
$a[5]	$a[5]	t	t
chr(32)	32	" "	" "
$a[0]	$a[0]	a	a
$a[(26+2)]	$a[(26+2)]	n	n
$a[27]	$a[27]	o	o
$a[5]	$a[5]	t	t
$a[25]	$a[25]	h	h
$a[8]	$a[8]	e	e
$a[3]	$a[3]	r	r
chr(32)	32	" "	" "
$a[29]	$a[29]	p	p
$a[8]	$a[8]	e	e
$a[3]	$a[3]	r	r
$a[62]	$a[62]	l	l
chr(32)	32	" "	" "
$a[62]	$a[62]	l	l
$a[2]	$a[2]	o	o
$a[38]	$a[38]	v	v
$a[4]	$a[4]	e	e
$a[3]	$a[3]	r	r

Т.о. на экран выведется **"just another perl lover"**

Вывод: в ходе выполнения данной лабораторной работы я изучил основной синтаксис языка Perl, а также базовые конструкции регулярных выражений в нём.