LING/C SC/PSYC 438/538

Lecture 8

Sandiway Fong
Adminstrivia

- Homework 4 not yet graded ...
Today's Topics

• Homework 4 review
• Perl regex
Homework 2 Review

• Sample data file:

Love for the the Bronx Bombers bubbled up in the absence of some local franchise.
On on on the table, we have some eggs.
I have home sharing turned on, on both my computer and my apple tv.
I wish that that question had an answer.
Because he had had too many beers already, he skipped the Friday office happy hour.
Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.

• First try.. *just try to detect a repeated word*

```perl
open($fh, $ARGV[0]) or die "$ARGV[0] not found!\n";
$line_num = 0;

while ($line = <$fh>) {
    chomp $line;
    chop $line if (substr($line,-1) eq ".");
    $line_num++;

    @words = split " ", $line;
    for ($i=0;$i<#words;$i++) {
        if (lc($words[$i]) eq lc($words[$i+1])) {
            print "Line $line_num: '$words[$i] at position $i+1, \" repeated\n";
        }
    }
}
```
Homework 2 Review

• Sample data file:

Love for the the Bronx Bombers bubbled up in the absence of some local franchise.
On on on the table, we have some eggs.
I have home sharing turned on, on both my computer and my apple tv.
I wish that that question had an answer.
Because he had had too many beers already, he skipped the Friday office happy hour.
Buffalo buffalo buffalo buffalo buffalo buffalo buffalo buffalo.

• Sample output:

    SBS2893:ling538-14 sandiway$ perl repeated.perl hw2.txt
    Line 1: 'the at position 3 repeated
    Line 2: 'On at position 1 repeated
    Line 2: 'on at position 2 repeated
    Line 4: 'that at position 3 repeated
    Line 5: 'had at position 3 repeated
    Line 6: 'Buffalo at position 1 repeated
    Line 6: 'buffalo at position 2 repeated
    Line 6: 'Buffalo at position 3 repeated
    Line 6: 'buffalo at position 4 repeated
    Line 6: 'buffalo at position 5 repeated
    Line 6: 'buffalo at position 6 repeated
    Line 6: 'Buffalo at position 7 repeated
Homework 2 Review

• Key: **think algorithmically...**
  – *think of a specific example first*

\[ w_1 \; w_2 \; w_3 \; w_4 \; w_5 \]

- Compare \( w_1 \) with \( w_2 \)
- Compare \( w_2 \) with \( w_3 \)
- Compare \( w_3 \) with \( w_4 \)
- Compare \( w_4 \) with \( w_5 \)
Homework 2 Review

- Generalize specific example, then code it up

   - Compare $w_1$ with $w_{1+1}$
   - Compare $w_2$ with $w_{2+1}$
   - Compare $w_{n-2}$ with $w_{n-2+1}$
   - Compare $w_{n-1}$ with $w_n$

Array @words

Array indices start from 0...

“for” loop implementation

Array indices end just before $#words...
Homework 2 Review

```perl
1 open($fh, $ARGV[0]) or die "$ARGV[0] not found!\n";
2
3
4 while ($line = <$fh>) {
5
6
7
8
9
10
11
12
13
14
15 }
```
Homework 2 Review

```perl
open($fh, $ARGV[0]) or die "$ARGV[0] not found!\n"

while ($line = <$fh>) {
  chomp $line;
  chop $line if (substr($line,-1) eq ".");
}
```
Homework 2 Review

```
1 open($fh, $ARGV[0]) or die "ARGV[0] not found!
2
3
4 while ($line = <$fh>) {
 5    chomp $line;
 6    chop $line if (substr($line,-1) eq ".")
7
8
9   @words = split " ", $line;
10
11
12
13
14
15 }
```
open($fh, $ARGV[0]) or die "ARGV[0] not found!\n";

while ($line = <$fh>) {
    chomp $line;
    chop $line if (substr($line, -1) eq ".");

    @words = split " ", $line;
    for ($i = 0; $i < #words; $i++) {
    }
}
open($fh, $ARGV[0]) or die "$ARGV[0] not found!\n";

while ($line = <$fh>) {
    chomp $line;
    chop $line if (substr($line,-1) eq ".");

    @words = split " ", $line;
    for ($i=0;$i<#words;$i++) {
        if (lc($words[$i]) eq lc($words[$i+1])) {
            
        }
    }
}
open($fh, $ARGV[0]) or die "$ARGV[0] not found!\n";

while ($line = <$fh>) {
  chomp $line;
  chop $line if (substr($line,-1) eq ".");

  @words = split " ", $line;
  for ($i=0;$i<$#words;$i++) {
    if (lc($words[$i]) eq lc($words[$i+1])) {
      print "Line $line_num: \"$words[$i] at position \"$i+1, \" repeated\n";
    }
  }
}
open($fh, $ARGV[0]) or die "ARGV[0] not found!\n";
=line_num = 0;

while ($line = <$fh>) {
  chomp $line;
  chop $line if (substr($line,-1) eq ".");
  $line_num++;

  @words = split " ", $line;
  for ($i=0;$i<#words;$i++) {
    if (lc($words[$i]) eq lc($words[$i+1])) {
      print "Line $line_num: "$words[$i] at position "$i+1, " repeated\n";
    }
  }
}
Homework 2 Review

• Sample data file:

Love for the the Bronx Bombers bubbled up in the absence of some local franchise. On on on the table, we have some eggs. I have home sharing turned on, on both my computer and my apple tv. I wish that that question had an answer. Because he had had too many beers already, he skipped the Friday office happy hour. Buffalo buffalo Buffalo buffalo buffalo buffalo buffalo buffalo Buffalo buffalo.

• Output:

SBS2893:ling538-14 sandiway$ perl repeated.perl hw2.txt
Line 1: 'the at position 3 repeated
Line 2: 'On at position 1 repeated
Line 2: 'on at position 2 repeated
Line 4: 'that at position 3 repeated
Line 5: 'had at position 3 repeated
Line 6: 'Buffalo at position 1 repeated
Line 6: 'buffalo at position 2 repeated
Line 6: 'Buffalo at position 3 repeated
Line 6: 'buffalo at position 4 repeated
Line 6: 'buffalo at position 5 repeated
Line 6: 'buffalo at position 6 repeated
Line 6: 'Buffalo at position 7 repeated
Homework 2 Review

• Second try.. *merging multiple occurrences*

```perl
open($fh, $ARGV[0]) or die "ARGV[0] not found!\n";
$line_num = 0;

while ($line = <$fh>) {
  chomp $line;
  chop $line if (substr($line,-1) eq ".");
  $line_num++;

  $repeated_word = ""; $repeats = 0;
  @words = split " ", $line;
  for ($i=0;$i<#words;$i++) {
    if (lc($words[$i]) ne lc($words[$i+1])) {
      if ($repeats > 0) {
        print "Line $line_num: '$repeated_word' at position $position occurs $repeats times\n";
        $repeated_word = ""; $repeats = 0;
      }
      else {
        if ($repeated_word) {
          $repeats++
        } else {
          $repeated_word = $words[$i]; $position = $i+1; $repeats = 2;
        }
      }
    } else {
      print "Line $line_num: '$repeated_word' at position $position occurs $repeats times\n"
    }
  }
}
```
Homework 2 Review

• Second try.. merging multiple occurrences
• Sample data file:

Love for the the Bronx Bombers bubbled up in the absence of some local franchise.
On on on the table, we have some eggs.
I have home sharing turned on, on both my computer and my apple tv.
I wish that that question had an answer.
Because he had had too many beers already, he skipped the Friday office happy hour.
Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.

• Output:

SBS2893:ling538-14 sandiway$ perl repeated2.perl hw2.txt
Line 1: 'the at position 3 occurs 2 times
Line 2: 'On at position 1 occurs 3 times
Line 4: 'that at position 3 occurs 2 times
Line 5: 'had at position 3 occurs 2 times
Line 6: 'Buffalo at position 1 occurs 8 times
Homework 2 Review

• Third try.. *implementing a simple table of exceptions*

```perl
open($fh, $ARGV[0]) or die "$ARGV[0] not found!\n"
$stable = ("had", 2, "that", 2);
$line_num = 0;
while ($line = <$fh>) {
    chomp $line;
    chop $line if (substr($line,-1) eq ".");
    $line_num++;
    $repeated_word = ""; $repeats = 0;
    @words = split " ", $line;
    for ($i=0;$i<@words;$i++) {
        if (lc($words[$i]) ne lc($words[$i+1])) {
            if ($repeats > $stable{repeated_word}) {
                print "Line $line_num: $repeated_word at position $position occurs $repeats times\n";
            }
            $repeated_word = "$words[$i]"; $position = $i+1; $repeats = 2;
        } else {
            if ($repeated_word) {
                $repeats++
            } else {
                $repeated_word = $words[$i]; $position = $i+1; $repeats = 2;
            }
        }
    }
    if ($repeats > 0) {
        print "Line $line_num: $repeated_word at position $position occurs $repeats times\n";
    }
}```
Homework 2 Review

- Third try.. *table of exceptions*
- Sample data file:

Love for the the Bronx Bombers bubbled up in the absence of some local franchise. On on on the table, we have some eggs. I have home sharing turned on, on both my computer and my apple tv. I wish that that question had an answer. Because he had had too many beers already, he skipped the Friday office happy hour. Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.

- Output:

```
SBS2893:ling538-14 sandiway$ perl repeated3.perl hw2.txt
Line 1: the at position 3 occurs 2 times
Line 2: On at position 1 occurs 3 times
Line 6: Buffalo at position 1 occurs 8 times
```

```
SBS2893:ling538-14 sandiway$ perl repeated2.perl hw2.txt
Line 1: 'the at position 3 occurs 2 times
Line 2: 'On at position 1 occurs 3 times
Line 4: 'that at position 3 occurs 2 times
Line 5: 'had at position 3 occurs 2 times
Line 6: 'Buffalo at position 1 occurs 8 times
```
Perl regex

• more powerful than simple wildcard matching, e.g. files
  – `rm *.jpg`, `rm PIC000?.JPG`

• Regular expression pattern matching:
  – regular expressions are patterns using operators:
    • `*` (zero or more occurrences),
    • `+` (one or more occurrences),
    • `?` (optional),
    • `|` (disjunction)

  – widely used in many areas
  – theoretically equivalent to Type-3 languages in the Chomsky hierarchy
    • less powerful than Context-free languages etc.
Perl regex

• Perl regular expression (re) matching:
  – `$a =~ /foo/`
  – `/…/` contains a regular expression
  – will evaluate to true/false depending on what’s contained in `$a`

• Perl regular expression (re) match and substitute:
  – `$a =~ s/foo/bar/`
  – `s/…match… /…substitute… /`
  – will modify `$a` by looking for a single occurrence of `match` and replacing that with `substitute`
  – `s/…match… /…substitute… /g`
  – `g = flag: global match and substitute`
Perl regex

- Typically useful with the standard code template for reading in a file line-by-line:

```perl
open($txtfile,$ARGV[0]) or die "$ARGV[0] not found!\n";
while ($line = <$txtfile>) {
  if ($line =~ /..regex../) {
    do stuff…
  }
}
```
Chapter 2: JM

<table>
<thead>
<tr>
<th>RE</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>/woodchucks/</td>
<td>“interesting links to woodchucks and lemurs”</td>
</tr>
<tr>
<td>/a/</td>
<td>“Mary Ann stopped by Mona’s”</td>
</tr>
<tr>
<td>/Claire_says,/</td>
<td>“Dagmar, my gift please,_ Claire says,_”</td>
</tr>
<tr>
<td>/DOROTHY/</td>
<td>“SURRENDER DOROTHY”</td>
</tr>
<tr>
<td>/!/</td>
<td>“You’ve left the burglar behind again!” said Nori</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>/[wW]oodchuck/</td>
<td>Woodchuck or woodchuck</td>
<td>“Woodchuck”</td>
</tr>
<tr>
<td>/[abc]/</td>
<td>‘a’, ‘b’, or ‘c’</td>
<td>“In uomini, in soldati”</td>
</tr>
<tr>
<td>/[1234567890]/</td>
<td>any digit</td>
<td>“plenty of 7 to 5”</td>
</tr>
</tbody>
</table>

Figure 2.1 The use of the brackets [ ] to specify a disjunction of characters.

character class: Perl lingo
Chapter 2: JM

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>/[A-Z]/</td>
<td>an upper case letter</td>
<td>“we should call it ‘Drenched Blossoms’”</td>
</tr>
<tr>
<td>/[a-z]/</td>
<td>a lower case letter</td>
<td>“my beans were impatient to be hoed!”</td>
</tr>
<tr>
<td>/[0-9]/</td>
<td>a single digit</td>
<td>“Chapter 1: Down the Rabbit Hole”</td>
</tr>
</tbody>
</table>

**Figure 2.2** The use of the brackets [ ] plus the dash – to specify a range.

<table>
<thead>
<tr>
<th>RE</th>
<th>Match (single characters)</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>^[A-Z]</td>
<td>not an upper case letter</td>
<td>“Oýfn pripitchik”</td>
</tr>
<tr>
<td>^Ss</td>
<td>neither ‘S’ nor ‘s’</td>
<td>“I have no exquisite reason for’t”</td>
</tr>
<tr>
<td>^.</td>
<td>not a period</td>
<td>“our resident Djinn”</td>
</tr>
<tr>
<td>[e`]</td>
<td>either ‘e’ or ‘’</td>
<td>“look up _ now”</td>
</tr>
<tr>
<td>a`b</td>
<td>the pattern ‘a ` b’</td>
<td>“look up a ` b now”</td>
</tr>
</tbody>
</table>

**Figure 2.3** Uses of the caret ^ for negation or just to mean `. We discuss below the need to escape the period by a backslash.
Chapter 2: JM

- Backslash lowercase letter for class
- Uppercase variant for all but class

<table>
<thead>
<tr>
<th>RE</th>
<th>Expansion</th>
<th>Match</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>\d</td>
<td>[0-9]</td>
<td>any digit</td>
<td>Party_of_5</td>
</tr>
<tr>
<td>\D</td>
<td>[^0-9]</td>
<td>any non-digit</td>
<td>Blue_moon</td>
</tr>
<tr>
<td>\w</td>
<td>[a-zA-Z0-9_]</td>
<td>any alphanumeric/underscore</td>
<td>Daiyu</td>
</tr>
<tr>
<td>\W</td>
<td>[^\w]</td>
<td>a non-alphanumeric</td>
<td>!!!!</td>
</tr>
<tr>
<td>\s</td>
<td>[\r\t\n\f]</td>
<td>whitespace (space, tab)</td>
<td></td>
</tr>
<tr>
<td>\S</td>
<td>[^\s]</td>
<td>Non-whitespace</td>
<td>in_Concord</td>
</tr>
</tbody>
</table>

*Figure 2.6* Aliases for common sets of characters.
Unicode and \w

- \w is [0–9A–Za–z_]  

Definition is expanded for Unicode:

```perl
use utf8;
use open qw(:std :utf8);

my $str = "school école École šola trường สกุล škole โรงเรียน";
@words = ($str =~ /(\w+)/g);
foreach $word (@words) { print "$word
" }
```

```bash
bash-3.2$ perl regex_utf.pl
school
école
École
šola
trường
สกุล
škole
โรงเรียน
```
# Chapter 2: JM

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>woodchucks?</td>
<td>woodchuck or woodchucks</td>
<td>“woodchuck”</td>
</tr>
<tr>
<td>colou?r</td>
<td>color or colour</td>
<td>“colour”</td>
</tr>
</tbody>
</table>

**Figure 2.4** The question mark (?) marks optionality of the previous expression.

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>/beg.n/</td>
<td>any character between beg and n</td>
<td>begin, beg’n, begun</td>
</tr>
</tbody>
</table>

**Figure 2.5** The use of the period (.) to specify any character.
Chapter 2: JM

Sheeptalk

the language of (certain) sheep, which consists of strings that look like the following:

   baa!
   baaa!
   baaaa!
   baaaaa!
   ...

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>zero or more occurrences of the previous char or expression</td>
</tr>
<tr>
<td>+</td>
<td>one or more occurrences of the previous char or expression</td>
</tr>
<tr>
<td>?</td>
<td>exactly zero or one occurrence of the previous char or expression</td>
</tr>
<tr>
<td>{n}</td>
<td>n occurrences of the previous char or expression</td>
</tr>
<tr>
<td>{n,m}</td>
<td>from n to m occurrences of the previous char or expression</td>
</tr>
<tr>
<td>{n,}</td>
<td>at least n occurrences of the previous char or expression</td>
</tr>
</tbody>
</table>

Figure 2.7  Regular expression operators for counting.
Chapter 2: JM

<table>
<thead>
<tr>
<th>RE</th>
<th>Match</th>
<th>Example Patterns Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>an asterisk “*”</td>
<td>“K<em>A</em>P<em>L</em>A*N”</td>
</tr>
<tr>
<td>.</td>
<td>a period “.”</td>
<td>“Dr. Livingston, I presume”</td>
</tr>
<tr>
<td>?</td>
<td>a question mark</td>
<td>“Why don’t they come and lend a hand?”</td>
</tr>
<tr>
<td>\n</td>
<td>a newline</td>
<td></td>
</tr>
<tr>
<td>\t</td>
<td>a tab</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.8 Some characters that need to be backslashed.
Chapter 2: JM

• Precedence of operators
  – Example: Column 1 Column 2 Column 3 ...
  – /Column [0-9]+ */
  – /(Column [0-9]+ *)*/
  – /house(cat(s | | |))/

• Perl:
  – in a regular expression the pattern matched by within the pair of parentheses is stored in designated variables $1 (and $2 and so on)

• Precedence Hierarchy:
Chapter 2: JM

http://perldoc.perl.org/perlretut.html

1. # extract hours, minutes, seconds
2. if ($time =~ /\d\d:\d\d:\d\d/) {
   # match hh:mm:ss format
   $hours = $1;
   $minutes = $2;
   $seconds = $3;
}

returns 1 (true) or "" (empty if false)

A shortcut: list context for matching

1. # extract hours, minutes, seconds
2. ($hours, $minutes, $second) = ($time =~ /\d\d:\d\d:\d\d/);
Chapter 2: JM

Backreferences

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

- \s/([0-9]+)/<\1>/

what does this do?

Backreferences give Perl regexps more expressive power than finite state automata (fsa)

The number operator can be used with other numbers. If you match two different sets of parenthesis, \2 means whatever matched the second set. For example,

/\textbf{the (.* \textbf{er}} \textbf{they (.* \textbf{), the \1er we \2/}

will match \textit{The faster they ran, the faster we ran} but not \textit{The faster they ran, the faster we ate}. These numbered memories are called registers (e.g., register 1, register 2,
Shortest vs. Greedy Matching

• default behavior
  – in Perl RE match:
    • *take the longest possible matching string*
  – aka *greedy matching*
    • *This behavior can be changed, see next slide*
Shortest vs. Greedy Matching


• Example:
  
  ```perl
  $_ = "The food is under the bar in the barn.";
  if ( /foo(.*?)bar/ ) {
    print "matched <$1>\n";
  }
  ```

• Output:
  – matched <d is under the >

• Notes:
  – ? immediately following a repetition operator like * (or +) makes the operator work in non-greedy mode
Shortest vs. Greedy Matching


• Example:
  
  ```perl
  $_ = "The foo d is under the bar in the bar n.";
  if ( /foo(.*?)bar/ ) {
      print "matched <$1>\n";
  }
  ```

• Output:
  - greedy: matched <d is under the bar in the >
  - shortest: matched <d is under the >
Shortest vs. Greedy Matching

• RE search is supposed to be fast
  – but searching is not necessarily proportional to the length of the input being searched
  – in fact, Perl RE matching can take exponential time (in length)
  – non-deterministic
    • *may need to backtrack (revisit) if it matches incorrectly part of the way through*
Global Matching: scalar context

`g` flag in the condition of a while-loop

In scalar context, successive matches against a string will have `//g` jump from match to match, keeping track of position in the string as it goes along. You can get or set the position with the `pos()` function. For example,

```perl
1. $x = "cat dog house"; # 3 words
2. while ($x =~ /(\w+)/g) {
3.  print "Word is $1, ends at position ", pos $x, "\n";
4. }
```

prints

1. Word is cat, ends at position 3
2. Word is dog, ends at position 7
3. Word is house, ends at position 13
Global Matching: list context

In list context, `//g` returns a list of matched groupings, or if there are no groupings, a list of matches to the whole regex. So

```
1. @words = ($x =~ /\w+/g);  # matches,
2.                             # $words[0] = 'cat'
3.                             # $words[1] = 'dog'
4.                             # $words[2] = 'house'
```

```
bash-3.2$ perl test.perl
cat dog house car
bash-3.2$
```

```
$s = "cats dogs houses cars";
@words = $s =~ /(\w+)s/g;
print "@words\n";
```
Split

• \(@array = \text{split } /re/, \ string\)
  – splits \(\text{string}\) into a list of substrings split by \(re\). Each substring is stored as an element of \(@array\).
• Examples (from perlrequick tutorial):

```
1. \(\text{x = "Calvin and Hobbes";}
2. \&word = \text{split } /\s+/, \x\);  \# \$word[0] = 'Calvin'
    \# \$word[1] = 'and'
    \# \$word[2] = 'Hobbes'
```
Split

If the empty regex `//` is used, the string is split into individual characters.

If the regex has groupings, then the list produced contains the matched substrings from the groupings as well.

```plaintext
1. $x = "/usr/bin";
2. @parts = split m!/(!, $x;  # $parts[0] = ''
# $parts[1] = '/'
# $parts[2] = 'usr'
# $parts[3] = '/'
# $parts[4] = 'bin'
```

Since the first character of $x matched the regex, `split` prepended an empty initial element to the list.
Matched Positions

Position information

In addition to what was matched, Perl (since 5.6.0) also provides the positions of what was matched as contents of 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

```
1.   $x = "Mmm...donut, thought Homer";
2.   $x =~ /(^Mmm|Yech)/..\.(donut|peas)/; # matches
3.   foreach $expr (1..$#) {
5.     print "Match $expr: "utivo expression ends: ($-[$expr],$+[expression])
}
```

prints

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

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:

1. $x = "the cat caught the mouse";
2. $x =~ /cat/;  # $^ = 'the ', $& = 'cat', $' = ' caught the mouse'
3. $x =~ /the/;  # $^ = '', $& = 'the', $' = ' cat caught the mouse'

In the second match, $^ equals '' because the regexp matched at the first character position in the string and stopped; it never saw the second 'the'. It is important to note that using $^ and $' slows down regexp matching quite a bit, while $& slows it down to a lesser extent, because if they are used in one regexp in a program, they are generated for all regexps in the program. So if raw performance is a goal of your application, they should be avoided. If you need to extract the corresponding substrings, use $[..] and $+[..] instead:

1. $^ is the same as substr( $x, 0, $-[0] )
2. $& is the same as substr( $x, $-[0], $+[0]-$-[0] )
3. $' is the same as substr( $x, $+[0] )