Reguljära uttryck - minnen

/^<w n=[0-9]+>.*<ana><ps>[A-Z]{2}<.*b>.*<\w>$/

```
1: <w n=6>i<ana><ps>PP<b>i</w>
2: <w n=7>grönskan<ana><ps>NN<m>UTR SIN
DEF NOM <b>grönska</w>
```

/^<w n=[0-9]+>**(.*)**<ana><ps>**([A-Z]{2})**<**(.*)**b>**(.*)**<\w>$/

$1 $2 $3 $4

	1	2
$1	i	grönskan
$2	PP	NN
$3	’’	m>UTR SIN DEF NOM <
$4	i	grönska

Variablerna $1 ($2, etc) är skrivskyddade. Du kan inte ändra deras värde och de skrivs över vid nästa matchning. Använd dem för att ”hämta” de matchade strängarna.

```
$lopord = $1;
$pos = $2;
$kanske_morf = $3;
$lemma = $4;
```


Perl är girigt!

Hur mycket ska matchas?

Kvantifikatorerna (’*’, ’+’) matchar alltid så mycket som möjligt.

/<(.*)>/ ger att $1 blir:

1: `w n=6>i<ana><ps>PP<b>i</w`

2: `w n=7>grönskan<ana><ps>NN<m>UTR SIN DEF NOM <b>grönska</w`

Men om vi bara vill matcha det som finns mellan ’<’ och ’>’?

/<(.*?)>/ ’*?’ minimal matchning. Ger att $1 blir:

1: `w n=6`

2: `w n=7`

split med RU:

Första argumentet i **split** kan vara ett reguljärt uttryck:
@rad = split(/<.*?>/,$rad);
@rad blir:
1: (”i”,””,”PP”,”i”)
2: (”grönskan”,””,”NN”,”UTR SIN DEF NOM”,”grönska”)

Om flera på varandra följande avgränsare ska tolkas som en avgränsare:
@rad = split(/(<.*?>)+/,$rad);
@rad blir:
1: (”i”,”PP”,”i”)
2: (”grönskan”,”NN”,”UTR SIN DEF NOM”,”grönska”)

s/// använder sig också av reguljära uttryck!

s/<.*?>//;
1: ’iPPi’
2: ’grönskanNNUTR SIN DEF NOMgrönska’

s/<.*?>/:/;
1: ’:i::PP:i:’
2: ’:grönskan::NN:UTR SIN DEF NOM:grönska:’

s/(<.*?>)+/:/;
1: ’:i:PP:i:’
2: ’:grönskan:NN:UTR SIN DEF NOM:grönska:’

Man kan ha variabler i RU

```
open(IN,Suc);
print ”Ange sökord!”;
$sok_ord = <STDIN>;
chomp($sok_ord);
$i = 0;
while (<IN>) {
        if (/\b$sok_ord\b/) {
                print $_;
                $i++;
        }
}
```

```
if ($i == 0) {
        print ”Tyvärr $sok_ord fanns
           inte i Suc!\n”;
        }
elsif ($ == 1) {
        print ”$sok_ord fanns en
             gång i Suc!”;
        }
else {
        print ”$sok_ord fanns $i
           gånger i Suc”;
}
```

Att veta om en fil gick att öppna:

```
open(IN,”SUC”) || die ”Filen gick inte att öppna!”;

# Om filen ”SUC” inte går att öppna så kommer
# Filen gick inte att öppna! Att skrivas ut på
# skärmen och programmet avbryts.
```


Program i flera filer (use):

```
# Vi har sett hur man kan organisera sina program
# i subrutiner. Man kan också del upp det i flera filer.

use(’subrutiner.pl’); # Säger till Perl att programmet
                      # också består av filen ’subrutiner.pl’

# Eller rättare sagt, att vi vill använda oss av kod
# som finns i filen ’subrutiner.pl’. Nu är det bara att
# anropa subrutinerna(!) som ligger i den filen.
```


Hasher på disk (tie/untie):

```
use(SDBM_File); # Perl ’vet’ var denna fil finns!
# Knyt hashen till en fil på disk
tie(%frekvhash, 'SDBM_File', 'diskhash');
# Vi lägger till/ ändrar i hashen
$frekvhash{’Kalle’} = ’G’;
# och stänger den
untie(% frekvhash);
# nu kommer allt vi la i hashen finnas kvar
# även nästa gång vi kör programmet!
```


Fortsätta på egen hand?

- Operatorers rangordning och ’associativity’
- Övriga styrsatser (kap 9)
  - Next, last och redo
- Logisk ’och’ och ’eller’ (kap 9)
- Bemästra reguljära uttryck (se länkar)
- Mer om filhandtag (kap 10)
- Formattering av utdata (kap 11)

Fortsätta på egen hand?

- Mer avancerad fiilhantering (kap 12 & 13)
- Mer om die och felmeddelanden ($!)
- Att struktuera sina program i filer (use)
- Läs vidare i perlfunc (mansidan för Perls inbyggda funktioner)

och för er egen skull:
UNDVIK SPAGETTIKOD!!!

LYCKA TILL!

Martin Hassel