Variablers räckvidd

I Perl är variablerna *globala* om man inte anger annat. Dvs, om du har variabeln ’**$a**’ så kommer alla operationer där variabeln **$a** tilldelas/interpoleras att påverka variabelns värde oberoende var i koden detta anges. Om du använder temporära värden i en subrutin så är det vettigt att göra variabeln *lokal*. På så vis, om du råkar ha en annan variabel med samma namn någon annanstans i koden, så hålls de isär.

OBS!

```
print ”Ge mig ett tal: ”;
$a = <STDIN>;
print ”Ge mig ett tal till: ”;
$b = <STDIN>;
chomp($a,$b);
$sum = addera_tva($a,$b);
print ”Summan av $a och $b är $sum.\n”;
$sum2 = addera_tva($a,$sum);
print ”Summan av $sum och $a är $sum2.\n”;
```

A

```
sub addera_tva {
        $sum = $_[0] + $_[1];
        return $sum; }
```

B

```
sub addera_tva {
        my($sum);
        $sum = $_[0] + $_[1];
        return $sum; }
```


Körning

Med A:
> Ge mig ett tal: *5*
> Ge mig ett tal till: *3*
> Summan av 5 och 3 är 8.
> Summan av 13 och 5 är 13.

Med B:
> Ge mig ett tal: *5*
> Ge mig ett tal till: *3*
> Summan av 5 och 3 är 8.
> Summan av 8 och 5 är 13.

my vs *local*

Med ’**my**’ gör vi så att variabeln bara påverkas i den egna subrutinen. Anropas en ny subrutin kommer den att se variabelns globala värde. Med ’**local**’ påverkas variabelns värde både i den egna subrutinen och i (av subrutinen) anropade subrutiner.

local

```
$value = ”original”;
tellme();
spoof();
tellme();

sub spoof {
   local($value) = ”temporary”;
   print ”Now it is $value.\n”;
   tellme();
}

sub tellme {
   print ”Value is $value.\n”;
}
```

my

```
$value = ”original”;
tellme();
spoof();
tellme();

sub spoof {
   my($value) =”temporary”;
   print ”Now it is $value.\n”;
   tellme();
}

sub tellme {
   print ”Value is $value.\n”;
}
```


Körning

local

```
Value is original.
Now it is temporary.
Value is temporary.
Value is original.
```

my

```
Value is original.
Now it is temporary.
Value is original.
Value is original.
```

Ännu en användbar Perl-lista

När man anropar ett program stoppas allt man skriver efter programnamnets namn (på kommandoraden) i listan @ARGV (argumentvektorn).

> mitt_prog infil utfil

```
open(IN,$ARGV[0]);
open(UT,">$ARGV[1]");
```


Reguljära uttryck

- är mönster
- en sträng som kan passas in i mönstret sägs matcha mönstret
- reguljära uttryck skrivs inom snedstreck i Perl, dessa kallas matchningsoperatorer
- matchningen kan lyckas (sant) eller misslyckas (falskt)

- matchningen sker mot strängen i '$_'
- vill man matcha mot någon annan sträng ska detta anges med '=~' (interpoleras)

T.ex.

$a =~ /reguljärt uttryck/;

Vi har redan använt reguljära uttryck:

- if (/<w/) { print "$_\n"; }

```
# Sant om strängen innehåller '<' direkt följt av 'w'
```

- s/<w//;

```
# När ett '<' följt av ett 'w' matchas byts de ut mot
# den tomma strängen, dvs 'ingenting'
```

-tr/a-zääö/A-ZÄÄÖ/;

```
# När ett tecken i första listan matchas byts det ut mot
# tecknet i mosvarande position i andra listan.
# Skulle kunna skrivas:

s/a/A/; s/b/B/; …, s/ö/Ö/;
```


Vi har matchat mot andra strängar än '$_':
$ord =~ tr/A-ZÄÄÖ/a-zääö/;

Olika sätt att skriva samma sak:

```
while (<IN>) {
        chomp;
        $rad = $_;
        if ($rad =~ /<w/) {
                ....
        }
}
```

```
while (<IN>) {
        chomp;
        if ( /<w/) {
                ....
        }
}
```


Enkla mönster:

- alla tecken matchar sig själva

(förutom specialtecknen som måste föregås av ett '\')

Ru	matchar	Ru	matchar
/a/	'a'	/\^/	'^'
/ab/	'ab'	/\?/	'?'
/1/	'1'	/\+/	'+'
/\\/	'\'	/\./	'.'
/\//	'/'	/\-/	'-' (bindestreck)
/*/	'*'	/ /	' ' (mellanslag)

Mängder

Ibland behöver man matcha ett av ett antal möjliga tecken. T.ex. en vokal.

RU	matchar
/[aeiouyåääö]/	’a’ el.’e’ el. ’i’....(bara en vokal)
/[a-zåääö]/	en bokstav
/[0-9]/	en siffra
/./	ett tecken vilket som helst

Inom ’[]’ behandlas varje tecken för sig.

Ibland vill man matcha vad som helst UTOM vissa kända tecken

RU	matchar
/[^a]/	allt utom strängen ’a’
/[^a-zåääö]/	allt utom en bokstav
/[^0-9]/	allt utom en siffra

Perls fördefinerade mängder:

Perl	RU	matchar
/\w/	/[a-zA-Z0-9_]/	en bokstav, siffra eller ’_’
/\d/	/[0-9]/	en siffra
/\s/	/[\r\t\n\f]/	ett blanktecken
/\W/	/[^ a-zA-Z0-9_]/	allt utom en bokstav, en siffra eller ’_’
/\D/	/[^ 0-9 /	allt utom en siffra
/\S/	/[^ \r\t\n\f]/	allt utom ett blanktecken

OBS ’\w’ och ’\W’ **bör** känna igen ÅÄÖ som bokstäver.
Dock kan inte detta garanteras! Var uppmärksam på det!

En lite speciell Perlmägd:

\b matchar ordgräns
\B matchar inte ordgräns

/\bi\b/ matchar ’i’ men inte ’pil’ eller ’ill’
/\Bi\B/ matchar ’pil’ men inte ’i’ eller ’ill’

Alternativa strängar:

/Stockholm|Sthlm|STHLM/

Flera tecken

RU	matchar
/a+/	ett eller fler ’a’
/a*/	noll eller oändligt många ’a’
/a?/	noll eller ett ’a’

’bar’ =~ /ba+r/	S	’bor’ =~ /ba+/	F
’bar’ =~ /ba*r/	S	’bor’ =~ /ba*/	S
’bar’ =~ /ba?r/	S	’bor’ =~ /ba?/	S

	/ba+r/	/ba*r/	/ba?r/
’bar’	S	S	S
’br’	F	S	S
’baar’	S	S	F

	/b[oa]+r/	/b[oa]*r/	/b[oa]?r/
’bar’	S	S	S
’br’	F	S	S
’baar’	S	S	F
’bor’	S	S	S
’boor’	S	S	F
’boar’	S	S	F
’baor’	S	S	F

Bestämma antalet själv

{min,max}

/ba{2,3}r/ matchar ’baar’ och ’baaar’ men inte ’bar’ eller ’br’ eller ’baaaar’....
/ba{2}r/ matchar endast ’baar’
/ba{,2}r/ matchar ’br’, ’bar’, ’baar’
/ba{2,}r/ matchar ’baar’, ’baaar’,...

Håll koll på era Reguljära uttryck!

Vad upprepas?
/abc*/ ’ab’, ’abc’, ’abcc’, ’abccc’,...
/(abc)*/ ’’, ’abc’,’abcabc’,’abcabcabc’,...

Förankra dem:
/^hej/ ’hej’ som står först i strängen
/hej$/ ’hej’ som står sist i strängen
/\bhej\b/ ’hej’ som självständigt ord

Inlämningsuppgift Lektion 4

```
Huvudprogrammet:
while(<IN>) {
  chomp;
  if(/<w/) {
    ($lopord,$lemma,$pos,@morf) = bearbeta_rad($_);
    prologisera($lopord,$lemma,$pos,@morf);
  }
}
skriv_ut();
```

Ni ska skriva subrutinerna ”bearbeta_rad”, ”prologisera” och ”skriv_ut”. Självklart får ni ha ytterliggare subrutiner men de ska INTE anropas från huvudkoden.