Perl

~

Redovisning

Övningsuppgifterna är rena övningsuppgifter och på intet sett obligatoriska, men gör dem gärna i alla fall. Det kommer att delas ut inlämningsuppgifter från och med *Lektion 2* Dessa inlämningsuppgifter ser jag helst att de redovisas vid nästkommande lektionstillfälle även om det inte är någon strikt inlämningstid på dessa. Inlämningsuppgifterna får lösas i grupper om två eller tre. De skall dock redovisas individuellt och det kan hända att jag ställer frågor om lösningarna. Om detta är fallet skall varje gruppmedlem kunna besvara den ställda frågan.

Det kommer även att delas ut en lite större inlämningsuppgift som är betygsgrundande och skall lösas samt redovisas individuellt. På denna examensuppgift kommer det att finnas en strikt inlämningstid som måste hållas för att ha chans till VG. För VG krävs även att man presenterar en ”snygg” lösning på exuppgiften. För G krävs att alla inlämningsuppgifter inklusive exuppgiften redovisas.

Redovisning sker genom att man demonstrerar en körning för mig samt att man mailar källkoden till mig. Vid demonstrationen skall programmet vara väl testat.

Betänk dock att det totala betyget på kursen är ett sammanvägt betyg av delmomenten Perl och Prolog II.

Alla filer som behövs för att lösa uppgifterna finns dels på **~martin/pubperl/** på mumin, samt på http://www.nada.kth.se/~xmartin/

Martin Hassel
xmartin@nada.kth.se

Ett stort tack till Lena och Gunnar för grundmaterialet till deras respektive kursomgång samt till Larry Wall for Perl.

Övningsuppgifter Lektion 1

1) Skriv ett program som beräknar omkretsen på en cirkel med radien 12,5. Omkretsen är 2π multiplicerat med radien, där π är ca 3,141592654.
2) Modifiera programmet i föregående övning så att det begär och tar emot en radie från tangentbordet (använd **STDIN**).
3) Skriv ett program som begär och läser in två värden från tangentbordet och som skriver ut resultatet av dessa två värden multiplicerat med varandra.
4) Skriv ett program som begär och läser in en textsträng och ett tal från tangentbordet och skriver ut strängen det antal gånger som det tal som togs emot. Utskriften skall dels göras så att varje utskrift av strängen hamnar på en egen rad och så att alla utskrifterna av strängen hamnar på samma rad. Tips: använd upprepningsoperatorn **x** och i det ena fallet **chomp**.
5) Skriv ett program som begär och läser in en lista med textsträngar från tangentbordet. Varje sträng skall läsas in ”på en egen rad”. Denna lista skall sedan skrivas ut i omvänd ordning. Om du sitter vid en UNIX-terminal måste du antagligen avsluta listan med *filslutstecknet* **CTRL-D**, funkar inte det testa med Plan 9, **CTRL-Z**. Tips: använd funktionen **reverse** och tänk på att läsa in lista i en listvariabel.
6) Skriv ett program som begär och läser in först ett tal och sedan en lista med textsträngar och sedan skriver ut den textsträng som, i listan, motsvaras av det inmatade talet. Tänk på att listans *första* element har *index 0*. Tips: för att ta ut ett värde ur en lista så ”anropar” man listan i skalär kontext; **$element = $lista[2];** som tar ut det *tredje* elementet ur listan **@lista** och placerar det i variabeln **$element**.
7) Skriv ett program som läser in en lista med textsträngar från tangentbordet och sedan skrivet ut en slumpvis utvald textsträng från listan. För att slumpvis välja ut ett element (en sträng) från lista **@lista** så skriv

   srand;

   i början av ditt program (detta initialiserar slumptalsgeneratorn) och använd sedan

   rand(@lista);

   när du behöver ett slumpat värde mellan 0 och längden på (antal element i) listan **@lista** minus ett.

Svaren till ovanstående uppgifter finns i **Appendix A** i Lamaboken (*Learning Perl* av Randal L Schwartz & Tom Christiansen). Dessa uppgifter är i princip identiska med uppgifterna till **kapitel 2 & 3**.

Kom ihåg att, under UNIX, alltid inleda ett perl-program med *Shebangen*. Se också till att skriva luftigt och att alltid använda *deskriptiva variabelnamn* och att *kommentera väl*. Förutom att jag skall förstå er kod så skall ni själva, helst, också förstå den när ni tittar på den x antal månader senare.

Övningsuppgifter Lektion 2

1) Gör ett program som läser in en sträng av tecken mellan *a* och *z* (med respektive versaler) och skriver ut en version med endast gemener, en med endast versaler och en med inledande versal och resten gemener. Använd dig av **uc** och **lc** (läs om dem i t.ex. den utmärkta dokumentationen till ActivePerl [http://velocity.activestate.com/docs/ActivePerl]).
2) Gör ett program som läser in en sträng av tecken mellan *a* och *ö* (med respektive versaler) och skriver ut en version med endast gemener, en med endast versaler och en med stort begynnelse bokstav. Använd dig av **tr//** och **=~**. Tips: Man kan använda split med den tomma strängen för att komma enskilda tecken.
3) Skriv ett program som läser in en text med ett ord på varje rad och matar ut allt utom de grammatiska orden. Använd (den knappast fullständiga) stopplistan i den *associativa arrayen* ("hashen") i filen **grammord.pl**
   Provkör programmet på **kn03.htext** som finns i samma katalog.
   1) Se till att tomma rader inte matas ut.
   2) Undvik också skiljetecken.
4) Utgå från programmet ovan och modifiera det så att bara ord av en given *ordklass* stoppas.
5) Övning 1-5 till kapitel 4 i Lamaboken (sidan 65 i 2:a utgåvan).
6) Övning 1-4 till kapitel 6 i Lamaboken (sidan 75 i 2:a utgåvan).

Inlämningsuppgift Lektion 2

1) Läs in filen **SUC** och ta fram alla rader som innehåller ord. Ta bort all *TEI-kodning*, och skriv ut radnummer, löpord, taggar och lemma i gemener till filen **SUC.lex**, de morfologiska taggarna ska presenteras i alfabetisk ordning. Exempel på utdata i filen:
   n=7 grönskan nn def nom sin utr grönska
   Programmet ska räkna hur många ord det finns i texten och skriva ut detta på skärmen. Var tvåhundrade raden ska det komma ett meddelande på skärmen. OBS! Filen **test** innehåller bara de 100 första raderna från **SUC**, du kan använda den för att testa under utvecklingsfasen.
   Förutom att demonstrera körning av programmet ska du bifoga en sida utskrift av **SUC.lex**.

Kom ihåg att, under UNIX, alltid inleda ett perlprogram med *Shebangen*. Se också till att skriva luftigt och att alltid använda *deskriptiva variabelnamn* och att *kommentera väl*. Förutom att jag skall förstå er kod så skall ni själva, helst, också förstå den när ni tittar på den x antal månader senare.

Övningsuppgifter Lektion 3

1) SUC-verb
Skriv ett program som läser in text i formatet beskrivet nedan. Programmet ska ur texten samla in alla verb, sortera dessa och mata ut verbformerna i bokstavsordning, ett på varje rad.

Det förutsatta informatet ser ut så här:

ORD	TAGG	LEMMA
Smygrustning	NN UTR SIN IND NOM	smygrustning
av	PP	av
raketvapen	NN NEU PLU IND NOM	raketvapen
</s>		
Av	PP	av
MATS	PM NOM	Mats
LUNDEGÅRD	PM NOM	Lundegård
DN:s	PM GEN	DN
korrespondent	NN UTR SIN IND NOM	korrespondent
.	MAD	.

med varje "kolumn" separerad med en tabb (**\t**) och elementen i "TAGG" separerade med mellanslag ( ). Detta format får man om man använder programmet **tabify_SUC** med SUC-text som indata. Hämta detta program till din aktuella katalog.

Provkör ditt program under utveckling på filen **kn03.suc**. När programmet är moget att utsättas för större mängder input, lägg även till filen **SUC** som input.

Så här ska ett anrop från kommandoraden se ut:
tabify_SUC INFIL | DITT_PROGRAM

Programmet skall alltså ta emot indata via en ’pipe’ | och måste därmed använda sig av diamantoperatorn <>. Och så här (ungefär) ska utdata se ut:

[...]
åskådliggjorde
åstadkomma
återberättas
återkom
återkommit
återvänder
öppna
öppnar
överskrida
övertog
[...]

2) Verbkedjor i SUC
   Skapa ett nytt program genom att utgå från det förra (3:1) och modifiera det så att det i stället hittar alla verbkedjor i indata. Med verbkedjor avses två eller fler omedelbart på varandra följande verbformer. Verben i respektive verbkedja ska matas ut på samma rad, separerade av mellanslag.

   Förutsatt indata och anrop är samma som i 3:1 och utdata ska se ut ungefär på följande sätt:

   [...]
   började packa
   kunde komma
   kunde ge
   kunde dricka
   skulle bli
   skulle rida
   skulle få sitta
   hade belamrat
   ha hälsat
   [...]

3) Övning 1 och 2 till kapitel 5 i Lamaboken (sidan 71 i 2:a utgåvan).
4) Övning 1-3 till kapitel 8 i Lamaboken (sidan 100 i 2:a utgåvan).

Inlämningsuppgift Lektion 3

1) Skriv ett program som läser in filen **SUC**, och:
   1. Räknar antal meningar i filen och skriver ut det på skärmen.
   2. Räknar antal förekomster av varje lemma som finns i texten. Detta ska skrivas ut till filen **SUC.frk**
   3. Räkna Type/Token kvoten, (dvs. antal unika ord delat med antal ordförekomster) och skriver ut det på skärmen.
   4. Skriv ut till filen **SUC.lex** alla unika ord som finns i filen på följande format: '**grönskan nn def nom sin utr grönska**', Löpordet ska ha sin kanoniska form, dvs. namn ska börja på versal, akronymer ska ha enbart versaler, mm. Lemmat ska alltså inte ändras. Orden ska presenteras i alfabetisk ordning.

   Använd dig av egna funktioner för att göra programmet *modulärt*. Varje basproblem ska ha sin egen funktion. Var noga med variablernas *räckvidd*, använd **my** och **local**. Det är speciellt viktigt när du bygger listor och hasher i funktionerna. Förutom körning av programmet ska du bifoga en sida utskrift av **SUC.lex** och **SUC.frk**.

Kom ihåg att, under UNIX, alltid inleda ett perlprogram med *Shebangen*. Se också till att skriva luftigt och att alltid använda *deskriptiva variabelnamn* och att *kommentera väl*. Förutom att jag skall förstå er kod så skall ni själva, helst, också förstå den när ni tittar på koden x antal månader senare.

Övningsuppgifter Lektion 4

1) Trigram i SUC
   Utgå från strukturen i program 3:1 och 3:2 och skriv ett program som matar ut trigram över SUC-texter med följande indata (samma som för 3:1 och 3:2):

ORD	TAGG	LEMMA
Smygrustning	NN UTR SIN IND NOM	smygrustning
av	PP	av
raketvapen	NN NEU PLU IND NOM	raketvapen
</s>		
Av	PP	av
MATS	PM NOM	Mats
LUNDEGÅRD	PM NOM	Lundegård
DN:s	PM GEN	DN
korrespondent	NN UTR SIN IND NOM	korrespondent
.	MAD	.

   Trigram är ett vanligt sätt att skaffa information om syntaktisk struktur. Ett trigram är en trio med tre på varandra följande ord. Sålunda skulle trigrammen för exemplet ovan se ut så här:

   Smygrustning av raketvapen
   av raketvapen </s>
   raketvapen </s> Av
   </s> Av MATS
   Av MATS LUNDEGÅRD
   MATS LUNDEGÅRD DN:s
   LUNDEGÅRD DN:s korrespondent
   DN:s korrespondent .

   Skriv programmet så att det lätt går att modifiera för trigram över tagg eller lemmaform istället.

2) Övning 1-3 till kapitel 8 i Lamaboken (sidan 100 i 2:a utgåvan).
3) Övning 1-5 till kapitel 7 i Lamaboken (sidan 100 i 2:a utgåvan).

Inlämningsuppgift Lektion 4

1) Skriv ett program som har som indata en SUC-fil och som utdata fyra filer med respektive substantiv, adjektiv, verb och övriga *ord*. Utfilerna ska kunna användas som lexikon för ett prologprogram och uppslagen ska ha följande notation:

   lex(löpord,Tagglista).

   Ordklasstaggen ska komma först i listan och ha formatet ’cat=tagg’, t.ex. ’cat=nn’. Morfologiska taggar ska göras om enligt följande:

SUC	Görs om till:
UTR/NEU	gender=_
IND/DEF	defness=_
SIN/PLU	numerus=_
SUB/OBJ	prof=_
SIN	numerus=sin
PLU	numerus=plu
NEU	gender=neu
UTR	gender=utr
MAS	gender=mas
DEF	defness=def
IND	defness=ind
NOM	case=nom
GEN	case=gen
SMS	case=sms
PRS	tempus=prs
PRT	tempus=prt
INF	tempus=inf
SUP	tempus=sup
IMP	tempus=imp
KON	mood=kon
PRF	perf=prf
AKT	voice=akt
SFO	voice=sfo
POS	degree=pos
KOM	degree=kom
SUV	degree=suv
SUB	proform=sub
OBJ	proform=obj

Huvudprogrammet ska se ut på följande sätt:

```
#! /usr/local/bin/perl -w
open(IN,'SUC');
while(<IN>) {
   chomp;
   if(/<w/) {
      ($lopord,$lemma,$pos,@morf) = bearbeta_rad($_);
      prologisera($lopord,$lemma,$pos,@morf);
   }
}
skriv_ut();
close(IN);
```

Det ni alltså ska skriva är subrutinerna ”bearbeta_rad”, ”prologisera” och ”skriv_ut”. Självklart får ni ha ytterliggare subrutiner men de ska **inte** anropas från huvudkoden. Lexikonen ni skapar ska vara alfabetiskt ordnade, morftaggarna ska däremot inte vara det, de ska alltså ha sin ursprungliga ordning. Dessutom ska det inte finnas dubbletter i lexikonen.

Skapa fyra stycken hasher, en för varje lexikon! Alltså: en hash med adjektiv, en med verb, en med substantiv och en med resterande ordklasser. De fyra hasherna som byggs upp i *prologisera()* måste vara globala så att *skriv_ut()* kan komma åt dem. Övriga variabler ska **helst** vara *my* eller *local* och måste därför tas emot med @_ och returneras med *return()*.

Nedan följer ett litet utdrag ur en tänkbar **SUC.lex** (så att ni ser hur resultatet skall bli):

```
lex('dessutom', [cat=ab]).
lex('det', [cat=dt, gender=neu, numerus=sin, defness=def]).
lex('det', [cat=pn, gender=neu, numerus=sin, defness=def, prof=_]).
lex('direkt', [cat=ab, degree=pos]).
lex('diskussioner', [cat=nn, gender=utr, numerus=plu, defness=ind, case=nom]).
lex('dit', [cat=ha]).
lex('doftar', [cat=vb, tempus=prs, voice=akt]).
lex('doppresent', [cat=nn, gender=utr, numerus=sin, defness=ind, case=nom]).
lex('driver', [cat=vb, tempus=prs, voice=akt]).
lex('drog', [cat=vb, tempus=prt, voice=akt]).
lex('drygt', [cat=jj, degree=pos, gender=neu, numerus=sin, defness=ind, case=nom]).
```

Kom ihåg att, under UNIX, alltid inleda ett perlprogram med *Shebangen*. Se också till att skriva luftigt och att alltid använda *deskriptiva variabelnamn* och att *kommentera väl*. Alla subrutiner skall även de ha deskriptiva namn. Förutom att jag skall förstå er kod så skall ni själva, helst, också förstå den när ni tittar på koden x antal månader senare.

Övningsuppgifter Lektion 5

1) Svensk verbmorfologi
   Skriv ett litet program som från tangentbordet (eller fil) läser in en imperativform av ett godtyckligt verb följt av en angivelse av vilken konjugation verbet tillhör.

   Programmet ska sedan utmata alla böjda former av verbet (dvs. imperativ presens, preteritum, infinitiv, present particip, perfekt particip [utr. sg.] och supinum).

   Använd gärna "pseudolistor" för hanteringen av verbsuffix. Med "pseudolista" menar jag en sträng med entydiga avgränsare mellan ändelserna. Strängen ska sedan kunna transformeras till en riktig lista med hjälp av 'split'.

   Om svensk verbmorfologi skulle vara ett flyktat stycke kunskap, kan denna åter fångas nedan (ur Dahls grammatik).

Former	Konjugationer			
	1	*2a*	*2b*	*3*
imperativ	kalla	väg	köp	sy
presens	kallar	väger	köper	syr
preteritum	kallade	vägde	köpte	sydde
infinitiv	kalla	väga	köpa	sy
presens particip	kallande	vägande	köpande	syende
prefekt particip	kallad	vägd	köpt	sydd
supinum	kallat	vägt	köpt	sytt

2) Regexmatchning
   Skriv ett litet program som läser rader från **STDIN** och testar raderna mot ett (godtyckligt) reguljärt uttryck. Om den inlästa raden matchar ska raden:

   TRÄFF: <DEN MATCHANDE RADEN>

   matas ut. Om den inlästa raden inte matchar ska raden:

   INGEN TRÄFF I: <DEN MATCHANDE RADEN>

   matas ut.

   Skriv programmet så att det lätt går att redigera det reguljära uttrycket.

3) Modifiera 5.2 så att man slipper meddelande om raden inte matchar och experimentera sedan med egna reguljära uttryck och SUC som input. Spar de reguljära uttrycken ni konstruerar i $kalärer för senare (åter-)användning. Förslag på reguljära uttryck:

    - alla konjunktioner;
    - tvåstaviga lemmaformer;
    - lemmaformer med inledande versal;
    - ordformer med bara versaler;
    - verb där lemmaformen inte slutar på -a;

4) Modifiera reguttryckprogrammet så att det tar SUC-rader som indata direkt, utan att dessa behöver förprocessas av 'tabify_SUC'. Kör programmet med några av de tidigare konstruerade reguttrycken.

Betygsgrundande slutuppgift

0. Inledning

I SUC markeras alla namn med omgivande markörer/"taggar" på detta sätt:

```
<NAME TYPE=PERSON>
<w n=5>MATS<ana><ps>PM<m>NOM<b>Mats</w>
<w n=6>LUNDEGÅRD<ana><ps>PM<m>NOM<b>Lundegård</w>
</NAME>

...

<NAME TYPE=WORK>
<w n=26>Gamla<ana><ps>JJ<m>POS UTR/NEU SIN DEF NOM<b>gammal</w>
<w n=27>testamentet<ana><ps>NN<m>NEU SIN DEF NOM<b>testamente</w>
</NAME>
```

1. Uppgift

Uppgiften består i att extrahera alla sekvenser av ord markerade på ovanstående vis. Utdata ska bestå av en rad som innehåller följande fält separerade med '\t' (tab):

- De ordformer namnet består av.
- Namntyp enligt 'TYPE=' i namntaggen.
- De ordklasser namnet består av.

Slutligen ska utdata innehålla en uppgift om antalet namn i materialet. För det två namnen i exemplet ovan skulle utdata se ut på följande sätt:

```
MATS LUNDEGÅRD          PERSON      PM PM
Gamla testamentet       WORK        JJ NN
=====================================
ANTAL NAMN: 2
```

2. Indata

Indata kan bestå av antingen "rena" SUC-filsrader som ovan eller av sådana rader förbearbetade av 'tabify_SUC'. Om du föredrar detta format måste detta program redigeras så att även namnmarkörerna passerar igenom på samma sätt som '</s>' och '</p>'.

3. Arbetets uppläggning

Det finns naturligtvis ett otal sätt att angripa problemet. De två alternativ som jag omedelbart urskiljer är att antingen:

- Läsa indata rad för rad och successivt spara intressanta rader på ett sätt liknande det i verbkedjeuppgiften.

eller att:

- Läsa in större delar av indata genom av sätta postavgränsaren till något annat än '\n' med hjälp av variabeln '$/'. Ett förslag till alternativ avgränsare skulle kunna vara '</s>', men andra minst lika lämpliga alternativ finns! Sedan kan relevant data exempelvis hämtas med parenteser i reguljära uttryck.

Välj någon av dessa förslag eller gör på något helt annat sätt. Uppgiften behöver heller inte nödvändigtvis lösas i ett och samma program. Skulle det finnas anledning att dela upp bearbetningen i fler program som "pajpar" (Pipe) indata till varandra så är detta acceptabelt.

4. *Extra uppgifter*
   Skulle man vilja meritera sig för VG på detta kursmoment ska man dessutom lösa någon av följande uppgifter:

   - Utöka programmet med analys av data och mata ut (tillsammans med informationen om antal namn) frekvensstatistik över namntyperna samt minst ett av dessa alternativ: information om genomsnittlig ordlängd för ett namn; det längsta namnet i ord räknat; förekommande ordklassekvenser med frekvensstatistik; något annat intressant.

   - Implementera uppgiften på flera olika sätt, exempelvis olika kombinationer av alternativen i 2. och 3. och jämför lösningarnas prestanda med hjälp av flera anrop av unixkommandot 'time', se mansidan för 'time' för mer information.

5. *Redovisning*
   Examinationsuppgiften ska redovisas i form av väl kommenterad kod och ett exempel på resultat. Av praktiska skäl är det bra om jag kan få både en pappersversion och en elektronisk version av dessa saker.

Kom ihåg att, under UNIX, alltid inleda ett perlprogram med *Shebangen*. Se också till att skriva luftigt och att alltid använda *deskriptiva variabelnamn* och att *kommentera väl*. Alla subrutiner skall även de ha deskriptiva namn. Dessutom skall alla subrutiners anrop kommenteras väl, dvs. alla variabler som subrutinen tar emot skall dokumenteras väl, liksom alla värden den returnerar. Man skall kunna förstå *vad* en subrutin *gör* utan att behöva läsa igenom *varje rad*. Viktigt är också att kommentera *exakt* vad ett reguljärt uttryck är tänkt att matcha!

Lycka till!
Martin Hassel…