Laborationer i kursmomentet Datoranvändning E1

`http://www.etek.chalmers.se/~hallgren/Eda/`

Laboration nr 10: Programmering i Maple

1996, 1997 Magnus Bondesson
98-11-23 Thomas Hallgren

1 Introduktion

Syftet denna gång är att ge dig en viss inblick i vad programmering är och hur den går till samt vad vi kan uppnå med sådan. Vi arbetar i Maple men förhoppningen är att du skall få en allmän förståelse. Laborationen går utöver vad som nämns i Gula Boken. Jag försöker dock här att ta med alla erforderliga huvuddrag.

Det ingår också övningar på några användbara UNIX-kommandon, denna gång `which`, `jobs`, `kill` och `ps`.

Detta är den sista laborationen på kursen. Den ska helst utföras i läsvecka 5 (23-27 Nov), men det kommer i begränsad omfattning att finnas handledare tillgängliga även i läsvecka 6. Sista handledda labbpasset (och sista möjligheten att bli godkänd) är på torsdagen den 3 december.

1.1 Förberedelser

Innan du kommer till laborationen är det lämpligt att ha tittat på några sidor i Gula Boken, denna gång avsnitt 9.8 och 9.4.8. Läs också åtminstone avsnitt 2 i detta laborations-PM och fundera igenom några av de inledande Maple-uppgifterna hemma.

1.2 Redovisning

Skriv svaren på alla Maple-uppgifterna i ett och samma Maple-dokument. Spara ofta, så att inte allt går förlorat om Maple skulle krascha. UNIX-uppgifterna redovisas genom svar på frågorna i detta laborations-PM.

2 Om programmering

Programmering handlar om att i vid mening automatisera ett förfarande, dvs i stället för att manuellt genomföra ett antal åtgärder, så skriver vi ett **program** som genomför dem. I alla problemlösningsverktyg finns programmeringsmöjligheter, bl a av det skälet att användaren skall kunna lösa nya problem och inte bara de som verktyget har mer eller färdiga lösningar till.

Det finns en del begrepp som är centrala för all programmering:

- Variabel

- Styrkonstruktionerna räknarsnurra och villkorssnurra
- Styrkonstruktionen villkorligt utförande
- Funktioner/procedurer
- Parametrar
- Rekursion
- Konstruktioner som gör att man klarar sig utan t ex snurror ibland. I Maple finns t ex *seq*, *map*, *add*, *sum*, *mul*, *product* m fl.

Hur dessa ser ut i just Maple framgår av Gula Boken, men låt oss ändå samla huvuddragen här. För detaljer hänvisas till boken.

Räknar-snurrans form:

```
for variabel from start by steg to slut do
      satser ;
od;
```

Semikolonet efter *satser* behövs egentligen inte här eller i det följande och jag sätter inte alltid ut det i exemplen.

Exempel

```
# Beräknar summan 1+2+3+ ... + 10
s:=0;
for i from 1 to 10 do
      s:=s+i;
od;
```

Inmatning av sådana här konstruktioner gör man i praktiken på något av de tre sätten (av de redigeringsskäl som vi redovisat tidigare under kursen):

- Om den är mycket kort: På en rad.
- Om den är litet längre: Rad för rad med `SHIFT/RETURN`-tangenterna mellan raderna. Den sista raden avslutas med retur-tangenten varvid konstruktionen utförs.
- Om den är längre: Mata in den i något redigeringsprogram, t ex *NEdit*. Spara under lämpligt namn. Skriv i Maple: `read 'filnamnet';` och konstruktionen utförs.

Från Maples synpunkt kan kod skrivas in hur som helst, t ex med allt på en rad eller ett ord per rad. Men för att koden skall bli läsvänlig mänskligt sett, bör man **strukturera** den så som vi visar i ovanstående och följande modeller.

Villkors-snurrans form:

```
while villkor do
      satser ;
od;
```

Det finns också en kombinerad version som inleds med for ... to *slut* while *villkor* do.

Exempel

```
# Beräknar summan 1+2+3+ ... + 10
s:=0;i:=1;
while i<=10 do
    s:=s+i;i:=i+1;
od;
```

Villkorskonstruktionen kan ta några olika former

if *villkor* then
      satser ;
fi;
	if *villkor* then
	      satser ;
	else
	      satser ;
	fi;
		if *villkor1* then
		   satser ;
		elif *villkor2* then
		   satser ;
		else
		   satser ;
		fi;

Exempel

```
# Bestämmer det största av två tal x och y, max är ett upptaget namn
if x>y then
   maxi:=x;
else
   maxi:=y;
fi;
```

En **funktionsdefinition** kan ges på ett par olika former. Den första har vi tidigare mött och kan användas när beräkningen inte behöver några hjälpvariabler.

```
funktionsnamn:=(parametrar)->uttryck i parametrarna;
```

Om det bara finns en parameter, behövs inga parenteser. Behövs hjälpvariabler (dessa införs i local-delen) får man ta till den allmännare formen

```
funktionsnamn:=proc(parametrar)
      local lokala_namn;
      satser ;
end;
```

Definitionen "ekas", vilket kan hindras om man ersätter det avslutande semi-kolonet med ett kolon. I båda fallen sker anrop med

funktionsnamn(*aktuella parametrar*);

Exempel

```
> f:=x->x^5;
        f := x -> x^5
> f(2);
        32
> f:=proc(x)
         x^5;
     end;
        f := proc (x) x^5 end
> f(2);
        32
> f:=(x,n)->x^n;
        f :=(x, n) -> x^n
> f(2,4);
        16
> f:=x->if x<0 then -x else x fi;
f := proc (x) options operator, arrow; if x < 0 then -x else x fi end
> f(-3);f(2);
        3
        2
> f:=(x,y)->x^2+y^2;
        f := (x, y) -> x^2+y^2
> f(1,2);
        5
> summa:=proc(n)
             local s,i;
             s:=0;
             for i from 1 to n do
                 s:=s+i;
             od;
          end;
summa := proc (n) local s, i; s := 0; for i to n while true do s :=
s+i od end
> summa(10);
55
> summa:=n->if n=0 then 0 else n+summa(n-1) fi;
summa := proc (n) options operator, arrow; if n = 0 then 0 else
n+summa(n-1) fi end
> summa(10);
55
```

Utskrifter i styrkonstruktioner och funktioner kan vålla en del problem, se sid 186 i Gula Boken. I korthet: Om du tycker att du får för många utskrifter, ändra systemvariabeln *printlevel* till 0 eller -1 före konstruktionen (återställ till standardvärdet 1 efteråt). Om du saknar en utskrift av något, framtvinga den genom att skriva `print(`*`de saker du vill ha utskrivna`*`)`. Känn dig inte tvingad att ta bort överflödiga utskrifter i uppgifterna.

3 Några erinringar angående Maple

Vid det här laget har du förmodligen glömt en del. Observera att tilldelningstecket i Maple är ":=" och att varje kommando skall avslutas med ";" (eller ":" för undertryckande av utskrift). Vid redigering flyttar man sig upp i den tidigare textmassan med musen (markören fixeras med **vänstra mustangenten** inte med någon av de andra två, som har med klistring att göra) eller piltangenter och vid utförandet av ett tidigare kommando (eventuellt redigerat) ersätts det tidigare svaret. Med musen eller piltangent kan man återvända till slutet. Undvik att trycka på returtangenten om något i textmassan är markerat; det markerade tas nämligen som ett kommando. Observera också att liksom i UNIX betyder i Maple stor och liten bokstav olika sak (oftast).

4 Uppgifter

Laborationstiden räcker säkert inte till för att göra allt. Uppgifterna bör göras i angiven ordning och *-märkta uppgifter i första hand. Övriga uppgifter är frivilliga.

4.1 Först några små UNIX-uppgifter

***Uppgift 1. Var finns alla kommandon?**

Kommandon i UNIX är program, som finns i filer. Med kommandot `which` kan man ta reda på var ett kommando finns. Kommandot `which ls` ger svaret `/usr/bin/ls`, t ex.

a. Var finns kommandot `date`? ________________________________

b. Var finns kommandot `v`? ______________________________

c. Var finns `netscape`? _____________________________

I de tidigare laborationerna har vi då och då använt program från katalogen `~hallgren/Intro/` och det har varit lite jobbigare än att använda vanliga kommandon, som t ex `ls`. Hur kommer det sig att det räcker att skriva `ls` för att kommandotolken ska hitta `/usr/bin/ls` när det inte räcker att skriva `giftrans` får att kommandotolken ska hitta `~hallgren/Intro/giftrans`? Svaret är att kommandotolken bara letar i de kataloger som anges av variabeln `$path`.

d. Hur många kataloger letar kommandotolken i? ______

Man kan lätt själv ändra vilka kataloger kommandotolken ska leta i. Antag t ex att vi vill lägga till `~hallgren/Intro/` i listan. Vi kan då skriva

```
set path = ( $path ~hallgren/Intro/ )
```

vilket betyder att det nya värdet på variabeln blir det gamla utökat med `~hallgren/Intro/`.

Anm. Om det finns två kommandon med samma namn får man det som finns i den katalog som står först i listan.

Anm. I hemkatalogen finns en dold kommandofil som heter `.cshrc`. Kommandona i den utförs varje gång en ny kommandotolk startas (t ex när man startar ett terminalfönster). Man kan stoppa in egna kommandon här, t ex för att lägga till egna kommandokataloger till `$path`-variablen.

*Uppgift 2. Att stoppa skenande program

Detta är till viss del repetition av Uppgift 12 och Uppgift 13 i Laboration nr 2. Se även dessa uppgifter.

Ibland händer det att program man startat inte fungerar så bra. Ibland går det kanske t om så galet att det normala sättet att avsluta programmet inte fungerar. Man kan då använda kommandot `kill` för att skicka en stoppsignal till programmet. Kommandot används så här:

`kill %`*`jobbnummer`*	Kommandot `jobs` ger en lista över de program som startats i *samma* terminalfönster. Denna lista innehåller *jobbnummer*, som man kan använda om man vill avbryta program.
`kill `*`processnummer`*	Med kommandot `ps` kan man få en lista över alla program som körs på datorn (egna eller egna+andras). Listan innehåller *processnummer*.
`kill -9 ...`	Avslutar ett program med våld. Programmet får ingen chans att städa upp efter sig.

Se även `man`ualbladen för `kill` och `ps`.

Om man startar ett program som använder för mycket minne blir ofta hela datorn väldig seg och man kanske inte kan göra någonting, inte ens starta ett terminalfönster för att kunna göra `kill` på programmet. Det kan också hända att ett program låser fönstersystemet. Man kan då använda `rlogin` eller `slogin` (se Laboration nr 8) från ett terminalfönster på en annan dator och köra `kill` på det.

- Kör programmet `~hallgren/Intro/mystiskt`. Det kommer att starta en process som du ska avbryta med `kill`. Du kommer att behöva använda `ps` för att ta reda på processnumret. Mer instruktioner skrivs ut då programmet körs.

  När du har avbrutit programmet får du ett kvittensmeddelande. Skriv ner det här:

  ____

4.2 Maple-uppgifter

För att få roligare uppgifter kommer vi successivt att introducera ytterligare detaljfakta om Maple. Meningen är därmed inte att du skall känna dig tvingad att lära dig dessa utantill och de är inte huvudsyftet med uppgifterna, varför jag försöker ge så mycket hjälp som möjligt. Fråga även handledaren vid behov.

Lösningar dyker upp på kursens hemsida så småningom.

***Uppgift 3. En funktion - repetition**

Skriv en Maple-funktion motsvarande $f(x)=x^4-1$. Kalla den t ex FUNK1. Bilda även derivatan FUNK1DER:=D(FUNK1);

***Uppgift 4. En räknarsnurra**

Skriv satser som beräknar och skriver ut [x,f(x),f'(x)] där f är funktionen i förra uppgiften för x=0, 0.1, 0.2, 0.3,..., 0.9, 1.0.

***Uppgift 5. Collatz problem med WHILE och IF**

Låt N vara ett positivt heltal större än 1. Om det är jämnt (kan avgöras med att man testar om N mod 2 = 0), halvera det, i annat fall multiplicera det med 3 och lägg till ett. Om det nya talet är 1, är vi nöjda. Upprepa annars förfarandet med det nya. Det har praktiskt visat sig att man alltid når 1, men ingen har lyckats bevisa det. Exempel: N=5 ger följden: 5,16,8,4,2,1. Skriv satser som givet N skriver ut följden i form av en sekvens eller lista.

Uppgift 6. Primtalstvillingar med WHILE och IF

Två heltal N och N+2 kallas primtalstvillingar om båda är primtal. Skriv satser som bestämmer det första tvillingparet som är större än 700. Gå igenom de tänkbara kandidaterna och bryt när det önskade paret hittats. Påminnes om att funktionen *isprime* avgör om ett tal är primtal eller ej.

Uppgift 7. Primtalstvillingar, forts. med FOR och IF

Skriv satser som på formen [17,19] skriver ut samtliga primtalstvillingar mindre än 2000 och även räknar antalet sådana.

***Uppgift 8. En funktion som ger sista siffran i ett positivt heltal**

Skriv en funktion *Sista(N)* som ger sista siffran i heltalet N. Sista siffran fås som heltalsresten vid division med 10, vilket Maple ger med `N mod 10`.
Exempel: `Sista (184)` ger 4.

***Uppgift 9. En funktion som ger första elementet i en lista**

Någon sådan behöver vi inte eftersom första elementet i en sekvens eller lista L kan fås med L[1], men gör ändå en sådan funktion First(L). Ex: `First([8,1])` ger 8.

Uppgift 10. En funktion som ger de två första elementen i en lista

Skriv en funktion *FirstPair(L)*, vars värde är listan bestående av de två första elementen i listan L.

*Uppgift 11. En funktion som kastar om elementen i en tvåelementlista

Skriv en funktion *Change(L)*, vars värde är en lista med elementen L[1] och L[2] omkastade. T ex skall `Change([7,9]);` ge [9,7].

*Uppgift 12. En funktion som summerar elementen i en lista av tal

Skriv en funktion *Summa(L)*, vars värde är summan av alla elementen (som förutsättes vara tal) i *L*. T ex `Summa([3,1,9]);` skall ge 13.

*Uppgift 13. En egen funktion

Gör en egen funktion som beräknar N-fakultet, dvs fungerar som den inbyggda *factorial*. Funktionen skall alltså beräkna 1*2*3*...*N.

*Uppgift 14. En funktion som letar i en lista

Skriv en funktion *Leta(L,value)*, som letar igenom listan *L* efter värdet *value* och returnerar motsvarande platsnummer (0 om värdet inte finns med). T ex `Leta([7,5,13,8],13);` ger 3.

Uppgift 15. En funktion behöver inte beräkna ett egentligt värde

Skriv en funktion *Rita(f)*, som ritar upp funktionen f(x) på intervallet [-1,1]. T ex skall Rita(x->x^2) rita en del av en parabel.

*Uppgift 16. Automatisera Newtons metod

Newtons metod för att bestämma ett nollställe till en funktion f(x) består väsentligen i Maple i man upprepar

```
x:=x-f(x)/D(f)(x);
```

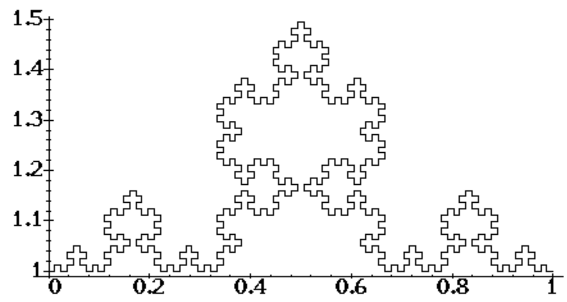
ett antal gånger. Först måste vi naturligtvis ha definierat funktionen och gett x ett startvärde. Automatisera beräkningarna med en snurra. Skriv för varje iterationsvärde ut dels det nya x-värdet, dels skillnaden mot det gamla. **Tips**: `printlevel:=0` före snurran och `print([x,x-xold])` ger precis de önskade utskrifterna.

Uppgift 17. En funktion som bestämmer ett nollställe till en godtycklig funktion

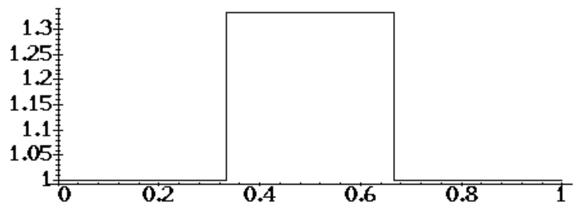
Skriv nu en Maple-funktion *Newton(f,x0)* som beräknar ett nollställe till *f(x)* i närheten av *x0* (eventuellt komplext; metoden fungerar utan ändringar precis lika bra då; för att komplexa nollställen måste man starta med komplext *x0*). Prova den på funktionen f(x)=x^4-1 som ju har de fyra nollställena 1,-1,i,-i.

Låt beräkningen avbrytas då skillnaden mellan de två senaste värdena är liten (t ex abs(x-xold)<0.00001), men gör maximalt 20 iterationer (flera villkor kan kombineras med `and`).

Uppgift 18. Välkommen till fraktalernas värld



Kurvan här intill är ett exempel på en s k fraktalkurva, ett begrepp som myntades av den fransk-amerikanske matematikern Mandelbrot i 1975. Fraktal betyder bruten och det är ju synbart en egenskap som kurvan har. Kurvan börjar i punkten (0,1) och slutar i (1,1). Det är inte ovanligt att man i programmeringskurser med grafikinslag tar upp ritandet av sådana. Dels därför att det kan vara skojigt, men framför allt för att rekursion är en behändig teknik i sammanhanget. Det går bra även i Maple, men man får tänka litet annorlunda än i ett traditionellt programspråk och tidsåtgången blir betydligt större.



Låt oss först förklara idén. Den ovan ritade kurvan är 4:e generationen i en familj vars 0:te generation utgörs av ett rakt horisontellt streck från (0,1) till (1,1) och vars 1:sta generation är kurvan (kallad generatorn eftersom den bestämmer de följande generationernas utseende). Nästa generation, den andra, får vi genom att ersätta de fem streck som bygger upp denna kurva med en förminskad version av generatorn, precis som vi fick den första generationen genom att ersätta det ursprungliga strecket med generatorn. Låt P0 och P1 vara start- och slutpunkt på ett generatorsegment och låt P2, P3, P4 och P5 vara de mellanliggande brytpunkterna (markerade för hand i figuren ovan). Om vi nu vill rita ett visst generationsnummer N, ritar vi helt enkelt närmast lägre nummer mellan P0 och P2, mellan P2 och P3, mellan P3 och P4, mellan P4 och P5 och till sist mellan P5 och P1. Detta är en rekursiv process, som lätt låter sig formuleras. En mindre svårighet är att beräkna koordinaterna för punkterna, så därför ger jag dem här (vi representerar punkter med en tvåelementslista P=[x,y], dvs P[1] är x-koordinaten):

```
dx:=(P1[1] - P0[1])/3: dy:=(P1[2] - P0[2])/3:
P2:=[P0[1]+dx, P0[2]+dy]: P3:=[P2[1]+dy, P2[2]+dx]:
P5:=[P1[1]-dx, P1[2]-dy]: P4:=[P5[1]+dy, P5[2]+dx]:
```

Skriv nu en rekursiv funktion *fract(N,P0,P1)* som genererar en sekvens *S* av samtliga brytningspunkter för kurvan i den N:te generationen, N>=0. Kurvan för N=4 kan sedan ritas med

```
plot([ fract( 4,[0,1],[1,1] )], scaling=constrained);
```

Jag påminner om att två sekvenser *S1* och *S2* kan sättas ihop till en enda med S1,S2.

Uppgift 19. Beräkna π med många siffror

En nyligen (under 1980-talet) upptäckt snabb metod för beräkning av talet π med massor av decimaler lyder som följer

1. Sätt $y_0 = \sqrt{2} - 1$ och $\alpha_0 = 6 - 4\sqrt{2}$

2. Upprepa för n=0,1,2,3,...

2.1. $y_{n+1} = \frac{1-b}{1+b}$, där $b = (1 - y_n^4)^{1/4}$

2.2. $\alpha_{n+1} = (1 + y_{n+1})^4 \alpha_n - 2^{2n+3} y_{n+1}(1 + y_{n+1} + y_{n+1}^2)$

Då närmar sig p=1/α_n talet π mycket snabbt när n växer; antalet riktiga siffror mer än fyrdubblas per upprepningssteg. Verifiera detta praktiskt med Maples hjälp enligt:

1. Sätt antalet siffror som Maple arbetar med till t ex 500 med Digits:=500;.

2. Vi behöver tre variabler y, a och p för y_n, α_n och p ovan. Ge y och a startvärden (n=0).

3. Beräkna i en räknarsnurra med 5 varv y = y_n och a = α_n samt p, för n=1,2,3,4,5. Beräkna med `evalf(p-Pi)` felet och skriv i varje varv ut *p* och felet. Notera hur felets storleksordning avtar.

4. Återställ antalet arbetssiffror med Digits:=10;

5 Avslutande kommentarer

Detta är inte en programmeringskurs, så meningen med denna laboration är som sagt bara att ge en liten inblick i vad programmering kan innebära. E:s programmeringskurs ligger i årskurs 2.

5.1 Information på webben

Hemsidan för kursen Programmeringsteknik E finns på adressen

`http://www.cs.chalmers.se/Cs/Grundutb/Kurser/ept/`

Andra kurser som ges av Institutionen för Datavetenskap finns på

`http://www.cs.chalmers.se/Cs/Grundutb/Kurser/`

Den historiska utvecklingen av datorer och datavetenskap sammanfattas på

`http://math.uwaterloo.ca/~shallit/Courses/134/history.html`

För den som är intresserad av hur program ser ut i olika programspråk rekommenderas sidan

`http://www.ionet.net/~timtroyr/funhouse/beer.html`

där ett och samma program finns översatt till över 200 olika programspråk.