

Databasestøttet Webpublicering, forår 2002

Forelæsning 5

Lister i Tcl

- Lister i Tcl
- Gennemløb af lister
- Andre interessante liste-kommandoer
- Eksempel: email-kartotek
- Eksempel: list reverse
- Udtagning af tilfældigt element fra en liste
- To typer af form-variabler
- Terningekast, frekvensliste og vandrette søjler

Kommandoen lindex

Det er ofte anvendeligt at kunne hente et element ud af en liste.

Hertil bruges kommandoen lindex

```
% set a [list "Grise" "Køer" "Får og gedder" "Hunde"]
Grise Køer {Får og gedder} Hunde
% lindex $a 0
Grise
% lindex $a 1
Køer
% lindex $a 2
Får og gedder
```

Bemærk:

- Elementerne i en liste er indexet fra 0 til $N - 1$, hvor N er antallet af elementer i listen.
- Kommandoen lindex returnerer den tomme streng ("") hvis listen indeholder for få elementer:
% lindex \$a 34
%

Lister i Tcl

Indtil nu har vi kunnet gemme en værdi i en variabel.

Med lister i Tcl kan vi gemme en sekvens af værdier i en variabel.

En værdi kan være et tal, en streng eller endnu en liste!

En liste skabes med kommandoen list

Eksempler:

```
% set a [list] ; # Den tomme liste
% set a [list "Grise"] ; # Liste med et element
Grise
% set a [list "Grise" "Køer"] ; # Liste med to elementer
Grise Køer
% set a [list "Grise" "Køer" "Får og gedder" "Hunde"]
Grise Køer {Får og gedder} Hunde
Kommandoen llength returnerer antallet af elementer i en liste:
% llength $a
4
% llength [list "lise@it.edu" "fenne@it.edu" "bjergoe@it.edu"]
3
```

Påhængning af ekstra elementer til en liste

Kommandoen lappend bruges til at påhænge ekstra elementer til en liste:

```
% set kendte_taarne [list "Sears Tower" "Eiffeltårnet"]
{Sears Tower} Eiffeltårnet
% lappend kendte_taarne "Rundetårn" "Gåsetårnet"
{Sears Tower} Eiffeltårnet Rundetårn Gåsetårnet
```

Kommandoen lappend opdaterer variabel-argumentet:

```
% set kendte_taarne
{Sears Tower} Eiffeltårnet Rundetårn Gåsetårnet
```

Det maksimale antal elementer i en liste er kun begrænset af datamaterns hukommelse.

Opgave: Hvad giver lindex \$kendte_taarne 1?

Opgave: Hvad giver llength \$kendte_taarne?

Gennemløb af lister

Brug kommandoen `foreach` til at gennemløbe en liste:

```
% foreach element $a {
  puts "$element er kloge."
}
Grise er kloge.
Køer er kloge.
Får og gedder er kloge.
Hunde er kloge.
```

Syntaks for kommandoen `foreach`:

```
foreach element list {
  krop
}
```

hvor

- *element* er et navn på en variabel, som kan bruges i kroppen (*krop*) af `foreach`-løkken
- *list* er listen, som skal gennemløbes
- *krop* er kroppen af `foreach`-løkken

Tfhøjskolen

Databasestøttet Webpublicering, forår 2002

Side 5-5

Andre interessante liste-kommandoer

```
% set a
Grise Køer {Får og gedder} Hunde
lrange
returnerer et udsnit af en liste.
% lrange $a 1 2
Køer {Får og gedder}
```

Husk, at første element har indeks 0.

De to tal angiver henholdsvis indeks for det første og sidste element, der skal returneres.

```
linsert
indsætelse af et element et vilkårligt sted i en liste
% linsert $a 3 "Katte"
Grise Køer {Får og gedder} Katte Hunde
% set a
Grise Køer {Får og gedder} Hunde
```

Kommandoen `linsert` opdaterer ikke `a`.

For at opdatere `a` skal vi anvende kommandoen `set`:

```
% set a [linsert $a 3 "Katte"]
Grise Køer {Får og gedder} Katte Hunde
```

Tfhøjskolen

Databasestøttet Webpublicering, forår 2002

Side 5-7

Eksempel: Sum af elementerne i en liste (`sum.tcl`)

Skriv en procedure, der giver en liste med tal som argument, returnerer summen af alle elementer.

```
set l [list 1 2 3 4 5 6 7 8 9]

proc sum {l} {
  set s 0
  foreach e $l {
    ...
  }
  return $s
}
```

Først laves en løsning, som anvender kommandoen `incr`

Programmet afprøves:

```
% source sum.tcl
% sum $l
45
%
```

Der næst laves en løsning, som anvender kommandoen `expr`

Tfhøjskolen

Databasestøttet Webpublicering, forår 2002

Side 5-6

Andre interessante liste-kommandoer - fortsat

```
lsort
sortering af en liste
% lsort $a
{Får og gedder} Grise Hunde Katte Køer
Kommandoen tager et valgfrit argument, der angiver hvorledes
der skal sorteres: -ascii, -integer, -real,
-increasing og -decreasing
% lsort -decreasing $a
Køer Katte Hunde Grise {Får og gedder}
```

`lreplace` erstatter en del af en liste med en ny.

```
% set c [list 0 1 2 3 4 5 6]
0 1 2 3 4 5 6
% lreplace $c 2 4 a b c
0 1 a b c 5 6
```

Erstatter del-listen fra indeks 2 til og med 4 med nye elementer `a`, `b` og `c`.

Tfhøjskolen

Databasestøttet Webpublicering, forår 2002

Side 5-8

Andre interessante liste-kommandoer - fortsat

```
lsearch søgning efter et element i en liste.  
Vi kan søge eksakt:  
% set a  
Grise Køer {Får og gedder} Katte Hunde  
% lsearch $a Katte  
3  
eller efter et mønster:  
% lsearch $a "kat*"  
3  
% lsearch $a "K*"  
1  
Husk, at første element har indeks 0.  
Hvis søgningen følger returneres -1:  
% lsearch $a kat  
-1
```

Eksempel: email-kartotek (email.tcl)

Idé: Brug af lister til at implementere et kartotek over email-adresser og navne.
Kartoteket repræsenteres som en liste af lister:

```
% set kartotek [list [list "nh@it.edu" "Niels Hallenberg"]  
                    [list "kenneth@it.edu" "Kenneth Riis"]]  
                {nh@it.edu {Niels Hallenberg}} {kenneth@it.edu {Kenneth Riis}}
```

Vi implementerer to procedurer:

- person_insert: indsættelse af person i kartotek
- person_lookup: givet en email-adresse, find et navn

Procedure person_insert

```
proc person_insert {kartotek email navn} {  
    return [insert $kartotek 0 [list $email $navn]]  
}
```

Vi indsætter forrest i listen.

Eksempel:

```
% set kartotek [person_insert $kartotek "gates@microsoft.com" "Bill Gates"]  
{gates@microsoft.com {Bill Gates}} {nh@it.edu {Niels Hallenberg}}  
{kenneth@it.edu {Kenneth Riis}}
```

Andre interessante liste-kommandoer - fortsat

```
concat sammensætning af lister  
% set b [list "Slange" "Abe"]  
Slange Abe  
% set a  
Grise Køer {Får og gedder} Katte Hunde  
% concat $a $b  
Grise Køer {Får og gedder} Katte Hunde Slange Abe  
join  
returnerer sammensætningen af elementerne i en liste som en lang streng  
% set emails [list "lise@it.edu" "fenne@it.edu" "bjergoe@it.edu"]  
lise@it.edu fenne@it.edu bjergoe@it.edu  
% join $emails " , "  
lise@it.edu, fenne@it.edu, bjergoe@it.edu  
Kommandoen join er f.eks. anvendelig, hvis man har en liste af personer, som skal modtage  
den samme mail.
```

Eksempel: email-kartotek - fortsat (email.tcl)

Procedure til opslag i kartotek (person_lookup):

```
proc person_lookup {kartotek email} {  
    foreach par $kartotek {  
        if { [string compare [lindex $par 0] $email] == 0 } {  
            # Vi fandt den email-adresse vi kiggede efter!  
            # Andet element i variabelen par indeholder navnet, som skal returneres  
            return [lindex $par 1]  
        }  
    }  
    # Der er ingen email-adresser i listen, som matcher  
    # Vi returnerer den tomme streng  
    return ""  
}
```

Eksempel:

```
% person_lookup $kartotek "kenneth@it.edu"  
Kenneth Riis
```

Bemærk, hvorledes vi anvender return inde i foreach-løkken.

Opgave: Hvorledes søger vi efter navn og ikke email i proceduren person_lookup?

Eksempel: email-kartotek - fortsat (email.tcl)

Lav en procedure email, som returnerer en liste med alle emailadresser i kartoteket:

```
proc emails {kartotek} {
    set emails [list]
    foreach e $kartotek {
        lappend _____ [ _____ ]
    }
    return $emails
}

Proceduren afprøves:

% emails $kartotek
gates@microsoft.com nholt.edu kenneth@it.edu
%
```

Udtagning af et tilfældigt element fra en liste

I sidste uge lærte vi om tilfældige tal.

For at lave en procedure som returnerer et tilfældigt udvalgt element fra en given liste har vi brug for at:

- kende antallet af elementer i listen (length)
- kunne udtage et element givet et indeks ind i listen (index).

```
proc udtag_fra_liste { l } {
    return [index $l [randomRange [length $l]]]
}
```

Eksempel: En web-service, som viser en tilfældigt valgt quote (quotes.tcl).

```
proc home_page { title body } { ... }
proc mk_quote { } {
    set quotes [list "The world will end in 5 minutes. Please log out..."
        lappend quotes "WARNING: Keyboard Not Attached. Press F10 to Continue."
        ...
        lappend quotes "Cannot find REALITY.SYS... Universe Halted."
        return "
    <table bgcolor=white>
    <tr><th>Category: Error Messages</th></tr>
    <tr><td><!--[udtag_fra_liste $quotes]</!--><b>Anon.</b><td></tr>
    </table><p>
    Press the <!--reload</!-->-button to get a new quote."
    ns_return 200 text/html [home_page "quotes" [mk_quote]]
}

Se http://hug.it.edu:8002/F2002/lec5/quotes.tcl
```

Eksempel: list reverse

En liste vendes ved løbende at indsætte det næste element i en ny liste som starter med at være tom:

Original liste	Ny liste
123	
123	1
123	21
123.	321

List reverse - den oprindelige liste ødelægges ikke!

```
proc lrev { l } {
    set res [list]
    foreach e $l {
        set res [linsert $res 0 $e]
    }
    return $res
}
```

Hvad er resultatet af at køre:

```
lrev [list a b c d]
```

Eksemplet typeseiting fra sidste uge (typeseiting4.html)

I sidste uge lærte vi, at flere værdier kan overføres til serveren under samme navn.

Betrakt følgende HTML formular:

```
<form action="typeseiting4.tcl" method=post>
Den tekst du skriver her
<input type=text name=tekst>
bliver formateret således:<p>
```

```
<blockquote>
kursiv: <input type=checkbox name=font value="i"><p>
fed: <input type=checkbox name=font value="b"><p>
understregning: <input type=checkbox name=font value="u"><p>
</blockquote>
```

Du kan vælge mellem sort og gul baggrund:

```
Gron <input type=radio name=farve value="Green">
Gul: <input type=radio name=farve value="Yellow" checked><p>
<input type=reset value="Forfra">
<input type=submit value="Formater tekst">
</form>
```

Variablen font vil antage nul, en eller flere værdier.

To typer af form-variabler

Der er altså to typer af form-variabler:

- form-variabler der kun kan antage én værdi
- form-variabler der kan antage nul, en eller flere værdier; nul, en eller flere værdier repræsenteres bedst ved brug af en liste

De nye procedurer som henter form-variabler find i vores Tcl program:

- `set_form_var form_var`, checker om `form_var` eksisterer, og i givet fald opretter en variabel med navn `form_var`. Hvis `form_var` findes med flere værdier, udvælges en værdi tilfældigt.
- `set_form_var_as_list form_var` checker om `form_var` eksisterer, og i givet fald opretter en variabel med navn `form_var`. Variablen indeholder en liste med alle værdier, som `form_var` er tildelt i HTML formularen.

Eksemplet typesetting fortsat (typesetting4.tcl)

```
proc home_page { title body } { ... }
proc lrev { l } { ... }
set_form_var tekst
set_form_var farve
set_form_var_as_list font

if { [info exists font] } &&
    [info exists tekst] &&
    [info exists farve] } {
    set prefix ""
    set postfix ""
    foreach e $font { append prefix "<$e>" }
    foreach e [lrev $font] { append postfix "</$e>" }

    ns_return 200 text/html [home_page "Typesetting Service" "
<table bgcolor=$farve>
<tr><td>
Teksten $tekst ser således ud med formatering $prefix $tekst $postfix<p>
</td></tr>
</table>" ]
} else {
    ns_return 200 text/html [home_page "Typesetting Service" "Du mangler
at indtaste en eller flere flere værdier." ]
}
```

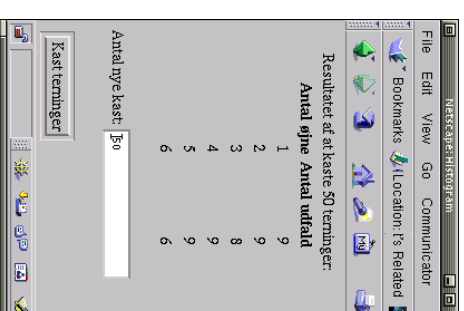
Se `http://hug.it.edu:8002/F2002/lec5/typesetting4.html`

Frekvensliste over terningekast

Antag, at vi kan kaste en terning 6000 gange.

Hvorledes kan vi optælle antallet af 1'ere, 2'ere osv?

Index	0	1	2	3	4	5	6
Indhold							



Optælling af søjlediagram (histogram) ved hjælp af lister

Vi kan lave en metode, som kaster en terning:

```
proc kast_terning { } {
    return [expr [randomRange 6] + 1]
}
```

Hvordan fordeler udfaldene sig ved 6000 kast med en terning, dvs. hvor tilfældig er tilfældigstalsgeneratoren?

Brug en liste `freq` til at indeholde antallet af forekomster af 1, 2, 3, 4, 5, 6.

Element `lindex 1` er antallet af enere, `lindex 2` er antallet af toere, osv.

Til at begynde med er alle antal 0: `set freq [list 0 0 0 0 0 0]`

Når terningen viser 4, så forøg `lindex 4` med 1. Dvs., udtør

```
set freq [lreplace $freq 4 4 [expr [lindex $freq 4] + 1]]
```

I almindelighed, hvis variablen `r` indeholder antal øjne som terningen viser, så udtør

```
set freq [lreplace $freq $r $r [expr [lindex $freq $r] + 1]]
```

hvor

- `lindex $freq $r` læser det nuværende antal

- `expr [lindex $freq $r] + 1` lægger 1 til det nuværende antal

- `set freq [lreplace $freq $r $r [expr [lindex $freq $r] + 1]]` opdaterer listen med det nye antal

Opbygning af histogram (histogram.tcl)

```
proc gen_histogram { antal } {
    set freq [list 0 0 0 0 0 0]
    for {set c 0} {$c < $antal} {incr c} {
        set r [kast_terning]
        set freq [lreplace $freq $r $r [expr [lindex $freq $r] + 1]]
    }
    set res "Resultatet af at kaste $antal terninger:
    <table align=center>
    <tr><th>Antal øjne</th><th>Antal udfald</th></tr></th><tr><th>\n"
    for {set c 1} {$c <= 6} {incr c} {
        append res "<br><td align=right>{$c}</td>
        <td align=right>[lindex $freq $c]</td></tr></th>\n"
    }
    append res "</table>\n"
    return $res
}
set form_var antal
if { [linfo exists antal] } {
    set antal 1000
}
ns_return 200 text/html [home_page "Histogram" "
[gen_histogram $antal]<p>
<form method=post action=histogram.tcl>
Antal nye kast: <input type=text value=\"$antal\" name=antal><p>
<input type=submit value=\"Kast terninger\">"]
Se http://hug.it.edu:8002/F2002/lec5/histogram.tcl
```

Histogram som et vandret søjlediagram, fortsat (histogram2.tcl)

```
proc gen_histogram { antal max_w } {
    set freq [list 0 0 0 0 0 0]
    for {set c 0} {$c < $antal} {incr c} {
        set r [kast_terning]
        set freq [lreplace $freq $r $r [expr [lindex $freq $r] + 1]]
    }
    set max [max_elem $freq]
    for {set c 1} {$c <= 6} {incr c} {
        set w [expr [lindex $freq $c] * $max_w / $max]
        append res "<table width=$w%>
        <tr><td bgcolor=red width=$w% [lindex $freq $c]</td></tr>
        </table>\n"
    }
    return $res
}
ns_return 200 text/html [home_page "Histogram" "
[gen_histogram $antal $max_w]<p>
<form method=post action=histogram2.tcl>
Antal nye kast: <input type=text value=\"$antal\" name=antal><p>
Maks. bredde af søjlediagram i procent af skærmens fulde bredde:
<input type=text value=\"$max_w\" name=max_w><p>
<input type=submit value=\"Kast terninger\">"]
Se http://hug.it.edu:8002/F2002/lec5/histogram2.tcl
```

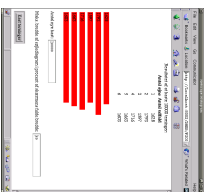
Histogram som et vandret søjlediagram (histogram2.tcl)

I HTML kan vi ved hjælp af tabeller vise vandrette søjler – dvs. ingen billeder er nødvendige.

Det ser bedst ud, når søjlerne skaleres således at den længste søjle får en fast længde, f.eks. 90% af browserens vindue.

For at skalere de resterende søjler skal vi finde et største tal i listen.

```
proc max_elem { l } {
    set max 0
    foreach e $l {
        if {$e > $max} {
            set max $e
        }
    }
    return $max
}
```



Derefter finder vi længden af de enkelte søjler ved $w_i = \frac{f_i \cdot 90}{max}$, hvor w_i er længden af søjle i og f_i er værdien i listen f ved indeks i .

Index i	0	1	2	3	4	5	6
Indhold f_i	0	37	52	35	45	58	53
Søjlelængde w_i	0	57	80	54	70	90	82