

Database-baseret Web-publicering, efterår 2001

Forelæsning 3, tirsdag den 11. september 2001.

Videre med Tcl og dynamiske web-sider

- Indrykning
- For-løkker
- Aritmetiske operatorer
- Procedurer – en gang mere
- Manipulation af strenge
- Overførsel af data fra HTML dokumenter til Tcl-programmer
- Eksempel: Valutahandel
- Ved fejl i dine Tcl-programmer på Web-serveren

Indrykning

Når du ikke laver konsistent indrykning, så er programmet svært at læse:

```
proc urllink { url name } { return "<a href=\"\$url\">\$name</a> "
  proc italic { arg
  } { return "<i>\$arg</i>"
  }
```

```
proc htmlpage { title
  body } {return
  "<html> <head><title>\$title</title>
  </head>
  <body> $body </body>
  </html>" }
```

Sammenlign med `html_utilities.tcl`.

Øvelse 1 – Kommentarservice

Dine to hjemmesider, f.eks.:

1. `www.it-c.dk/courses/W2/E2001/Lb/1b1.html`
2. `www.it-c.dk/courses/W2/E2001/Lb/1b2.html`

Dine kommentarlink bliver (som indsættes i den side du vil kommentere)

```
http://greenspun.com/com/dwebE2001/Lb/1b1.html
og
```

```
http://greenspun.com/com/dwebE2001/Lb/1b2.html
```

På Loquacious er der i databasen registreret følgende:

```
dwebE2001 = www.it-c.dk/courses/W2
```

Informationen `dwebE2001/Lb/1b1.html` og `dwebE2001/Lb/1b2.html` identificerer nu henholdsvis 1 og 2 ovenfor.

Jeg kan kommentere alle sider under `www.it-c.dk/courses/W2/E2001/`

For-løkker – en kortere notation for while-løkker (For1.tcl)

Med for-løkker kan man det samme som med while-løkker.

Hverken mere eller mindre!

```
while1.tcl:
set counter 200
while { $counter >= 1 } {
  puts "I love web-programming"
  incr counter -1
}
```

omskrevet til en for-løkke:

```
for {set counter 200} {$counter >= 1} {incr counter -1} {
  puts "I love web-programming"
}
```

I hvilken rækkefølge udføres kommandoerne?

I vælger selv om I vil anvende while- eller for-løkker.

while-løkker er oftest mere intuitive for begyndere.

For-løkker

En For-løkke har denne form:

```
For {initialisering} {test} {optælling} {
kommandoer
}
```

Den virker sådan:

- (1) Udtør *initialisering*.
- (2) Udtægn *test*; hvis 0, (dvs. falsk) fortsæt efter løkken.
- (3) Udtør løkke-kroppen *kommandoer*.
- (4) Udtør *optælling*.
- (5) Fortsæt med punkt 2.

Kommandoerne under For kaldes *løkketroppen*.

Opgaver (For3.tcl)

Skriv en løkke, der udskriver 64, 32, 16, 8, 4, 2, 1:

```
For {set i ___} {___ >= ___} {set i [expr ___ / ___]} {
puts "$i"
}
```

Skriv en løkke, hvor der udskrives 2, 4, 6, 8, ..., 100:

```
For {set i ___} {___ <= ___} {incr i ___} {
puts "$i"
}
```

Skriv en løkke, hvor der udskrives 100, 110, 120, ..., 200:

```
For {set i ___} {___ <= ___} {incr i ___} {
puts "$i"
}
```

Eksempler (For2.tcl)

Standard-løkke, hvor programmet udskriver _____:

```
For {set i 10} {$i >= 1} {incr i -1} {
puts "$i"
}
```

Standard-løkke, hvor programmet udskriver _____:

```
For {set i 1} {$i <= 10} {incr i 2} {
puts "$i"
}
```

Standard-løkke, hvor programmet udskriver _____:

```
For {set i 1} {$i <= 100} {set i [expr $i * 2]} {
puts "$i"
}
```

En For-løkke kan altid omskrives til en while-løkke

```
For {initialisering} {test} {optælling} {
kommandoer
}
```

omskrevet til while:

```
initialisering
while {test} {
kommandoer
optælling
}
```

Indlejrede løkker – lad os lave en rigtig multiplikationstabel (mul.tcl)

```
For { set i 1 } { $i < 10 } { incr i } {  
  set line ""  
  for { set j 1 } { $j < 10 } { incr j } {  
    append line " [expr $i * $j]"  
  }  
  puts $line  
}
```

Det ydre løkke kører 9 gange. i = 1, 2, ..., 9.

For hvert i kører den indre løkke med j = 1, 2, ..., 9.

Hvor mange gange udføres kommandoen append i alt?

Aritmetiske operatører og deres udregningsrækkefølge (præcedens)

Operator	Betydning	Eksempler
*	Multiplikation	1.5 * 60 24 * 60
/	Division	134.0 / 60 134 / 60
%	Rest	N/A 134 % 60
+	Addition	1.1 + 60 14 + 60
-	Subtraktion	1.1 - 60 134 - 60

Operatører med høj præcedens binder stærkere end operatører med lav præcedens, og udregnes først.

Operatørerne *, / og % har højere præcedens end + og - og udregnes altså før + og -.

Når operatørerne har samme præcedens (binder lige stærkt) regnes fra venstre mod højre.

Ulykker

Hvad er der galt her (ulykke1.tcl):

```
For {set i 0} {$i < 10} {incr i -1} {  
  puts "$i"  
}
```

og her (ulykke2.tcl):

```
set j 0  
while {$j < 10} {  
  incr j 0  
  puts "$j"  
}
```

Eksempel: Hvilken værdi og type har følgende udtryk (ar1.tcl)?

Aritmetisk udtryk	Resultatværdi
1.5 * 60	
150.0 / 60	
150 / 60	
134 % 60	
1.1 + 60	
1.1 - 60	

Vi kan lave vilkårligt komplicerede udtryk, f.eks.:

$22 - 34 + 43 \% 34 * 23 + 122 / 43 22 * 23 + 43$ der giver 302.92364646

Uden præcedensregler kan udtryk være tvetydige: vil $2 + 4 * 4$ give 24 eller 18?

Eftersom * har højere præcedens end +, udregnes $4 * 4$ først, og derefter lægges 2 til.

Hvis vi vil lægge sammen først, skal vi benytte parenteser: $(2 + 4) * 4$ giver 24.

Eftersom * og / har samme præcedens, regnes fra venstre mod højre: $2 * 5 / 2$ giver 5

Vi kan dividere først ved at sætte en parentes: $2 * (5 / 2)$ giver 4

Sammenligningsoperatorer og deres præcedens

Operator	Betydning	Eksempel
<	Mindre end	$x < 60$
<=	Mindre end eller lig med	$x \leq 60$
>	Større end	$x > 60$
>=	Større end eller lig med	$x \geq 60$
==	Lig med	$x == 60$
!=	Forskellig fra	$x != 60$

De aritmetiske operatorer *, /, %, +, - binder alle stærkere end sammenligningsoperatorerne <, <=, >, >=.

Sammenligningsoperatorerne <, <=, >, >= binder alle stærkere end == og !=.

Logiske operatorer og deres præcedens

Operator	Betydning	Eksempel
!	Ikke (Negation)	$!(x == 60)$
&&	Og (Konjunktion)	$0 \leq x \ \&\& \ x < 60$
	Eller (Disjunktion)	$x < 0 \ \ x \geq 60$

Operatoren ! binder stærkere end && som binder stærkere end ||.

! binder også stærkere end sammenligningsoperatorerne og de aritmetiske operatorer.

&& og || binder svagere end sammenligningsoperatorerne og de aritmetiske operatorer.

Der udregnes fra venstre mod højre.

Hvis *udtryk1* er falsk i *udtryk1 && udtryk2* så udregnes *udtryk2* ikke.

Hvis *udtryk1* er sandt i *udtryk1 || udtryk2* så udregnes *udtryk2* ikke.

Eksempel (arit2.tcl)

Lad *x* have værdien 2 og *y* have værdien 4.

Logisk udtryk	Resultatværdi
$1 == 1$	
$x != y$	
$x < 3 + y$	
$(x + y > 3) == 0$	
$0 != x < 3$	
$x == y == 0$	

Tallet 1 svarer til sand og tallet 0 svarer til falsk.

Eksempel: Hvad er resultatet af følgende logiske udtryk (bool1.tcl)?

Lad *x* = 2 og *y* = 4.

Boolsk udtryk	Resultatværdi
$! 0$	
$! 1$	
$! 1 == 0$	
$!(1 == 0)$	
$1 \ \&\& \ 0$	
$0 \ \ 1$	
$x + y > 3 \ \&\& \ x < y$	
$x + y == 3 \ \ x < 4$	
$x < y \ \&\& \ (3 * 4 == 2 * 6 - 1 * 2 + 2) == !(3 < x)$	

Husk, at 1 svarer til sand og 0 til falsk.

Procedurer igen (multi.tcl)

Proceduren multi tager to parametre, en streng word og et tal n

```
proc multi {word n} {
  set res ""
  while {$n >= 1} {
    append res $word
    set n [expr $n - 1]
  }
  return $res
}
```

Nedenfor kaldes multi med argumenterne "All work and no play makes Jack a dull boy." og 348:

```
puts "[multi "All work and no play makes Jack a dull boy." 348]"
```

Forskellen på brug af puts og return (gang_med_to.tcl)

Eksempel:

```
proc gang_med_to { tal } {
  return [expr 2 * $tal]
}
set a [gang_med_to 230]
puts "Variablen 'a' er sat til værdien '$a'"
```

Udskiftes return med puts i proceduren så bliver variablen 'a' sat lig den tomme streng ("") fordi gang_med_to så returnerer den tomme streng:

```
proc gang_med_to { tal } {
  puts [expr 2 * $tal]
}
```

Procedure-kroppen udføres først når proceduren kaldes!

Returværdier fra procedurekald

Hvis en procedure ikke afsluttes med en return-kommando returneres resultatet af den i proceduren sidste udførte kommando.

Eksempel:

```
proc add { a b } {
  return [expr $a + $b]
}
```

kan altså også skrives:

```
proc add { a b } {
  expr $a + $b
}
```

Det anbefales altid at skrive return, således at den værdi der returneres angives eksplicit – programmet er da nemmere at læse.

Eksempel: HTML-utilities (html_utilities.tcl)

```
proc urlLink { url name } {
  return "<a href=\"\$url\"> $name</a>"
}
proc italic { arg } {
  return "<i>$arg</i>"
}
proc htmlpage { title body } {
  return "<html>
<head><title>$title</title></head>
<body>
$body
</body>
</html>"
}
```

Eksempel: HTML-utilities - fortsat (html_utilities.tcl)

Filen html_utilities.tcl hentes ind i tcl1sh forløberen:

```
% source html_utilities.tcl
% set w2_url "http://www.it.edu/courses/W2/E2001/"
http://www.it-c.dk/courses/W2/E2001
% htmlpage "Kursets hjemmeside" "Se [urlink $w2_url "DWEB"]"
<html>
<head><title>Kursets hjemmeside</title></head>
<body>
Se <a href="http://www.it-c.dk/courses/W2/E2001">DWEB</a>
</body>
</html>
%
```

Tegnstringen som htmlpage returnerer kan vises i en browser:

Med ns_return kan vi lave vores første dynamiske web-side (dyn_web.page.tcl).

```
set page [htmlpage "Kursets hjemmeside" "Se [urlink $w2_url "DWEB"]"]
ns_return 200 text/html $page
```

Manipulation af strenge - fortsat (string.tcl)

Udtag en del af en streng med del-kommandoen range:

```
set mystring "Web-programming with Tcl"
puts "range 4 14 af '$mystring' giver '[string range $mystring 4 14]'"
giver
range 4 14 af 'Web-programming with Tcl' giver 'programming'
```

Første tegn i en streng har indeks 0!

Store og små bogstaver med del-kommandoerne tolower og toupper:

```
set myname "Hugo Petersen"
puts "tolower af $myname giver [string tolower $myname]"
puts "toupper af $myname giver [string toupper $myname]"
giver
tolower af Hugo Petersen giver hugo petersen
toupper af Hugo Petersen giver HUGO PETERSEN
```

Manipulation af strenge (string.tcl)

Længden af en streng med kommandoen string og første-argument length:

```
set mystring "Dette er en pæn lang streng"
puts "Længden af $mystring er [string length $mystring] tegn."
```

Sammenligning af strenge med kommandoen string og første-argument compare:

```
puts "A compare B: [string compare "A" "B"]."
```

```
puts "A compare A: [string compare "A" "A"]."
```

```
puts "B compare A: [string compare "B" "A"]."
```

Sløt tegn fra enderne af en streng:

```
set mystring " MyString "
```

```
puts "trim '$mystring' giver: '[string trim $mystring]'"
```

```
puts "lefttrim '$mystring' giver: '[string trimleft $mystring]'"
```

```
puts "righttrim '$mystring' giver: '[string trimright $mystring]'"
```

Tcl på web-serveren

- Sider med endelsen .tcl bliver fortolket af web-serveren
- Tcl-programmer på web-serveren må **ikke** gøre brug af **puts** kommandoen
- I stedet bruges kommandoen ns_return til at sende en HTML side tilbage til en browser:
ns_return 200 text/html " ... "

Et simpelt web-server program (simple.tcl)

Følgende program returnerer en simpel HTML-side til browseren:

```
set name "Niels Hallenberg"
set email "nh@it-c.dk"
ns_return 200 text/html "<html>
<head>
<title>Home page for $name</title>
<body>
Home page for $name <p> <hr>
<a href=\"mailto:$email\">$email</a>
</body>
</html>"
```

Overførsel af data fra HTML dokument til Tcl program

```
my_first.html:
<html>
</body>
<form method=post action=my_first.tcl>
  <input type=text name=login>
  <input type=submit value="Login">
</form>
</body>
</html>

my_first.tcl:
set_form_variables
# Nu er variabelen login tilgængelig
ns_return 200 text/html "<html>
<body>
Du skrev $login i tekstboksen på den forrige HTML side.
</body>
</html>"
```

Kommandoen `set_form_variables` er meget nyttig, idet den erklærer en variabel svarende til hver `input`-boks i HTML dokumentet, f.eks. variabelen `login`.

HTML forms i flere detaljer, fortsat (Form.tcl)

```
set_form_variables
ns_return 200 text/html "<html>
<head>
<title>Form</title>
</head>
<body bgcolor=white>
<center>
<h1>Form - du har trykket på følgende knapper</h1>
Input-boks: $alm_boks<p>
Password-boks: $psw_boks<p>
Valgliste: $sel<p>
Radiogrupper: $radio<p>
Tekstområde: $tekstomraade<p>
Valgknop 1: $valg1<p>
Valgknop 2: $valg2<p>
</center>
</body>
</html>"
```

HTML forms i flere detaljer (Form.html)

```
<html>
<head>
<title>HTML Forms</title>
</head>
</body>
<form method=post action=form.tcl>
  Denne form viser eksempler på forskellige input-bokse:<p>
  En almindelig input-boks: <input type=text name=alm_boks><p>
  En password input-boks: <input type=password name=psw_boks><p>
  En valgliste:<select name=sel><option>Valg 1<option>Valg 2</select><p>
  En radio-gruppe: Knap 1:<input type=radio name=radio value="knap 1">
  Knap 2:<input type=radio name=radio value="knap 2"><p>
  Et tekstområde: <textarea name=tekstomraade cols=20 rows=4></textarea><p>
  Et par valg-knapper: Knap 1:<input type=checkbox name=valg1 value="valg 1">
  Knap 2:<input type=checkbox name=valg2 value="valg 2"><p>
<input type=submit value="Prøv">
</form>
</body>
</html>
```

Valutahandel (Valutahandel.tcl)

For at bygge en valutahandelservice kræves kun to filer.

```
<html>
<head>
<title>Valutahandel</title>
</head>
</body>
<form method=post action=valutaberegning.tcl>
  Hvor mange kroner vil du veksle til dollars? <p>
  <input type=text size=10 name=kroner>
  <input type=submit value="Beregn">
</form>
</body>
</html>
```

Valutahandel - fortsat (valutaberegning.tcl)

```
# set the form-variable 'kroner'
set_form_variables

# compute dollar amount
set dollarkurs 7.34
set dollars [expr ($kroner - 20.0) / $dollarkurs]
ns_return 200 text/html "<html>
<head>
<title>Valutahandel - resultat</title>
</head>
<body bgcolor=white>
<center>
<h1>Valutahandel - resultat</h1>
  For $kroner kroner modtager du \$$dollars <p>
  <a href="//hug.itu.dk:8077/F2001/lec3/valutahandel.html">Tilbage</a>
</center>
</body>
</html>"
```

Valutahandel - test af inddata

Hvilke problemer er der med valutahandel-servicen?

- Hvad sker der når brugeren indtaster et beløb på mindre end kr. 20?
- Hvad sker der hvis brugeren ikke indtaster et tal?

Afsending af email fra Tcl

Kommandoen ns_sendmail bruges til at sende email fra Tcl programmer på web-serveren:

```
ns_sendmail to from subject body
```

Eksempel(email.tcl):

```
set manglende_afleveringer 7
set oevelse 1
ns_sendmail "rh@it.edu" "kenneth@it.edu" "Oevelse $oevelse" "
Hej Niels,
```

Der er stadig \$manglende_afleveringer studerende der ikke har afleveret oevelse \$oevelse.

Mvh. Kenneth"

Web-server fejlhåndtering

• Ved "Server Error" ser du en fejlmeddelelse i din browser:
http://hug.itu.dk:8077/E2001/lec3/fej1.tcl.

```
• "Uendelig løkke" på web-serveren (ulykke3.tcl):
set i 10
while { $i >= 0 } {
  set i 1
}
```

Genstart web-serveren ved brug af "Web Server Services":
http://hug.itu.dk:8077/webserver.html

Øvelse 3

• Brug kurssets Q&A-service:

```
http://www.greenspun.com/board/q-and-a.tcl?topic=Webdesig111
```

• Brug CourseGrader til at aflevere øvelsen: http://hug.itu.dk:8002/vu/index.tcl