Lisp 5: Makroer

Eirik Alderslyst Nygaard Øystein Ingmar Skartsæterhagen

Programvareverkstedet

29. april 2010

(Introduksjon: Litt om evaluering)

(Funksjonskall: Alle argumentformene evalueres først, så kalles funksjonen med de resulterende verdiene som argumenter)

```
(+ (* 2 2) (- 10 9))  ⇒  (+ 4 1)
                      ⇝  5
```

(Spesielle former: Egne regler. Argumentene blir ikke nødvendigvis evaluert, og de kan bli evaluert flere ganger)

```
          (quote foo) ⇝ foo
    (if t 37 (/ 1 0)) ⇝ 37
(loop
  for i from 1 to 5   ⇝ (1 4 9 16 25)
  collect (* i i))
```

(Hva er en makro?)

(En makro er som en spesiell operator vi kan lage selv)

(Dette betyr at vi kan utvide språket ikke bare med nye funksjoner, men også med nye kontrollstrukturer)

(En makro skrives omtrent som en funksjon, men den jobber med uttrykk istedenfor verdier)

(Uttrykkene i et program blir først *makroekspandert*, så *evaluert*)

(Eksempel: WHILE)

(Hvis vi vil ha en *while*-løkke kan vi skrive:)

```
(loop while FOO do BAR)
```

(Hva om vi heller vil skrive

```
(while FOO BAR)
```

?)

(Vi kan lage WHILE selv – som en makro!)

```
(defmacro while (test expr)
  (list 'loop
        'while test
        'do expr))
```

(Eksempel: WHILE)

```
CL-USER> (defvar foo 0)
FOO
CL-USER> (while (< foo 5)
           (progn (format t "~A~%" foo) (incf foo)))
0
1
2
3
4
NIL
CL-USER>
```

(MACROEXPAND, MACROEXPAND-1)

(MACROEXPAND tar inn en form og gir ut formen den ekspanderer til)

(MACROEXPAND-1 gjør det samme, men utfører bare én ekspansjon)

(Vi makroekspanderer WHILE)

```
CL-USER> (macroexpand
          '(while (< foo 5)
            (progn (format t "~A~%" foo) (incf foo))))
(BLOCK NIL
  (SB-LOOP::LOOP-BODY
   NIL
   ((UNLESS (< FOO 5) (GO SB-LOOP::END-LOOP)))
   ((PROGN (FORMAT T "~A~%" FOO) (INCF FOO)))
   ((UNLESS (< FOO 5) (GO SB-LOOP::END-LOOP))) NIL))
T
```

(Vi makroekspanderer WHILE)

```
CL-USER> (macroexpand-1
          '(while (< foo 5)
            (progn (format t "~A~%" foo) (incf foo))))
(LOOP WHILE (< FOO 5)
      DO (PROGN (FORMAT T "~A~%" FOO) (INCF FOO)))
T
CL-USER>
```

(Eksempel: SHOW)

(Sett at vi har lyst til å skrive ut verdien av en variabel `X` på formen «X = 37». Vi kan skrive

```
(format t "X = ~A" x)
```

– men hvis vi gjør dette mange ganger vil vi kanskje lage en funksjon:)

```
(defun show-var (var value)
  (format t "~A = ~A" var value))
```

(Dermed kan vi skrive

```
(show-var 'x x)
```

isteden. Men vi vil heller skrive bare:)

```
(show x)
```

(Vi lager SHOW)

```
(defmacro show (var)
  (list 'format t
        "~A = ~A"
        (list 'quote var)
        var))
```

(Nå vil

```
(show x)
```

ekspanderes til

```
(format t "~A = ~A" 'x x)
```

)

(Kvasisitering)

(I stedet for å bruke `LIST` til å bygge opp uttrykk som returneres fra makroer kan man bruke kvasisitering)

(Kvasisitering fungerer på samme måte som vanlig sitering, men gir deg i tillegg mulighet til å evaluere elementer i listen som er sitert)

(Komma (,) brukes for å evaluere)

```
`(foo bar ,(+ 2 5)) ⇝ (FOO BAR 7)
```

(SHOW med kvasisitering)

```
(defmacro show (var)
  (list 'format t
        "~A = ~A"
        (list 'quote var)
        var))
```

```
(defmacro show* (var)
  `(format t "~A = ~A" ',var ,var))
```

(Forskjeller på makroer og funksjoner)

(Kall til funksjoner og makroer ser like ut, men oppfører seg forskjellig)

(En funksjon ser bare *verdiene* til argumentene, ikke formene)

(For makroer er det omvendt: De ser bare formene, ikke verdiene)

(En makro returnerer ikke det endelige resultatet av makrokallformen, men et nytt uttrykk som skal evalueres)

(Hva vi kan gjøre med makroer)

(Hindre evaluering (som i `IF` og `COND`))

(Gjenta evaluering (som i `LOOP`))

(Lage lokalt miljø (som `LET`))

(Eksempel: Gjenta evaluering)

```
(defmacro twice (something)
  `(progn ,something
          ,something))
```

```
CL-USER> (twice (format t "mjau~%"))
mjau
mjau
NIL
CL-USER> (macroexpand-1 '(twice (format t "mjau~%")))
(PROGN (FORMAT T "mjau~%") (FORMAT T "mjau~%"))
T
CL-USER>
```

(Eksempel: Ombytting av to verdier)

```
(defmacro swap-wrong (a b)
  `(let ((tmp ,b))
     (setf ,b ,a)
     (setf ,a tmp)))
```

(Hva er problematisk med denne?)

(Problemet med SWAP-WRONG)

```
(defmacro swap-wrong (a b)
  `(let ((tmp ,b))
     (setf ,b ,a)
     (setf ,a tmp)))
```

```
(let ((x 5) (y 37))
  (swap-wrong x y)
  (list x y))          ⇝  (37 5)
```

```
(let ((x 5) (tmp 37))
  (swap-wrong x tmp)
  (list x tmp))        ⇝  (5 37)
```

(Variabelfanging)

```
CL-USER> (macroexpand-1
  '(let ((x 5) (tmp 37)) (swap-wrong x tmp) (list x tmp)))
(LET ((X 5) (TMP 37))
  (SWAP-WRONG X TMP)
  (LIST X TMP))
NIL
CL-USER>
```

(`SWAP-WRONG` fanger variabelnavnet `TMP` fra konteksten der den blir kalt)

(Hvordan kan vi unngå dette?)

(Vi trenger et variabelnavn som garantert aldri vil bli brukt i koden som kaller makroen vår)

(Redningen: GENSYM)

(Funksjonen `GENSYM` lager et nytt symbol som er forskjellig fra alle andre symboler)

```
(gensym)        ⇝ #:G786
(gensym "FOO")  ⇝ #:FOO787
```

(Fungerende SWAP)

```
(defmacro swap (a b)
  (let ((tmp (gensym "SWAP-TMP")))
    `(let ((,tmp ,b))
       (setf ,b ,a)
       (setf ,a ,tmp))))
```

(Parameterlister)

(Parameterlisten til en makro kan inneholde `&optional`, `&key` og `&rest`, som forventet)

(Har også `&body`: Samme som `&rest`, men antyder at resten er en «kropp»)

(Destrukturerende parameterlister)

(Anaforiske makroer)

anafo'risk a2

1. om stil: som bruker anaforer
2. språkv: som viser tilbake til noe tidligere i en setning el. tekst
   a-e pronomen

```
(while (get-something)
  (foo it))
```

(Anaforisk IF)

```
(defmacro aif (test consequence &optional alternative)
  `(let ((it ,test))
     (if it
         consequence
         alternative)))
```

(Anaforisk WHILE)

```
(defmacro awhile (test &body body)
  `(loop
      for it = ,test then ,test
      while it
      do (progn ,@body)))
```

```
CL-USER> (awhile (read) (format t "~A~%" (eval it)))
(+ 2 2)
4
nil
NIL
CL-USER>
```

(Oppgaver)

1. Skriv en makro `LET-LIST` som oppfører seg som `LET`, men tar verdiene fra en liste. Den skal kunne kalles slik:

```
(let-list (a b c) '(3 4 5)
  (format t "a=~A, b=~A, c=~A~%" a b c))
```

   (Dette eksempelet skal skrive ut «a=3, b=4, c=5»)

2. Skriv `ACOND`, den anaforiske varianten av `COND`.
3. Skriv en makro `INFIX` som gir verdien av et uttrykk skrevet på innfiksform (det kan antas at alle operasjoner tar to argumenter og at uttrykket er fullt parentetisert):

$$(\mathit{infix}(3 + (2 * 4))) \rightsquigarrow 11$$

(Løsningsforslag: LET-LIST)

```
(defmacro let-list (vars list &body body)
  `(apply
    (lambda ,vars ,@body)
    ,list))
```

(Løsningsforslag: ACOND)

```
(defmacro acond (&rest clauses)
  (if (not clauses)
      nil
      (let ((test (first (first clauses)))
    (body (rest (first clauses))))
`(aif ,test
      ,@body
      (acond ,@(rest clauses))))))
```

(Moral: Makroer kan være rekursive)

(Løsningsforslag: INFIX)

```
(defun infix-to-prefix (expr)
  (if (listp expr)
      (let ((arg1 (first expr))
    (func (second expr))
    (arg2 (third expr)))
(list func
      (infix-to-prefix arg1)
      (infix-to-prefix arg2)))
      expr))

(defmacro infix (expr)
  (infix-to-prefix expr))
```

(Moral: Det kan være hensiktsmessig å definere en funksjon som gjør mye av jobben til en makro)