Försättsblad till skriftlig tentamen vid Linköpings Universitet

(fylls i av ansvarig)

Datum för tentamen	*2010-06-11*
Sal	*VALMAT*
Tid	*08-12*
Kurskod	*TDDC74*
Provkod	*TEN1*
Kursnamn/benämning	*Programmering-abstraktion och modellering*
Institution	*IDA*
Antal uppgifter som ingår i tentamen	*6*
Antal sidor på tentamen (inkl. försättsbladet)	*6*
Jour/Kursansvarig	*Anders Haraldsson*
Telefon under skrivtid	*Ankn. 1403 eller Mobil: 070-514 77 09*
Besöker salen ca kl.	*9-10*
Kursadministratör (namn + tfnnr + mailadress)	*Anna Grabska Eklund Ankn. 23 62, annek@ida.liu.se*
Tillåtna hjälpmedel	*Inga*
Övrigt	
Vilken typ av papper ska användas, rutigt eller linjerat	*Valfritt*
Antal exemplar i påsen	

Tekniska högskolan vid Linköpings universitet
Institutionen för datavetenskap
Anders Haraldsson

TDDC74 Programmering, abstraktion och modellering

Tentamen

Fredag 11 juni 2010 kl 8-12

Uppgifterna löses direkt på denna uppgiftslapp, som lämnas in i sedvanligt tentamensomslag.

Skriv tydligt så att inte dina lösningar missförstås. Använd väl valda namn på parametrar etc.

Även om det i uppgiften står att du skall skriva en funktion, så får du gärna skriva ytterliggare hjälpfunktioner, som kan vara nödvändiga.

På uppgifterna kan halva poäng utdelas. Totalt kan 24p erhållas.

Betygsgradering:

12-16,5	betyg 3
17-20,5	betyg 4
21-24	betyg 5

Lycka till

Uppgift 1 Beräkning av uttryck (2 poäng)

Vi ger följande uttryck för beräkning. Vilket värde skrivs ut eller om det blir fel, beskriv typen av fel.

```
> (define (a x y) (+ x y z))

> (define b (lambda (z) (list a 1 z)))

> (define z (+ 1 2 3))

> 'a                    =

> a                     =

> (a)                   =

> (a 1 5)               =

> (a z z)               =

> (define c (b 1))

> c                     =

> ((lambda (x y) (+ x x y y)) (car (cdr c)) 5)      =

> ((car  c) 1 2)        =
```

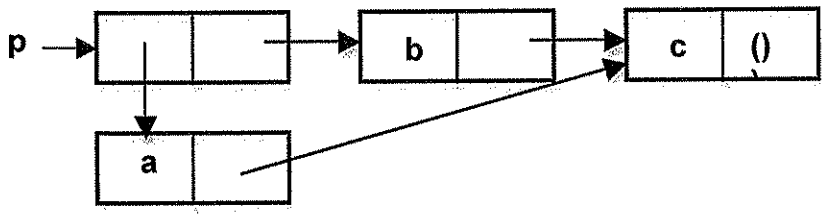
0-1 rätt	0p
2-3 rätt	0,5p
4-5 rätt	1,0p
6-7 rätt	1,5p
8 rätt	2,0p

Uppgift 2. Listor, cons-celler och pekare (3 poäng)

2a Vad blir värdet i parentesformat av följande uttryck. Rita även upp värdet med den grafiska representationen med cons-celler och pekare (box and pointer notation).

```
> (define test-a (list '(1 2) (cons 'a '())))
> test-a  =
```

2b. Skriv Scheme-uttryck som skapar följande struktur:



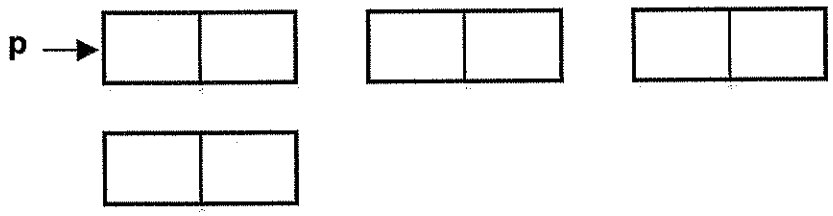
2c. Utgående från strukturen i 2b. Vad blir värdet av p och q (i parentesformat) efter att följande uttryck har beräknats. Visa sedan ändringarna i nedanstående cons-celler längst ned på sidan.

```
> (define q (mcar p))
> (set-mcdr! q (mcdr p))
> (set-mcar! (mcdr p) 'v)
> p   =
> q   =
```

Skriv nedan ett Scheme-uttryck, som länkar ur (tar bort) det andra elementet i listan p. Vilket värde i parentesformat har sedan p och q.

```
> p   =
> q   =
```

Visa nedan hur strukturen ser ut efter ändringarna. Rita pilar och lägg in värden.



Uppgift 3. Rekursiva procedurer (8 poäng)

3a. Skriv en *rekursiv* funktion (kapital *startkapital antal-år ränte-sats*), som beräknar hur kapitalet ökar med räntan efter ett antal är, givet startkapital i kronor, antal år (>= 0)och räntesats (i procent).

Startkapitalet 1000:- efter 2 år med 10% ränta. Efter 1 år är det 1100:-. Efter två år 1210:-

> (kapital 1000 2 10) = 1210

Funktionen får ej definieras så att man med ett enda uttryck direkt beräknar kapitalet, utan skall ske steg för steg med en rekursiv modell. Beskriv sedan om din funktion gör en *rekursiv* (recursive process) eller *iterativ* (linear process) processlösning.

Motivera och visa hur ovanstående exempel beräknas genom att genomföra substitutionsutveckling.

(define (kapital kap år ränta)

3b. Skriv en *rekursiv* funktion ta-fram-talen, som ur en lista med godtyckliga element, returnerar en ny lista med bara de tal (testas med number?), som finns på listans toppnivå. Talen skall i resultatet komma i samma ordning som i ursprungslistan. Beskriver din lösning en rekursiv eller iterativ processlösning?

> (ta-fram-talen '(a 1 (b 2) 3 4 c d 5)) = (1 3 4 5)

(define (ta-fram-talen lista)

Uppgift 3. Rekursiva procedurer, forts

3c. Skriv en *rekursiv* funktion platta-ut, som tar en godtycklig lista (ej med punkterade par). och "plattar" ut den, dvs alla inre parenteser tas bort och alla elementen (oavsett nivå) hamnar på den översta nivån.

```
> (platta-ut '((a b) c (d (e f)) g))          = (a b c d e f g)
```

```
(define (platta-ut lista)
```

3d. Skriv en högre ordningens funktion (sum *term-fn start slut steg-fn*), som summerar en godtycklig term, beskriven som en funktion *term-fn* med index från *start* till *slut* och med steg som anges med funktionen *steg-fn*, som appliceras på det tidigare indexet för att ge nästa index.

Summera alla kvadrater mellan 1 och 5 med steget 1

```
> (sum (lambda (i) (* i i)) 1 5 (lambda (i) (+ i 1)))      = 55
```

Summera varannan fakultet mellan 2 och 8, dvs 2!+4!+6!+8! skrivs

```
> (sum fak 2 8 (lambda (i) (+ i 2)))                       = 41066
```

```
(define (sum term-fn start slut steg-fn)
```

Uppgift 4. ADT – Abstrakta datatyper och objektorientering (3 poäng)

En **ring** är en datastruktur med *ringelement*, som är cirkulär och som har en *toppekare* som anger ett ringelement. Man kan flytta toppekaren framåt eller bakåt för att ange föregående eller nästa ringelement och förflyttningarna är cyklist, dvs man kan gå runt flera varv i strukturen och det finns ingen början eller slut. Man kan lägga till ett nytt element före det ringelement, som toppekaren för anger.

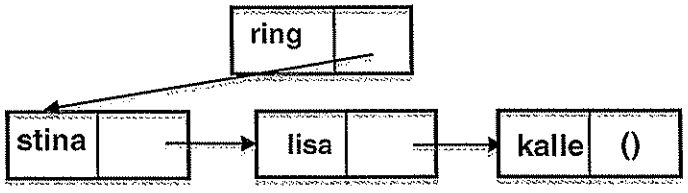
Vi skapar en abstrakt datatyp **ring** med följande primitiva operationer:

```
skapa-ring : -> ring                    ; Skapar en tom ring
lägg-till : element x ring -> ring      ; Lägger in ett element före elementet toppekaren anger
                                        ; Toppekaren anger det nya elementet
ta-bort : ring -> ring                  ; Tar bort det element som toppekaren anger
                                        ; Toppekaren anger nästföljande ringelement
framåt : ring -> ring                   ; Flyttar toppekaren ett steg framåt
bakåt : ring -> ring                    ; Flyttar toppekaren ett steg bakåt
tom-ring? : ring -> sanningsvärde
toppen : ring -> element                ; Returnerar ringelementet, som toppekaren anger
skriv-ut : ring ->                      ; Skriver ut elementen
```

Vi kan nu skapa en ring min-ring och använda operationerna:

```
> (define min-ring (skapa-ring))
> (set! min-ring (lägg-till 'kalle min-ring))
> (set! min-ring (lägg-till 'lisa min-ring))
> (skriv-ut min-ring)                   = [lisa kalle ... ]
> (set! min-ring (lägg-till 'stina min-ring))
> (skriv-ut min-ring)                   = [stina lisa kalle ...]  (*)
> (toppen min-ring)                     = stina
> (set! min-ring (framåt min-ring))
> (toppen min-ring)                     = lisa
> (skriv-ut min-ring)                   = [lisa kalle stina ...]
> (set! min-ring (bakåt min-ring))
> (skriv-ut min-ring)                   = [stina lisa kalle ...]
> (set! min-ring (bakåt min-ring))
> (set! min-ring (bakåt min-ring))
> (toppen min-ring)                     = lisa
> (skriv-ut min-ring)                   = [lisa kalle stina ...]
> (set! min-ring (ta-bort min-ring))
> (skriv-ut min-ring)                   = [kalle stina ...]
> (set! min-ring (ta-bort min-ring))
> (set! min-ring (ta-bort min-ring))
> (skriv-ut min-ring)                   = [...]
> (tom-ring? min-ring)                  = #t
```

Vi representerar ringen med en conscell med symbolen ring och första ringelementet (toppekaren). Ringelementen representeras som en vanlig lista (ej cirkulärt). Att gå framåt och bakåt i ringen innebär att listan måste struktureras om. Nedan visas ringen markerad (*) ovan.



Vi definierar de primitiva operationerna:

```
(define (skapa-ring)  (cons 'ring '()))

(define (lägg-till e ring) (cons 'ring (cons e (cdr ring))))

(define (ta-bort ring) ???)  ; uppgift 4

(define (toppen ring)  (car (cdr ring)))

(define (framåt ring) ???)   : uppgift 4

(define (bakåt ring)
   (if (tom-ring? ring)
      ring
      (cons 'ring (flytta-sista-först (cdr ring)))))

(define (flytta-sista-först lista)
   (define (iter l res)
      (if (null? (cdr l))
         (cons (car l) res)
         (iter (cdr l) (append res (list (car l))))))
   (iter lista '()))

(define (tom-ring? ring) (null? (cdr ring)))

(define (skriv-ut ring)
  (display "[")
  (map (lambda (e) (display e) (display " ")) (cdr ring))
  (display "...]")  (newline))
```

Uppgift:

Definiera funktionerna ta-bort och framåt.

```
(define (ta-bort ring)
```

```
(define (framåt ring)
```

Uppgift 5. Objektorientering (3 poäng)

Föreslå hur du skulle skapa en objektorienterad packetering av den abstrakta datatypen **ring** från uppgift 4, enligt den modell som gjort i denna kurs, t ex i laboration 4. Denna uppgift kan lösas även om du ej löst uppgift 4.

Utgå från koden i uppgift 4 och beskriv hur du skulle göra. Du behöver inte ge fullständig kod utan kan hänvisa till kod, som redan finns beskriven i uppgiften. Det är själva objektorienterade packeteringskoden som är intressant.

Användningen skulle kunna se ut som följer (men även andra modeller kan vara ok):

```
> (define min-r (make-ring))   ; skapa ett objekt med konstruktorn t ex make-ring
> (min-r 'lägg-till 'kalle)    ; skicka meddelande till objektet för att lägg in kalle
> (min-r 'lägg-till 'lisa)
> (min-r 'skriv-ut)            ; skicka meddelande för att skriva ut ringen.
> (min-r 'lägg-till 'stina)
> (min-r 'skriv-ut)
> (min-r 'toppen)              ; skicka meddelande för ge toppelementet som värde
> (min-r 'framåt)
   ... etc ...
```

Uppgift 6. Procedurobjekt och omgivningsdiagram (5 poäng)

6a. Vi har följande definition av en funktion f med två lokalt definierade funktioner g och h.

```
(define (f a)
   (define (g x)
      (+ x a))                    ; frågan är vilket a som avses, finns 2 möjligheter
   (define (h a)
      (g (+ a 1)))
   (h 2))
```

Man kan tänka sig i ett programmeringsspråk två olika svar, beroende på hur programmeringsspråket hanterar bindning av variabler, sk *statiskt* resp *dynamisk* bindning.

(f 5) = 5 ? med dynamisk bindning
(f 5) = 8 ? med statisk bindning

Vad gäller för Scheme? Visa med omgivningsmodellen vilket värde Scheme får, dvs du kan med detta avgöra vilken bindningsmodell Scheme har. Ange i vilken ordning ramar och procedurobjekt skapas.

6b. Vi önskar funktioner som kan ackumulera talvärden, genom att det tidigare sparade värdet förändras med hjälp av någon aritmetisk operation. Initialt gäller värdet 0 och operationen +.

Vi önskar en funktion ack, som skapar sådana ackumulatorfunktioner. Den skapade funktionen tar ett argument, ett tal, en operator eller symbolen value. Ges value som argument returneras det ackumulerade värdet. Exempel.

```
> (define c (ack))      ; en ny ackumulator med initialt värdet 0 och + som operator
> (c 3)                 ; lägg till 3
> (c 'value)            ; = 3, initialt 0+3
> (c 4)                 ; lägg till 4
> (c 1)                 ; lägg till 1
> (c 'value)            ; = 8, dvs 3+4+1
> (c -)                 ; operatorn ändras till -
> (c 7)                 ; dra bort till 7
> (c 'value)            ; =  1 ,dvs 8-7
> (define d (ack))      ; en ny ackumulator med initialt värdet 0 och + som operator
> (d 4)                 ; lägg till 4
> (d *)                 ; operatorn ändras till multiplikation
> (d 3)                 ; multiplicera med 3
> (d 'value)            ; = 12, dvs 4*3
```

```
(define (acc)
```

Rita upp hur omgivningsdiagrammet ser ut efter det att funktionen ack har definierats och de tre första exemplen ovan har utförts.