Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2011-08-17
Sal (1) Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och ringa in vilken sal som avses	TER3
Tid	8-12
Kurskod	TDDC74
Provkod	TEN1
Kursnamn/benämning **Provnamn/benämning**	Programmering - abstraktion och modellering Skriftlig tentamen/duggor
Institution	IDA
Antal uppgifter som ingår i tentamen	6
Jour/Kursansvarig Ange vem som besöker salen	Anders Haraldsson
Telefon under skrivtiden	0705-147709
Besöker salen ca kl.	09:00
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, ankn. 2362, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Vilken typ av papper ska användas, rutigt eller linjerat	
Antal exemplar i påsen	

AID-nummer:	Datum: 2011-08-17	1
Kurskod: TDDC74	Provkod:TEN1	

Tekniska högskolan vid Linköpings universitet
Institutionen för datavetenskap
Anders Haraldsson

TDDC74 Programmering, abstraktion och modellering

Tentamen

Onsdag 17 augusti 2011 8-12

Uppgifterna löses direkt på denna uppgiftslapp, som lämnas in i sedvanligt tentamensomslag. Räcker inte utrymmet kan du komplettera med lösa papper.

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 ytterligare hjälpfunktioner, som kan vara nödvändiga.

Enligt den nya standarden i Scheme/Racket använder man en ny uppsättning funktioner för cons-celler man vill ändra pekare i (dvs muterbara cons-celler), dvs mcons, mcar, mcdr, set-mcar!, set-mcdr! etc. Det går lika bra att använda cons, car, cdr, set-car!, set-cdr!.

Betygsgradering:

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

Lycka till

Uppgift 1. Beräkning av uttryck och quote (2 poäng)

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

```
> (define a 10)

> (define b a)

> (define c 'a)

> (define d (list a (quote b) b c))

> d                                    =

> (a b c)                              =

> '(list a 1 5)                        =

> (define (f a x) (+ a x))

> (f a a)                              =

> (define (g x a)
    (define (h y) (+ y a))
    (h (+ x b))))

> (g b b)                              =

> (h 1)                                =

> (define k (lambda (x y) (y (y x))))

> (k 10 (lambda (x) (+ x 2)))          =

> (k f list)                           =
```

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. Rekursiva funktioner (9 poäng)

2a. (3p) Skriv en *rekursiv* funktion (tabort-tal lst), som tar bort alla talen på toppnivån i en lista.
> (tabort-tal '(a 2 3 b c 4 d)) => (a b c d)

(define (tabort-tal lst)

Beskriver din funktion en *rekursiv* eller *iterativ* (linear) *processkösning*?

Använd din lösning och visa med *substitutionsmodellen* exemplet (tabort-tal '(a 2 c 3)).

2b. (3p) Skriv en funktion (taborta-alla-tal lst), som tar bort alla talen oavsett nivån i en lista.
> (tabort-alla-tal '(a (2 (c 4) e) (5))) => (a ((c) e) ())

(define (tabort-alla-tal lst)

Uppgift 2. Rekursiva funktioner, forts

2c. (2p) Skriv en rekursiv funktion (skriv-ut första nästa -fn slutvillkor-fn), som skriver ut ett antal värden, från första, sedan beräknas nästföljande med funktionen nästa-fn, och sedan nästa etc, tills värdet uppfyller funktionen slutvillkor-fn. Utskriften kan göras med display och newline.

För att skriva ut de udda talen från 1 till 7 skriver vi:

```
> (skriv-ut 1 (lambda (e) (+ e 2))  (lambda (e) (= e 9)))
1
3
5
7
```

```
(define (skriv-ut första nästa-fn slutvillkor-fn)
```

2d. (1p) Använd funktionen skriv-ut för att skriva ut alla primtalen mellan 2 och 100.
Vi antager vi har en funktion prime?, som avgör om ett tal är ett primtal eller ej.

```
(skriv-ut 2 nästa?? slutvillkor??)
2
3
5
etc
```

Komplettera med kod för de två saknade argumenten *nästa??* och *sluvillkor??*

nästa?? =

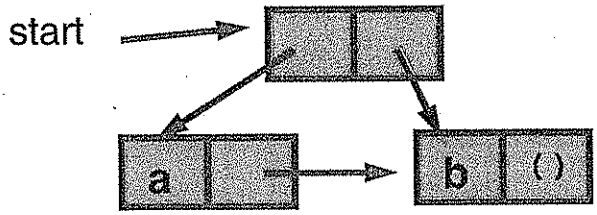
slutvillkor?? =

Uppgift 3. Par/cons-celler (3 poäng)

3a. (1p) 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 start (cons (list 'x) (list 'a '(b))))
start =>
```

3b. (2p) Skriv ett Scheme-uttryck som skapar följande struktur




```
(define start
```

Rita sedan en ny struktur efter det att följande har beräknats:

```
(let ((c-cell (mcons 'c '())))
  (set-mcar! start (mcdr (mcar start)))
  (set-mcdr! (mcdr start)  c-cell)
  (set-mcdr! start c-cell))
```

Uppgift 4. Objektorienterad modellering (3 poäng)

Nedan följer en objektorienterad modellering av personer med sina leksaker. Varje sak har ett namn och kan ha en färg samt varje person har ett namn och en leksakslista.

```
(define (skapa-sak namn)
  (let ((färg 'ingen-färg))
    (lambda (medd . arg-lista)
       (cond ((eq? medd 'namn) namn)
             ((eq? medd 'färg?) färg)
             ((eq? medd 'ny-färg) (set! färg (car arg-lista)))
             (else "fel meddelande")))))

(define (skapa-person namn)
  (let ((leksaker '()))

    (define (lägg-till-leksak sak-obj)
      (set! leksaker
        (cons (cons (sak-obj 'namn) sak-obj) leksaker))
      (void))

   (define (skriv-ut-leksaker)
     (map (lambda (par) (display (car par)) (display " "))
          leksaker)
     (newline))

   (define (leksaker-färg färg)
     ; se uppgift 4c

   (define (dispatch msg)
     (cond ((eq? msg 'namn) (lambda () namn))
           ((eq? msg 'lägg-till-leksak) lägg-till-leksak)
           ((eq? msg 'skriv-ut-leksaker) skriv-ut-leksaker)
           ((eq? msg 'alla-med-viss-färg) leksaker-färg)
           (else (error "fel meddelande"))))
   dispatch))

(define min-docka (skapa-sak "dockan-anna"))
; ge "dockan-anna" färgen gul, se uppgift 4a

(define min-bil (skapa-sak "racerbil"))
; ge "racerbil" färgen röd, se uppgift 4a

(define min-boll (skapa-sak "tennisboll"))
; ge "tennisboll" färgen gul, se uppgift 4a

(define lilla-lisa (skapa-person "Lisa"))
; se uppgifter i 4b
```

Uppgift 4. Objektorienterad modellering, forts

4a. (1p) Skriv Scheme-uttryck, som ger de tre sakerna en färg.

4b. (1p) Skriv Scheme-uttryck, som lägger in de tre sakerna i "Lisa".

4c. (1p) Komplettera koden för leksaker-färg, som ger en lista med namnen på de leksaker som har en given färg. Vi kan då t. ex. få reda på att "dockan-anna" och "tennisboll" är gula.

Uppgift 5. Högre ordningens funktioner (3 poäng)

Skriv först en procedur (compose-2 *f g*), som tar två funktioner *f* och *g* och som *returnerar* den procedur, som utför kompositionen av dessa två funktioner. I matematiken brukar komposition betecknas med en ring o. Vi har sambandet f o g (x) = (f (g x))

```
> (define kadr (compose-2 car cdr))   ; skapa en funktion som tar fram andra elementet
> (kadr '(a b c))  = b
```

Skriv sedan en generell funktion compose, som tar ett godtyckligt antal funktioner (minst 1 st) och som returnerar en funktion som utför funktionskomposition på all dessa funktioner.

```
> ((compose car) '(a b c d))  = a
> (define längd-efter-kdddr (compose length cdr cdr cdr))
> (längd-efter-kdddr  '(a b c d e))  = 2 , dvs längden efter 3 cdr är en lista med 2 element
```

Uppgift 6. Procedurobjekt, omgivningsdiagram (4 poäng)

6a. (2p) I samband med omgivningsdiagram förklara

- bindning (*binding*)
- ram (*frame*)
- omgivning (*environment*)
- procedurobjekt (*procedure object*)

Förklara sedan hur omgivningsdiagrammet används då man i Scheme ger följande uttryck:

- definition: (define variabel uttryck) i en omgivning En
- tilldelning: (set! variabel uttryck) i en omgivning En
- lambda-uttryck: (lambda formella-parametrar kropp) i en omgivning En
- funktionsapplicering: (procedur-uttryck aktuella-parametrar ...) i en omgivning En

Förklara begreppen med hjälp av följande exempel genom att rita upp omgivningsdiagrammet, där ramarna märks ordentligt och ange i den ordning som deluttrycken beräknas och relativt vilken ram samt vad som händer i omgivningsdiagrammet:

```
> (define x 10)
> (define f (lambda (x y) (set! x (+ x 1)) (+ x y)))
> (f x 5) = ?
```

Uppgift 6. Procedurobjekt, omgivningsdiagram, forts

6b. (2p) Vilka värden skrivs ut och rita omgivningsdiagrammet.

```
> (define all '())

> (define (memory-generator start)
      (let ((sum start))
         (lambda (val) (set! sum (+ sum val))
                       (set! all (cons val all))
                        sum)))

> (define adder (memory-generator 10))
> (adder 20) =
> (adder 10) =
> all  =
```

Omgivningsdiagrammet: