Försättsblad till skriftlig tentamen vid Linköpings Universitet

(fylls i av ansvarig)

Datum för tentamen	*2010-08-18*
Sal	-
Tid	*14-18*
Kurskod	*TDDC74*
Provkod	*TEN1*
Kursnamn/benämning	*Programmering- abstraktion och modellering*
Institution	*IDA*
Antal uppgifter som ingår i tentamen	*7*
Antal sidor på tentamen (inkl. försättsbladet)	*6*
Jour/Kursansvarig	*Anders Haraldsson*
Telefon under skrivtid	*281403 / 070 5147709*
Besöker salen ca kl.	*15 och 17*
Kursadministratör (namn + tfnnr + mailadress)	*Anna Grabska Eklund Ankn. 23 62, annek@ida.liu.se*
Tillåtna hjälpmedel	
Övrigt (exempel när resultat kan ses på webben, betygsgränser, visning, övriga salar tentan går i m.m.)	
Vilken typ av papper ska användas, rutigt eller linjerat	*Svaren skrivs på tentamenslappen*
Antal exemplar i påsen	

Tekniska högskolan vid Linköpings universitet
Institutionen för datavetenskap
Anders Haraldsson

TDDC74 Programmering, abstraktion och modellering

Tentamen

Onsdag 18 augusti 2010 kl 14-18

Uppgifterna löses direkt på denna uppgiftslapp, som lämnas in i sedvanligt tentamensomslag. Räcker inte utrymmet kan extra blad lämnas in.

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

Observera: För studenter som gått kursen före vt 2010.
Sedan PRAM-kursen vt 2010 har en ny standard för Scheme använts. Enda skillnaderna är att när man gör pekarändringar i cons-celler måste dessa ha skapats med mcons, och delarna tas ut med mcar och mcdr, samt att ändringar görs med set-mcar! resp set-mcdr!. Vi tillåter i denna tenta även den gamla konventionen där ni bara använder cons, car, cdr, set-car! samt set-cdr!. Man kallar dessa ändringsbara cons-celler för muterbara, därav m'et i namnet. De nya namnen används i första hand i uppgift 2.

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 '(+ 1 2))

> (define b (cdr a))

> (define (f m) (+ m (car b)))

> (define g f)

> 'a                    =

> a                     =

> (b)                   =

> (f a)                 =

> (g (eval a))          =

> (define (h f n) (+ (f n) (g n)))

> (h – 10)              =

> (h (lambda (i) (+ (f (- i 1)) (g i))) 10)  =
```

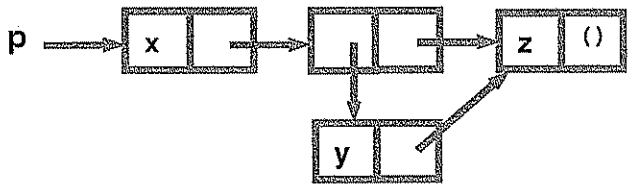
0-1 rätt	0p
2-3 rätt	0,5p
4-5 rätt	1,0p
6-7 rätt	1,5p
8 rätt	2,0p

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

2b. Skriv Scheme-uttryck som skapar följande struktur

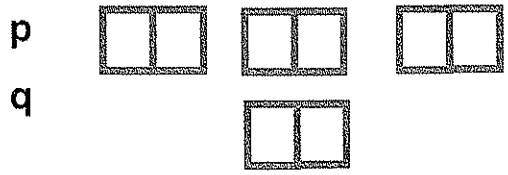


2c. Utgående från strukturen i 2b. Vad blir värdet av p och q efter att följande uttryck har beräknats.

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

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

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



Uppgift 3. Rekursiva funktioner över tal (4 poäng)

2a. (2p) Skriv först två varianter av fakultetsfunktion fak, en som gör en *rekursiv* och en som gör *en iterativ processlösning*[1]. Gör sedan en substitutionsutveckling av (fak 3) för respektive modell.

Iterativ processlösning:

(define (fak n) Gör en substitutionsutveckling av (fak 3)

Rekursiv processlösning:

(define (fak n) Gör en substitutionsutveckling av (fak 3)

[1] När vi anger en egenskap för en funktion kan egenskapen gälla i en underfunktion / hjälpfunktion.

Maclaurin-utveckling. Den elementära funktionen e^x kan beräknas med följande serie:

$$e^x = 1 + x/1! + x^2/2! + x^3/3! + \ldots + x^k/k! + \ldots.$$ (i oändlig serie)

2b. (2p) Skriv sedan en funktion exp, som beräknar e^x med ovanstående serieutveckling[2]. Funktionen skall ta med det antal termer i serien som den globala variabeln antal-termer har.

```
(define antal-termer 100)

(define (exp x)
```

[2] Funktionen för att beräkna x^y är (expt x y), dvs (expt 3 2) = 9

Uppgift 4. Rekursiva funktioner över sekvenser (5 poäng)

4a. (2p) Skriv en funktion filtrera-alla, som tar en godtycklig lista med listor i listor, och tar bort elementen som uppfyller en predikatfunktion. Listan innehåller ej punkterade par. I exemplet skalla alla talen tas bort från en godtycklig listas.

(filtrera-alla '(a 1 2 b (3) (c (4 d))) number?) = (a b (c (d)))

(define (filtrera-alla lista)

4b. (3p) Skriv en funktion summera-samman, som på toppnivån i en lista summerar samman direkt efterföljande tal. Listan kan innehålla godtyckligt antal tal och symboler. I nedanstående första exempel summeras 1+2+3 till 6 samt 4+5+6 till 15:

(summera-samman '(1 2 3 a 4 5 6)) = (6 a 15)
(summera-samman '(a 1 2 3 b 4 c d 5 6 e)) = (a 6 b 4 c d 11 e)

(define (summera-samman lista)

Uppgift 5. Högre ordningens procedurer (3 poäng)

Skriv först en procedur (compose-2 *f g*), som tar två funktioner *f* och *g* (av ett argument) 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 second (compose-2 car cdr)) *; skapa funktionen som tar fram andra elementet*
(second '(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-cdddr (compose length cdr cdr cdr))
> (längd-efter-cdddr '(a b c d e)) = 2 , *dvs längden efter 3 cdr är en lista med 2 element*

Uppgift 6. Objektorienterad modellering (4 poäng)

Nedan följer en objektorienterad modellering av ett kassaskåp, som vi ger en låskombination när skåpet skapas och som vi kan låsa resp. låsa upp och lagra resp. hämta saker om kassaskåpet är upplåst. Varje sak är även ett objekt med ett namn.

```
(define (make-thing name)
      (lambda (msg)
         (cond ((eq? msg 'name) name)
               (else "error; wrong message to make-thing"))))

(define (make-container initial-content)
      (let ((content initial-content))
        (lambda (msg)
           (cond ((eq? msg 'add) ...se uppgift 6c ...)
                 ((eq? msg 'get) (lambda (name)
                                   (set! content (delete-thing[3] name content))))
                 ((eq? msg 'content) content)
                 (else "error; wrong message to make-container")))))

(define (make-safe comb)
      (let ((inner (make-container '()))
            (state 'unlocked))
        (define (unlock input)
           (if (eq? state 'unlocked)
               'already-unlocked
               (if (= input comb)
                   (set! state 'unlocked)
                   'wrong-combination)))
        (lambda (msg)
           (cond ((eq? msg 'unlock) unlock)
                 ((leq? msg 'lock) (lambda () (set! state 'locked))
                 ((eq? msg 'add) ... se uppgift 6c ...)
                 ((eq? msg 'get) ... på samma sätt som för add ...)
                 (else "error; wrong message to make-safe")))))


(define annas-kassaskåp (make-safe 4532))
```

[3] Definitionen på delete-thing är ej nödvändig för att lösa uppgiften.

6a. (2p) Med detta exempel förklara följande begrepp inom objektorientering och visa med exempel på motsvarande begrepp i ovanstående kod:

klass

instans

metod

meddelandesändning

6b. (1p) Skriv Scheme-uttryck som i annas-kassaskåp, som vi nu antager är låst, låser upp detta och sedan lägger in ett objekt med namnet annas-guldring.

6c. (1p) Skriv kod för ... *se uppgift 6c* ... för att kunna lagra ett element i kassaskåpet, om kassaskåpet är öppet. Går det bra returneras värdet ok, annars värdet safe-is-locked.

Uppgift 7. Omgivningsdiagram (3 poäng)

Följande uttryck ges till Scheme:

```
(define (increaser start)
    (let ((value start))
       (lambda (delta)
          (if (= delta 0)
             (set! value start)
             (set! value  (+ delta value)))
          value)))

(define i (increaser 10))
 (i 2)
```

Rita hur omgivningsdiagrammet ser ut efter det att ovanstående 3 uttryck har beräknats. Ange i vilken ordning bindningar och procedurobjekt har skapats och vilka eventuella ändringar som gjorts.

Vilka värden ger sedan nedanstående 3 uttryck

(i 2) => ?
(i 0) => ?
(i 5) => ?