Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2014-08-30
Sal (2) Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och ringa in vilken sal som avses	TER2 TERE
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	Johannes Schmidt
Telefon under skrivtiden	07 25 72 18 03
Besöker salen ca kl.	ja
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, ank. 2362, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Vilken typ av papper ska användas, rutigt eller linjerat	valfritt
Antal exemplar i påsen	

TDDC74 Programmering: Abstraktion och modellering
Tentamen, lördag 30 augusti 2014, kl 08–12

Läs alla frågorna först, och bestäm dig för i vilken ordning du vill lösa uppgifterna.

Skriv tydligt och läsligt. Använd **väl valda namn** på parametrar och **indentera** din kod.

Även om det i uppgiften står att du skall skiva en procedur/funktion, så får du skriva ytterligare hjälpfunktioner som kan vara nödvändiga.

Observera att poängavdrag kan ges för onödigt komplicerade eller ineffektiva lösningar.

Betygsgränser:

12 – 15,5	betyg 3
16 – 19,5	betyg 4
20 – 24	betyg 5

Lycka till!

Uppgift 1. Beräkning av uttryck (4 poäng)

Vi ger DrRacket följande uttryck att beräkna (i ordningen som anges).

Vilka värden returneras? Om det blir fel, beskriv varför (vilken typ av fel).

```
> (define a (+ (cons (+ 9 5) 'p) 0))
> a
> (define x '(1 7 x))
> x
> (define b (caddr x))
> b
> (define c (apply (lambda (a b c) (+ a b)) x))
> c
> (define f (lambda (n m) (- m (cadr x))))
> f
> (define h (lambda (a b c) (f f b)))
> h
> (define z (h 1 2 3))
> z
> (define w (f (f h 42) h))
> w
```

Uppgift 2. Box-pointer diagram (4 poäng)

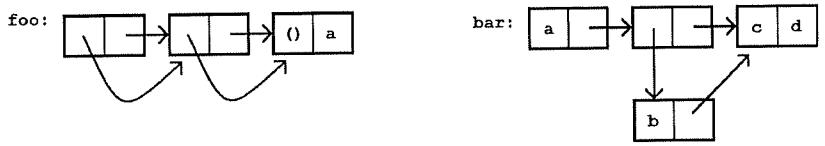
a) Antag att vi har evaluerat följande Scheme-uttryck (i ordning).

```
(define a 57)
(define b (lambda (a b) 2))
(define x (cons a b))
(define y (cons 'a 'b))
(define z (list a 2 3))
```

Rita box-pointer-diagram för dessa strukturer. Var noga med pekarna!

```
x
y
z
(list x y z)
(cons a 'a)
(cons (b a b) (cons 5 (cdr z)))
```

b) Skriv Schemekod som skapar dessa två strukturer. Du får bara använda muterbara par när/om det krävs. Du får använda `define` och `let` var du vill.



Uppgift 3. Omgivningsdiagram (4 poäng)

Betrakta koden nedan:

```
(define function
  (let ((amount 0))
    (lambda message
      (cond
        ((null? message)
         (set! amount (+ 1 amount)) amount)
        ((eq? (car message) 'augment)
         (set! amount (+ amount (cadr message))) amount)
        ((eq? (car message) 'reset)
         (set! amount 0) amount)
        ((eq? (car message) 'get-amount)
         amount)))))

(function)
(function 'get-amount)
(function 'reset)
(function 'augment 37)
(function 'get-amount)
```

Rita omgivningsdiagram som visar det som händer när man evaluerar koden ovan.

Markera globala omgivningen (GE) tydligt, och numrera ramarna i den ordning de skapas (E1, E2,...). Du behöver inte skriva ut all kod i procedurkropparna (body), men det ska tydligt framgå vilken kod du hänvisar till.

Tips Det kan underlätta att skriva om koden utan syntaktiskt socker (exempelvis för `define`- och `let`-uttryck). Det enda du behöver redovisa är dock omgivningsdiagram.

Uppgift 4. Rekursiva procedurer (4 poäng)

Skriv en rekursiv funktion `(tabort-tal lst)`, som tar bort alla tal från toppnivån i en lista.

Så här ska den fungera:

```
> (tabort-tal '(a 2 3 b c 4 d))
(a b c d)

> (tabort-tal '(x (2 3 underlista-behandlas-ej) z 4 w))
(x (2 3 underlista-behandlas-ej) z w)}
```

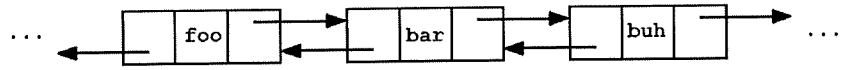
OBS! Du får använda `append` om du vill. Använd `number?` för att testa om ett element är ett tal.

Besvara även följande:

- Skapar din lösning en rekursiv eller en iterativ process?
- Använd din lösning och visa med substitutionsmodellen vad som händer när man evaluerar
  `(tabort-tal '(a 2 c 3))`.

Uppgift 5. Abstrakt datatyp (6 poäng)

Vi vill implementera en dubbellänkad lista (som i föreläsning 9). I vanliga länkade har vi tillgång till nuvarande elements värde, och en pekare till nästa element. I en dubbellänkad lista har vi dessutom pekare till föregående element i listan (så har inte ett par, utan en trippel). Här är ett box-pointer-diagram som beskriver hur det fungerar:



OBS! Läs igenom och försök besvara samtliga deluppgifter, även om du inte har ett fullständigt svar på 5a.

Uppgift 5a (2 poäng) Implementera konstruktorn `(make-triple left value right)`, som skapar ett element i en dubbellänkad lista. Välj intern representation själv. Implementera även getters och setters som svarar mot den representation du har valt: `(get-left triple)`, `(get-value triple)`, `(get-right triple)`, `(set-left! triple left)`, `(set-value! triple value)`, `(set-right! triple right)`.

Exemplet nedan visar hur man ska kunna konstruera en dubbellänkad lista med värdena 1, 2 och 3:

```
> (define el1 (make-triple '() 1 '()))
> (define el2 (make-triple '() 2 '()))
> (define el3 (make-triple '() 3 '()))
> (set-right! el1 el2)
> (set-left! el2 el1)
> (set-right! el2 el3)
> (set-left! el3 el2)
> (define my-dlist el1)
```

OBS!

- Fundera noga på om du vill använda vanliga par/listor eller muterbara (föränderliga).
- För enkelhets skull kan vi representera den tomma dubbellänkade listan med `'()` (den vanliga tomma listan).

Uppgift 5b (1 poäng) Rita ett box-pointer-diagram över `my-dlist` som skapas i exemplet i 5a. Diagrammet ska följa stilen i skissen ovan (och visa hur tripplarna hänger samman, snarare än hur du har valt att implementera dem).

Uppgift 5c (3 poäng) Implementera proceduren `(dlist->list dlist)` som tar en dubbellänkad lista och genererar en vanlig länkad lista med samma innehåll.

Så här ska den fungera (`my-dlist` från uppgift 5a):

```
> (dlist->list my-dlist)
(1 2 3)
```

OBS! `dlist->list` får inte vara beroende av hur du valt att implementera elementen i listan, utan måste använda procedurerna som definierats i 5a.

Uppgift 6. Begreppsfrågor (2 poäng)

Förklara vad en *högre ordningens procedur* är, och ge ett exempel på en sådan. Du kan antingen ge ett exempel på en färdig funktion i Scheme/Racket, eller skriva en egen (bifoga isåfall koden).