Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2015-10-21
Sal (1)	TER4
Tid	14-18
Kurskod	TDDC74
Provkod	TEN1
Kursnamn/benämning **Provnamn/benämning**	Programmering - abstraktion och modellering Skriftlig tentamen/duggor
Institution	IDA
Antal uppgifter som ingår i tentamen	5
Jour/Kursansvarig Ange vem som besöker salen	Johannes Schmidt
Telefon under skrivtiden	0725721803
Besöker salen ca klockan	ca 15:15 och 17:00
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, anna.grabska.eklund@liu.se, ankn. 2362
Tillåtna hjälpmedel	Inga
Övrigt	
Antal exemplar i påsen	

TDDC74 Programmering: Abstraktion och modellering
Tentamen, onsdag 21 oktober 2015, kl 14–18

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.Väl valda namn omfattar exempelvis att inte blanda språk.

Om du vill avsluta *samtliga* parenteser i en funktionsdefinition, kan du om du vill skriva en stor parentes. Observera att detta innebär att du sluter *samtliga* öppna parenteser från och med sista `define` du öppnade.

Även om det i uppgiften står att du skall skriva 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	betyg 3
15,5 – 18,5	betyg 4
19 – 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 (list 'a 'b))
> a
> (define b (cons a 'b))
> b
> (define c ((lambda (a b) 22) b))
> c
> (define flush (lambda (x) 0))
> flush
> (define help (lambda (cmd) flush))
> help
> (define g (lambda (a b) (a b)))
> g
> (define d (help 'help))
> d
> (define r (g d 5))
> r
```

Uppgift 2. Box-pointer diagram (4 poäng)

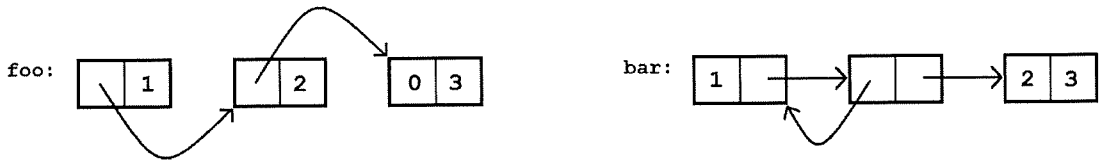
Del a i denna uppgift bör lösas på en enda sida.

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

```
(define x (list 1))
(define y (list 3 4))
(define z (list x y x))
(define w (cons 1 (cdr (cdr z))))
```

Rita box-pointer diagram för strukturerna `x`, `y`, `z`, och `u`. Var noga med pekarna!

b) Skriv Schemekod som skapar dessa två strukturer. Du får endast använda muterbara par på en struktur där det krävs. Du får använda `define` och `let` var du vill.



Uppgift 3. Högre ordningens procedurer (4 poäng)

Repeterande beräkning av en funktions värde är ett viktigt koncept i matematik. Har vi en funktion f och ett startvärde s, så betyder en repetering att vi beräknar $f(s)$. Två repeteringar betyder $f(f(s))$, tre $f(f(f(s)))$, osv. Noll repeteringar står för bara s. Implementera en funktion `repeat` med tre argument, `f`, `times`, `start`, som repeterande applicerar `f` på startvärdet `times` gånger, med startvärde `start`. Du kan anta att `times` alltid är ett positivt heltal.

Här är exempel kod av en användning:

```
> (define (inc n) (+ n 1))
> (define (double n) (* n 2))

> (repeat inc 0 13)
13
> (repeat inc 1 13)
14
> (repeat inc 5 13)
18

> (repeat double 0 3)
3
> (repeat double 1 3)
6
> (repeat double 2 3)
12
> (repeat double 3 3)
24
```

Uppgift 4. Evalueringsmodeller (5 poäng)

Vi betraktar funktionen f som är definierad enligt följande:

$$f(n) = \begin{cases} 3 \text{ om } n = 0 \\ 1 \text{ om } n = 1 \\ 2f(n-2) - f(n-1) \text{ om } n \geq 2 \end{cases}$$

a) **(1,5 poäng)** Implementera funktionen i Racket.

b) **(1,5 poäng)** Använd substitutionsmodellen för att beräkna `(f 2)`.

c) **(2 poäng)** Rita omgivningsdiagram som visar det som händer när man evaluerar `(f 2)`. Markera globala omgivningen (GE) tydligt, och numrera ramarna i den ordning de skapas (E1, E2,...).

Uppgift 5. Data-abstraktion och omgivningsdiagram (7 poäng)

Den här uppgiften har tre delar, men dessa kan lösas oberoende av varandra.

Betrakta följande kod (en egen implementation av muterbara par):

```
(define (mycons a b)
  (lambda (cmd v)
    (cond
      ((eq? cmd 0) a)
      ((eq? cmd 1) b)
      ((eq? cmd 2) (set! a v))
      ((eq? cmd 3) (set! b v)))))

(define (mycar p) (p 0 0))
(define (mycdr p) (p 1 0))
(define (myset-car! p v) (p 2 v))
(define (myset-cdr! p v) (p 3 v))

(define x (mycons 1 2))
(myset-cdr! x 6)
```

a) **(2 poäng)** Vi ger DrRacket följande sex uttryck att beräkna. Vilka värden returneras? Om det blir fel, beskriv varför (vilken typ av fel).

```
x
(mycar x)
(mycdr x)
(myset-car! 8)
(mycar x)
(mycdr x)
```

b) **(3 poäng)** Rita omgivningsdiagram som visar det som händer när man evaluerar den inledande koden i uppgiften (kod i uppgiften, utom det i 5a). 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.

c) **(2 poäng)** Implementera en trippel med hjälp av vår egen implementation av par. Implementerar en konstruktor `make-triple` (med tre parameter) och sedan selektorer `get-left`, `get-middle`, `get-right` så att de fungerar som förväntat. Här är exempel kod av en användning:

```
> (define mytriple (make-triple 1 2 3))
> (get-left mytriple)
1
> (get-middle mytriple)
2
> (get-right mytriple)
3
```