Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2015-02-12
Sal (1)	TER2
Tid	8-10
Kurskod	TDDC74
Provkod	KTR1
Kursnamn/benämning **Provnamn/benämning**	Programmering - abstraktion och modellering Frivillig dugga
Institution	IDA
Antal uppgifter som ingår i tentamen	4
Jour/Kursansvarig Ange vem som besöker salen	Johannes Schmidt
Telefon under skrivtiden	ankn. 13 98
Besöker salen ca klockan	Ja. ca 8:30 och 9:30
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, ankn. 2362, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Antal exemplar i påsen	

TDDC74 Programmering: Abstraktion och modellering
Dugga 1, kl 8—10, 12 feb 2015

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.

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 `define`.

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

OBS! Du får använda `define` för att definiera procedurer och namn. Men du får inte använda det (eller `set!`) för att ändra på/uppdatera variabelvärden. Tanken är att lösningarna ska vara strikt funktionella.

Betyg: Det finns tre duggor i kursen. På varje dugga kan man få som mest 12p (så totalt 36p för alla tre).

För att bli godkänd på en dugga krävs minst 3p.

För att kunna få betyget 3 på duggorna måste du sammanlagt (på alla tre duggor) ha fått minst 18p, för betyget 4 gäller minst 23p och för betyget 5 minst 27p.

För att bli godkänd på duggaserien måste du dessutom ha godkänt på samtliga duggor.

Lycka till!

Uppgift 1 (3 poäng)

Antag att vi har evaluerat följande uttryck i Scheme.

```
(define x (- 1 2))
(define y (* x (- x 1)))
(define inc (lambda (x) (+ x 1)))
(define (g n) (if (< n 3) 1 (+ (g (- x 1)) 1)))
(define (h foo)
  (lambda (x y)
    (inc (foo x))))
```

Vad blir värdet av följande uttryck?

```
1. y
2. (+ x y)
3. (< y x)
4. (g (inc x))
5. (lambda (g h x) (g (h x)))
6. ((h inc) 1 2)
```

Uppgift 2 (3 poäng)

Funktionen f är definierad enligt följande:

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

Implementera funktionen i Scheme.

Använd substitutionsmodellen för att visa vad som händer när man evaluerar uttrycket

```
(f 4)
```

Skapar din lösning en rekursiv eller en iterativ process? Motivera ditt svar.

Uppgift 3 (3 poäng)

Konstruera proceduren `count-occ`, som har parametrarna `digit` och `n`, och som räknar antalet gånger `digit` förekommer i talet `n`.

a) I denna uppgift kan du anta att `digit` är en siffra från 1 till 9 (vi utesluter alltså siffran 0).

Dessa körexempel visar hur det ska fungera:

```
> (count-occ 3 720538437)
2
> (count-occ 8 40256)
0
> (count-occ 3 30)
1
> (count-occ 3 3)
1
> (count-occ 5 0)
0
```

b) Skriv ett program `count-occ2` som kan räkna antalet gånger som en siffra mellan `0-9` förekommer. Funktionaliteten är alltså samma som ovan, men vi tillåter även siffran `0`. `(count-occ2 0 0)` ska bli `1`. Du behöver inte ta hänsyn till inledande nollor i talet (skriver användaren in `007` ska det behandlas som `7`).

Om ditt program från 3[a] redan hanterar nollor, skriv en notis om det på b-uppgiften.

Uppgift 4 (3 poäng)

Vad blir värdet av följande uttryck om det evalueras i Scheme?

```
((lambda (n m)
   (let ((r (remainder n m))
         (z (random m)))
     (- (+ (* r r) z) z)))
 5 3)
```

Skriv sedan om uttrycket utan syntaktiskt socker. Dvs så att **let**-uttrycken ersätts med motsvarande **lambda**-uttryck.