Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2014-02-13
Sal (3) Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och ringa in vilken sal som avses	TER2 TER3 TERE
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	013-14 22 31
Besöker salen ca kl.	ja
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	valfritt
Antal exemplar i påsen	

TDDC74 Programmering: Abstraktion och modellering
Dugga 1, kl 8—10, 13 feb 2014

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 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 (* 2 (- x 1)))
(define square (lambda (x) (* x x)))
(define double (lambda (x) (+ x x)))
(define (f h)
  (lambda (x)
    (square (h (double x)))))
```

Vad blir värdet av följande uttryck?

1. `y`
2. `(+ x y)`
3. `(square (double x))`
4. `(> y x)`
5. `(lambda (x y) (- x y))`
6. `((f square) 1)`

Uppgift 2 (3 poäng)

Funktionen f är definierad enligt följande:

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

Implementera funktionen i Scheme.
Använd substitutionsmodellen för att visa vad som händer när man evaluerar uttrycket

```
(f 3)
```

Skapar din lösning en rekursiv eller en iterativ process?

Uppgift 3 (3 poäng)

Skriv funktionen `product` med två argument, `start` och `end`, som beräknar produkten av alla heltal mellan `start` och `end`. Ni får förutsätta att `start` inte är större än `end`.

Så här skall den fungera:

```
> (product 3 3)
3
> (product 3 4)
12
> (product 4 6)
120
> (product 1 5)
120
```

Skriv därefter funktionen `acc`, med argument `start`, `end` och `proc`. Denna ska samla ihop värden på samma sätt som `product`, men använda (den medskickade) operationen `proc` istället för multiplikation när den kombinerar värden.

Använd sedan `acc` för att omdefiniera funktionen `product` och för att definiera funktionen `sum` som beräknar summan av alla heltal mellan `start` och `end`.

```
> (sum 3 4)
7
> (product 4 6)
120
> (sum 1 5)
15
```

Uppgift 4 (3 poäng)

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

```
(let ((a 5))
  (let ((a 13)
        (b a))
    (+ a (- b 7))))
```

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