Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2011-02-04
Sal (1) Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och ringa in vilken sal som avses	TER1
Tid	14-16
Kurskod	TDDC74
Provkod	KTR1
Kursnamn/benämning **Provnamn/benämning**	Programmering - abstraktion och modellering Frivillig dugga
Institution	IDA
Antal uppgifter som ingår i tentamen	6
Jour/Kursansvarig Ange vem som besöker salen	Anders Haraldsson
Telefon under skrivtiden	070 5147709
Besöker salen ca kl.	15
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund anna.grabska.eklund@liu.se Telefon: 013-28 23 62
Tillåtna hjälpmedel	Inga
Övrigt	
Vilken typ av papper ska användas, rutigt eller linjerat	
Antal exemplar i påsen	

AID-nummer:	Datum: 2011-02-04	1
Kurskod: TDDC74	Provkod: KTR1	

Tekniska högskolan vid Linköpings universitet
Institutionen för datavetenskap
Anders Haraldsson

TDDC74 Programmering, abstraktion och modellering

DUGGA 1

Fredag 4 feb 14-16

Uppgifterna löses direkt på denna uppgiftslapp, som lämnas in i sedvanligt tentamensomslag. Räcker inte utrymmet kan du komplettera med lösa papper.

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 ytterligare hjälpfunktioner, som kan vara nödvändiga.

Betygsgradering: Det är tre duggor. Varje dugga ger 12p, dvs totalt 36p. För att passera en dugga krävs minst 3p på duggan. Totalt skall du på de tre duggorna för betyget 3 ha minst 20p. För betyget 4 minst 25p och för betyget 5 minst 30p.

Lycka till

Uppgift 1 (2 poäng)

Vi gör följande definitioner:

```
(define x  (+ 1 2 3))

(define (f x) (+ (* 2  x) 3))

(define g (lambda (y) (+ x (f y))))

(define (h x) (lambda (y) (* x y)))
```

Vad blir värdet av följande uttryck? Om uttrycket ej går att beräkna, ange vad det blir för fel.

```
x          =

(x)        =

f          =

(f (+ 1 x)) =

(g)        =

(f (g 2))  =

(h 1)      =

((h  2)  3)  =
```

Poängsättning:

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 (1 poäng)

Vi har följande funktion

```
(define (f x)
   (if (< (hämta-värde x) 0)
      0
      (hämta-värde x)))
```

Vi antager vi har en funktion hämta-värde, som i en databas hämtar fram ett värde och som är kostsam, så att man inte vill dubblera anrop till den i onödan.

Vi har diskuterat tre olika sätt att lösa problemet, som vi kan karakterisera som *lokal variabel*, *hjälpfunktion* eller *lambda-uttryck*. Skriv om funktion f på två av dessa sätt så att onödiga dubblerade anrop ej görs.

Första sättet:

```
(define (f x)
```

Andra sättet:

```
(define (f x)
```

Uppgift 3 (3,5 poäng)

I laborationen bearbetade ni enskilda siffror i ett positivt nummer (> 0). Där infördes två primitiver last-digit och but-last-digit, som ni lämpligen använder i denna uppgift. Dessutom infördes number-of-digits, som ni också kan använda i dessa uppgifter.

```
(last-digit 123) = 3
(but-last-digit 123) = 12
(number-of-digits 3421) = 4
```

3a. (1p) Definiera först två funktioner add-digit-last och add-digit-first, som tar ett tal (> 0) och en siffra och lägger till siffran sist resp. först. Lägger vi 0 först i ett tal ges ingen effekt.

```
(add-digit-last 5 123) = 1235
(add-digit-last 0 123) = 1230
(add-digit-first 5 123) = 5123
(add-digit-first 0 123) = 123
```

```
(define (add-digit-last digit number)
```

```
(define (add-digit-first digit number)
```

Uppgift 3, forts

3b. (1,5p) Skriv en funktion number-of-zeros, som räknar antalet nollor i ett tal (> 0). Ett tal kan ej ha inledande nollor.

(number-of-zeros 10020300) = 5

(define (number-of-zeros number)

Beskriver din lösning en *rekursiv* (linear) eller *iterativ* processlösning? Motivera genom att göra en utveckling av (number-of-zeros 102) med substitutionsmodellen.

3c. (1p) Skriv en funktion calc-new-number, som ändrar vissa siffror i ett tal enligt följande: Om siffran < 5 skall den dubbleras, är 5 <= siffran < 8 skall siffran vara som den är, annars skall siffran minskas med 2

(calc-new-number 123456789) = 246856767

(define (calc-new-number number)

Uppgift 4 (2 poäng)

MacLaurinutveckling: Man kan använda summor för att beräkna funktionsvärden. Följande samband gäller:

$$\ln(1+x) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{x^k}{k} \qquad \text{för } |x| < 1$$

Vi vill använda denna formel för att beräkna ln i intervallet 1-2.

Vi antar vi har en global variabel *steps* som anger hur många termer i summan, som skall beräknas. Vill vi räkna med 100 termer skriver vi

```
(define *steps* 100)
```

Topp-funktionen kallar vi calc-ln, som tar ett argument mellan 1 och 2. Lämpligen införs en rekursiv underfunktion som gör själva summeringen av termerna.[1]

```
(define (calc-ln x)   ; 1 <= x < 2
```

[1] Funktionen expt utför "upphöjt", t ex (expt 2 4) beräknar 2^4

Uppgfit 5 (1,5 poäng)

Här kan de primitiva funktionerna från uppgift 3 användas.

5a. (1p) Definiera en funktion remove-digits, som tar ett positivt tal och en villkorsprocedur, som tar bort de siffror som uppfyller villkorsproceduren.

(remove-digits 123456 odd?) = 246 ; tar bort de udda siffrorna (1, 3 och 5)

```
(define (remove-digits number filter-fn)    ; filter-fn tar en siffra som argument
```

5b. (0,5p) Använd remove-digits för att svara på följande frågor:

Vad blir värdet av

```
(remove-digits 123456789 (lambda (d) (or (< d 2) (> d 5))))
=
```

Komplettera funktionen remove-a-digit, som tar bort alla förekomster av ett visst givet tal.

(remove-a-digit 3 132334) = 124 , dvs alla 3'orna tas bort

```
(define (remove-a-digit digit number)
    (remove-digits number _____))
```

Uppgift 6 (2p)

Skriv en funktion average-fn, som tar en funktion f som argument och som returnerar en ny funktion som värde, med följande samband:

$$g(x) = (f(x) + f(x+1)) / 2$$

Vi skall kunna skriva

```
(define (f x) (+ (* x x) 1))
```

((average-fn f) 1) = 3½ , dvs vi har beräknat (f(1) + f(2)) / 2 = (2 + 5) / 2

Vi skulle även kunna definiera g med

```
(define g (average-fn f))
```

och då skriva

```
(g 1)
```

Visa genom noggrann utveckling med substitionsmodellen, vilket värde som erhålls av

```
((average-fn (lambda (x) (+ x 5))) 1)
```