Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2016-03-03
Sal (3)	TER2 TER3 TERE
Tid	8-10
Kurskod	TDDC74
Provkod	KTR2
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	Jalal Maleki Johan Billman
Telefon under skrivtiden	0706-071963
Besöker salen ca klockan	ca. 08:45
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, ankn. 2362, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Antal exemplar i påsen	

Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2016-03-03
Sal (3)	TER2 TER3 TERE
Tid	8-10
Kurskod	TDDC74
Provkod	KTR2
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	Jalal Maleki Johan Billman
Telefon under skrivtiden	0706-071963
Besöker salen ca klockan	ca. 08:45
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, ankn. 2362, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Antal exemplar i påsen	

Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2016-03-03
Sal (3)	TER2 TER3 TERE
Tid	8-10
Kurskod	TDDC74
Provkod	KTR2
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	Jalal Maleki Johan Billman
Telefon under skrivtiden	0706-071963
Besöker salen ca klockan	ca. 08:45
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 2, kl 8—10, 3 mars 2016

Läs alla frågorna först och bestäm dig för i vilken ordning du vill lösa uppgifterna. Uppgifterna är inte ornade i någon särskild ordning!

Skriv tydligt och läsligt. Använd **väl valda namn** på parametrar och **indentera** din kod. Väl valda namn inkluderar konsekvent språk med mera.
Stora parenteser får användas för att avsluta alla öppna parenteser.

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 värden associerade med namn. `mcons` och associerade procedurer får inte heller användas i denna dugga. 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 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.

Lycka till!

Uppgift 1 (3 poäng)

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

```
(define x 12)
(define y 33)
(define z 'yy)
(define a (list x y))
(define b (cons 1 2))
(define c (cons x (cons y z)))
(define d (cons a a))
(define e (cons 'y (cons 'medtek (cons 'matte '()))))
(define f (list (cons 'du 'to) (cons 'två 'do)))
```

Rita "box-pointer"-diagram för de strukturer som a, b, c, d, e och f representerar.

Uppgift 2 (3 poäng)

Skriv en funktion `translate`, vilken givet ett ord och en ordlista med översättningar, returnerar översättningen på ordet. Om ordet inte finns i ordboken skall `#f` returneras. Följande exempel visar hur funktionen skall fungera.:

```
> (define dictionary
    '((bazar . bâzâr) (broder . barâdar) (fader . pedar))
      (du . to) (jag . man) (moder . mâdar) (människa . âdam)
      (namn . nâm) (tak . tâq) (vacker . zibâ) (vän . dust))

> (translate 'jag dictionary)
'man

> (translate 'mamma dictionary)
#f

> (translate 'moder dictionary)
'mâdar
```

Uppgift 3 (3 poäng)

Vi skulle vilja representera och behandla information om vilka filmer personer har sett. I samband med detta skulle vi vilja skapa en dataabstraktion där vi är intresserad av en persons namn (`namn`), födelsedag (`birthday`) och en lista på vilka filmer personen har sett (`films`).

a) Välj en lämplig representation för detta ändamål och definiera följande funktioner för dataabstraktionen.

   1) Konstruktor `(make-person name birthday films)`
   2) Selektorer `person-name`, `person-birthday`, `person-movies`

   Så här tänker vi använda dataabstraktionen:

```
> (define person-1
     (make-person
        '(Linnea Axberg)
        920311
        '((room) (spartacus) (the revenant)
          (local hero))))

> (person-name person-1)
'(Linnea Axberg)

> (person-birthday person-1)
920311

> (person-movies person-1)
'((room) (spartacus) (the revenant) (local hero))))
```

b) Skriv en funktion `common-movies` som tar två personer som argument och returnerar en lista som innehåller de filmer som båda personerna har sett. Om personerna inte sett samma filmer returneras `'()`.

   Exempel:

```
> (define person-2
     (make-person
        '(Anna Persson)
        960104
        '((iceman) (the big short) (jaws)
          (the revenant) (spotlight))))
> (common-movies person-1 person-2)
'((the revenant))
```

Uppgift 4 (3 poäng)

Ett tal kallas *överflödande* om summan av dess riktiga delare[1] är större än talet själv. Alla talets delare förutom talet själv är dess *riktiga* delare. Exempelvis är 12 *överflödande* då summan av dess riktiga delare 1, 2, 3, 4 och 6 är 16.

a) Skriv ett predikat `abundant?` vilket testar om ett tal är överflödande eller ej. Kom ihåg att ni får definiera hjälpprocedurer för att bryta ner problemet. Din lösning skall använda de relevanta procedurerna ur procedursamlingen `enumerate`, `map`, `filter`, och `accumulate` på lämpligt sätt. Se nedan för definitionerna, dessa behöver **inte** skrivas av för att användas.

```
(define (enumerate low high)
  (if (> low high)
      '()
      (cons low (enumerate (+ low 1) high))))

(define (map proc L)
  (if (null? L)
      '()
      (cons (proc (car L)) (map proc (cdr L)))))

(define (filter pred? L)
  (cond ((null? L) '())
        ((pred? (car L))
         (cons (car L)
               (filter pred? (cdr L))))
        (else
         (filter pred? (cdr L)))))

(define (accumulate proc null-value L)
  ; Same as the procedure REDUCE from lectures
  (if (null? L)
      null-value
      (proc (car L)
            (accumulate proc null-value (cdr L)))))
```

[1] *proper divisors* på engelska