# Försättsblad till skriftlig tentamen vid Linköpings universitet

| | |
|---|---|
| **Datum för tentamen** | 2015-03-05 |
| **Sal (2)** | TER2 TERE |
| **Tid** | 8-10 |
| **Kurskod** | TDDC74 |
| **Provkod** | KTR2 |
| **Kursnamn/benämning**<br>**Provnamn/benämning** | Programmering - abstraktion och modellering<br>Frivillig dugga |
| **Institution** | IDA |
| **Antal uppgifter som ingår i tentamen** | 4 |
| **Jour/Kursansvarig**<br>Ange vem som besöker salen | Johannes Schmidt |
| **Telefon under skrivtiden** | ankn. 13 98 |
| **Besöker salen ca klockan** | ca kl. 8:30-9:00 |
| **Kursadministratör/kontaktperson**<br>(namn + tfnr + mailaddress) | Anna Grabska Eklund, anna.grabska.eklund@liu.se, ankn. 2362 |
| **Tillåtna hjälpmedel** | inga |
| **Övrigt** | |
| **Antal exemplar i påsen** | |

# Försättsblad till skriftlig tentamen vid Linköpings universitet

| | |
|---|---|
| **Datum för tentamen** | 2015-03-05 |
| **Sal (2)** | TER2 TERE |
| **Tid** | 8-10 |
| **Kurskod** | TDDC74 |
| **Provkod** | KTR2 |
| **Kursnamn/benämning**<br>**Provnamn/benämning** | Programmering - abstraktion och modellering<br>Frivillig dugga |
| **Institution** | IDA |
| **Antal uppgifter som ingår i tentamen** | 4 |
| **Jour/Kursansvarig**<br>Ange vem som besöker salen | Johannes Schmidt |
| **Telefon under skrivtiden** | ankn. 13 98 |
| **Besöker salen ca klockan** | ca kl. 8:30-9:00 |
| **Kursadministratör/kontaktperson**<br>(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
# Dugga 2, kl 8—10, 5 mars 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)

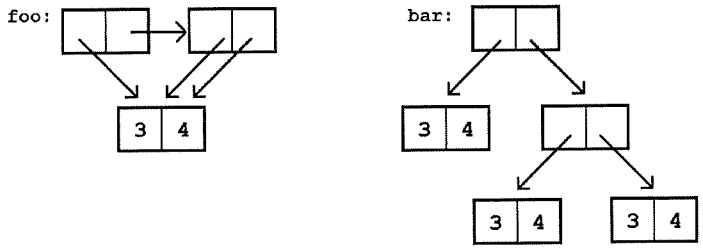
a) Antag att vi har evaluerat följande uttryck i Racket.

```
(define x 42)
(define y (lambda (z) z))
(define a (cons x y))
(define b (cons 'x 'y))
(define c '(0 1 2))
```

Rita box-pointer-diagram för dessa strukturer. Var noga med pekarna!

```
a
b
c
(cons x (cons c (cons y '())))
(cons a 'a)
(cons (y 5) (cons x y))
```

b) Skriv Racketkod som skapar dessa två strukturer:




Förtydligande: samtliga pilar pekar på hela paret (det vill säga, *inte enbart* car-delen eller cdr-delen).

## Uppgift 2 (3 poäng)

I denna uppgift ska du byta ut ett element på en given plats i en lista. Skapa proceduren `replace-at-rank` som har parametrarna `seq`, `index` and `value`. `seq` ska vara en lista med heltal, och `index` och `value` heltal. Proceduren ska returnera en kopia av listan `seq`, men där heltalet på plats `index` har bytts mot `value`.

List-index börjar på 0. Det vill säga, om listan `seq` har $n$ element så är 0 första elementet och $n - 1$ det sista.

Detta ska fungera:

```
> (replace-at-rank '(1 1 1 1 1 1) 0 42)
'(42 1 1 1 1 1)
> (replace-at-rank '(1 1 1 1 1 1) 3 42)
'(1 1 1 42 1 1)
```

Du kan anta att `seq` och `value` är heltal (ingen kontroll behöver göras), och att det finns något på plats `index` (det vill säga, $0 \leq$ `index` $\leq n - 1$).

## Uppgift 3 (3 poäng)

**OBS! Läs alla deluppgifter.** Det kan hända att du kan lösa uppgift b, även om du inte har en egendefinierad lösning på uppgift a.

Vi säger att ett tal är *perfekt* om summan av dess delare (undantaget talet självt) är talet. Till exempel är talet 6 perfekt. Dess delare (undantaget talet 6) är $1, 2, 3$, och $1 + 2 + 3 = 6$. Talet 14 är inte perfekt, för $1 + 2 + 7 = 10 \neq 14$.

Nästa perfekta tal efter 6 är 28 (verifiera gärna).

a) Skriv ett predikat `perfect?` som testar om ett tal är perfekt eller ej. Se till att bryta ned problemet i delproblem, och skapa hjälpprocedurer.

b) Använd några av procedurerna `filter`, `accumulate`, `map`, och `enumerate` för att skapa en lista av alla perfekta tal mellan 10 och 4242 (inklusive gränserna). Du behöver såklart inte använda alla procedurerna, om det inte behövs för att lösa uppgiften.

   Hur procedurerna fungerar, och hur de anropas, har diskuterats vid föreläsning 5. Kod för dessa (som ni *inte* behöver bifoga i ert svar!) finns på nästa sida.

```
(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)
  (if (null? L)
      null-value
      (proc (car L)
            (accumulate proc null-value (cdr L)))))

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

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

## Uppgift 4 (3 poäng)

**OBS! Läs alla deluppgifter.** Det kan hända att du kan lösa uppgift b-c, även om du inte har en egendefinierad lösning på uppgift a.

**Uppgift** Vi skapar en datatyp för bilar, för att i större program slippa fundera närmare på hur data är representerat. De relevanta egenskaper som en bil kan ha i vår tillämpning är dess färg, maximala hastighet, och antal platser (`colour`, `size` och `speed`).

a) Bestäm dig för hur en bil ska representeras, och implementera därefter

   1) en konstuktor (`make-car colour size speed`) som returnerar en bil

   2) selektorer för de olika egenskaperna; `get-colour`, `get-size`, `get-speed`

   Exempel:

```
(define ferrari (make-car 'red 2 320))
(define astonmartin (make-car 'blue 2 225))
(define trabi (make-car 'gray 4 165))

> (get-colour astonmartin)
'blue
> (get-speed ferrari)
320
> (get-speed trabi)
165
```

b) Implementera jämförelseoperatorerna `bigger?` och `faster?`. Dessa jämför två bilar utifrån `size` respektive `speed`. Koden i dessa jämförelseoperatorer får inte vara beroende av representationen som du valt i 4a. Koden i 4b ska alltså fungera även om man helt byter representation och ändrar koden i 4a.

   Exempel:

```
> (faster? ferrari trabi)
#t
> (bigger? ferrari trabi)
#f
```

c) En `ferrari` är snabbare än en `trabi`. Men det är inte uppenbart att en `ferrari` därför skulle vara bättre. Definiera en bils `goodness` som produkten av dess `size` och dess `speed`.

   Implementera jämförelseoperatorn `better?` som jämför två bilar utifrån denna nya egenskap. Vilken av exempelbilarna ovan är bäst - `ferrari` eller `trabi`? Varför?