Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2015-03-25
Sal (2)	TER1 TERE
Tid	14-16
Kurskod	TDDC74
Provkod	TEN1
Kursnamn/benämning **Provnamn/benämning**	Programmering - abstraktion och modellering Skriftlig tentamen/duggor
Institution	IDA
Antal uppgifter som ingår i tentamen	4
Jour/Kursansvarig Ange vem som besöker salen	Anders Märak Leffler
Telefon under skrivtiden	0731011291
Besöker salen ca klockan	ca kl. 14:30
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Antal exemplar i påsen	

Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2015-03-25
Sal (2)	TER1 TERE
Tid	14-16
Kurskod	TDDC74
Provkod	TEN1
Kursnamn/benämning **Provnamn/benämning**	Programmering - abstraktion och modellering Skriftlig tentamen/duggor
Institution	IDA
Antal uppgifter som ingår i tentamen	4
Jour/Kursansvarig Ange vem som besöker salen	Anders Märak Leffler
Telefon under skrivtiden	0731011291
Besöker salen ca klockan	ca kl. 14:30
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Antal exemplar i påsen	

TDDC74 Programmering: Abstraktion och modellering
Dugga 3, kl 14—16, 25 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. Väl valda namn omfattar exempelvis att inte blanda språk.

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 sista `define` du öppnade.

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! Denna dugga får du använda alla primitiver, inklusive kommandon som `set!`, `set-mcar!`, `set-mcdr!` för att ändra bindningar och ändra i strukturer.

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)

Betrakta koden nedan:

```
(define (f x)
  (define (g x)
    (if (< x 2)
        1
        (+ (g (- x 1)) 1)))
  (g x))

(f 1)
(f 2)
```

Rita omgivningsdiagram som visar det som händer när man evaluerar koden ovan. Markera globala omgivningen (GE) tydligt, och numrera ramarna i den ordning de skapas (E1, E2,...).

Du behöver inte skriva ut all kod som finns i procedurkroppar (body), men det ska tydligt framgå vilken kod du hänvisar till.

Uppgift 2 (3 poäng)

Vi vill representera ett bankkonto, där varje konto-objekt håller koll på just sitt saldo. Man kan antingen sätta in pengar på kontot (`add`) eller sätta saldot till något särskilt (`set`).

Skapa konstruktorn `make-account` som tar ett inledande saldo som argument, och ger tillbaka ett bankkonto-objekt (tekniskt sett ett procedurobjekt) som fungerar enligt nedan:

```
> (define account1 (make-account 100))
> (define account2 (make-account 220))
> (account1 'add 0)
100
> (account1 'set 88)
88
> (account1 'add 100)
188
> (account2 'add 0) ;; oberoende av account1
220
> (account2 'add -200)
20
> (+ (account2 'set 838469716578797182657000) 73)
838469716578797182657073
> (account1 'add 0)  ;; account1 som tidigare
188
```

Notera att varje add/set alltid returnerar det nya saldot.

Uppgift 3 (3 poäng)

OBS! Läs hela uppgiften. Du kan lösa del 3b utan att ha löst 3a.

a) **(2 poäng)** En muterbar lista kan innehålla såväl element utanför listan, som hänvisningar till listan själv. Implementera `count-self-ref` som tar en muterbar lista, och ger antalet sådana självreferenser. Du kan anta att listan slutar med den tomma listan (vi har alltså ingen ringstruktur).

Så här ska funktionen fungera:

```
> (define L1 (mlist 1 2 3 4))
> (count-self-ref L1)
0
> (define L2 (mlist 1 2 3 4))
> (set-mcar! (mcdr L2) L2)
> (count-self-ref L2)
1
> (define L3 (mlist 1 2 3 4))
> (set-mcar! L3 L3)
> (set-mcar! (mcdr (mcdr L3)) L3)
> (set-mcar! (mcdr (mcdr (mcdr L3))) L3)
> (count-self-ref L3)
3
```

OBS!

- För att avgöra om två variabler hänvisa till samma objekt ska du använda jämförelse-operatorn `eq?`

b) **(1 poäng)** Rita ett box-pointer-diagram som beskriver `L3` från uppgiften ovan. Rita så att man tydligt kan följa hur strukturen skapas/uppdateras (visa inte bara slutresultatet).

Uppgift 4 (3 poäng)

Vi definierar ett slags databasobjekt som för enkelhets skull enbart innehåller heltal.

Inuti objekten (som beskrivs nedan) representeras allt som en **vanlig lista** innehållande heltal. Databasobjekten ska ha följande metoder:

1. `get-content`: returnerar hela databasens innehåll (hela listan)
2. `insert!`: sätter in ett givet heltal i databas
3. `present?`: kolla om ett givet heltal finns i databas
4. `delete!`: tar bort alla förekomster av ett givet heltal i databas

Ett exempel på objekten i användning följer på nästa sida.

Koden nedan visar konstruktorn för våra databasobjekt när vi har redan implementerat kommandona `get-content` och `insert!`:

```
(define database%
  (class object%

    (field [content '()])

    (define/public (get-content) content)

    (define/public (insert! val)
      (set! content (cons val content)))

    (define/public (present? val)
      ... din kod ...)

    (define/public (delete! val)
      ... din kod ...)

    (super-new)))
```

Uppgift Implementera `present?` och `delete!`.

Här ser du hur våra databasobjekt ska fungera:

```
(define db
  (new database%))

> (send db get-content)
'()
> (send db insert! 7)
> (send db insert! 6)
> (send db insert! 5)
> (send db insert! 4)
> (send db get-content)
'(4 5 6 7)
> (send db present? 6)
#t
> (send db present? 9)
#f
> (send db insert! 9)
> (send db present? 9)
#t
> (send db insert! 7)
> (send db insert! 11)
> (send db insert! 2)
> (send db insert! 7)
> (send db get-content)
'(7 2 11 7 9 4 5 6 7)
> (send db delete! 2)
> (send db get-content)
'(7 11 7 9 4 5 6 7)
> (send db delete! 7)
> (send db get-content)
'(11 9 4 5 6)
> (send db delete! 1) ;; element som ej finns i db
> (send db get-content)
'(11 9 4 5 6)
```

OBS!

- Självklart får du använda `set!` i denna uppgift, men du får inte använda muterara listor eller par
- Glöm inte att det kan vara nyttigt att definiera hjälpprocedurer eller använda högre ordningens procedurer.
- För att jämföra heltal får du använda den vanliga jämförelseoperatorn, det vill säga =