Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2013-02-28
Sal (2) Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och ringa in vilken sal som avses	TER1 (TER2)
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
Telefon under skrivtiden	ankn. 19 63 eller 070-607 19 63
Besöker salen ca kl.	ca. kl. 09
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 2, Tid: kl 08-10, Datum: 2013-02-28

Läs alla frågorna först och bestäm dig för den ordning som passar dig bäst. Skriv tydligt för att öka läsbarheten. Använd **väl valda namn** på parametrar och **indentera** din kod.

Även om det i uppgiften står att du skall skriva en procedur/funktion, så får du gärna skriva ytterligare hjälpfunktioner som kan vara nödvändiga.

Duggan är baserad på kapitel 1-2 I kursboken varför man inte får använda procedurer som `SET!`, `SET-MCAR!` och `SET-MCDR!`. Ingen av uppgifterna behöver sådana procedurer som ändrar i variabler.

Betygsgradering: Det finns tre duggor i kursen. Varje dugga ger 12p, dvs totalt 36p för alla tre. För att passera en dugga krävs minst 3p på duggan. Totalt skall du på de tre duggorna för betyget 3 ha minst 18p, för betyget 4 minst 23p och för betyget 5 minst 27p.

Lycka till!

Uppgift 1

Antag att z är definierad enligt nedan:

```
(define z (cons 12 33))
```

Rita följande strukturer grafisk (som cons-lådor och pekare):

```
a)   (define a (list 5 2 3))
b)   (define b (cons 5 (cons 2 3)))
c)   (define c (cons (cons 6 7) 3))
d)   (define d (list (list 8 8) 34))
e)   (define e (cons z z))
f)   (define f (list z (cons 12 33) z))
```

Uppgift 2

Definiera proceduren **pick** som tar två argument, ett positivt heltal **n** och en lista och returnerar det **n**:te elementet i listan.

Under rekursionen kanske man upptäcker att elementet inte finns pga att listan är för kort, i så fall skall man skriva ut ett meddelande enligt nedan. Det är onödigt att beräkna listans längd för att se huruvida det finns rätt antal element eller inte, detta upptäcker vi medan vi går genom listan stegvis.

```
> (pick '(1 2 3 4 5) 4)
4
> (pick '(1 (2 3) 4 5) 2)
(2 3)
> (pick '(a b c) 3)
c
> (pick '(1 b 3 x 2) 6)
Index out of range.
```

Använd sedan substitutionsmodellen för att visa hur uttrycket **(pick '(w x y z) 3)** beräknas.

Uppgift 3

Skriv proceduren **uniques** med ett argument **lista** innehållande symboler och/eller heltal. Proceduren skall returnera en ny lista som innehåller samma element som argumentet men utan dubbletter. Ordningen på elementen i resultatet är inte viktigt.

Se exemplen nedan.

```
> (uniques '(1 2 4 7 5 45 43 22 4 2 45))
(1 7 5 43 22 4 2 45)

> (uniques '(11 33 scm racket scm 11 22 scm))
(33 racket 11 22 scm)
```

Uppgift 4

Vi skapar en datamängd över flygresor enligt följande:

```
(define flights
  (list
   ;;       airline orig dest flt# dprt arrv
   ;;
   (make-flight 'AS 'YVR 'SEA 2039 0930 1020)
   (make-flight 'AS 'SEA 'YVR 2126 1220 1310)
   (make-flight 'AS 'SEA 'ANC   85 1150 1430)
   (make-flight 'AS 'ANC 'FAI   93 1530 1620)
   (make-flight 'NW 'SEA 'MSO 2199 1700 2025)
   (make-flight 'UA 'YVR 'SEA 1483 0605 0650)
   (make-flight 'UA 'ANC 'FAI 1744 2220 2314)
   (make-flight 'UA 'SEA 'ANC 1760 1410 1636)
   (make-flight 'UA 'SEA 'SFO 1203 0630 0835)
   (make-flight 'UA 'SEA 'SFO 1764 1130 1334)
   (make-flight 'UA 'SEA 'OAK 1143 1228 1444)
   (make-flight 'UA 'SEA 'ORD  154 0700 1248)
   (make-flight 'UA 'SFO 'SEA 1706 1030 1234)
   (make-flight 'UA 'SFO 'RDD 7005 1420 1535)
   (make-flight 'UA 'SFO 'EWR   12 0800 1622)
   (make-flight 'UA 'RDD 'SFO 7014 0645 0800)
   (make-flight 'UA 'EWR 'LHR  906 1855 0645)
   (make-flight 'UA 'ORD 'EWR  362 1330 1648)))
```

make-flight tar följande information och skapar en flygresa: förkortningen för flygbolaget som äger planet, förkortningen för den flygplats där resan börjar, förkortningen för den flygplats där resan slutar, flygresans nummer, avgångstid, ankomsttid. Konstruktorn definierar vi så här:

```
(define make-flight
  (lambda (carrier origin dest flight-no depart arrive)
    (list carrier origin dest flight-no depart arrive)))
```

I exemplen nedan antar vi att **my-flight** är definierad som:

```
(define my-flight
        (make-flight 'UA 'EWR 'LHR  906 1855 0645))
```

a) Definiera följande selektorer:

(flight-carrier my-flight) returnerar **UA**
(flight-origin my-flight) returnerar **EWR**
(flight-destination my-flight) returnerar **LHR**
(flight-no my-flight) returnerar **906**
(flight-depart my-flight) returnerar **1855**
(flight-arrive my-flight) returnerar **645**

b) Definiera en procedur **find-flights** med två argument som går genom flygresor och plockar ut de resor där ett visst flygbolag flyger till en viss flygplats. Använd selektorer för att komma åt information i en flygresa. Exempel:

```
> (find-flights 'UA 'EWR)
((UA SFO EWR   12 0800 1622)
 (UA ORD EWR  362 1330 1648))
```

```
;;;Det är inte så relevant för uppgiften, men i
;;;vilket fall, här är förkortningarna som
;;;förekommer i datasamlingen ovan.
;;;
;;;  Flygbolagen
;;;
;;;    AS:  Alaska Airlines
;;;    NW:  Northwest Airlines
;;;    UA:  United Airlines
;;;
;;;  Flygplatserna
;;;
;;;    ANC: Anchorage, AK
;;;    EWR: Newark, NJ
;;;    FAI: Fairbanks, AK
;;;    LHR: London/Heathrow
;;;    OAK: Oakland, CA
;;;    ORD: Chicago, IL/O'Hare
;;;    MSO: Missoula, MT
;;;    RDD: Redding, CA
;;;    SEA: Seattle/Tacoma, WA
;;;    SFO: San Francisco, CA
;;;    YVR: Vancouver, BC
```