Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2012-08-15
Sal (1) Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och ringa in vilken sal som avses	TER2
Tid	8-12
Kurskod	TDDC74
Provkod	TEN1
Kursnamn/benämning **Provnamn/benämning**	Programmering - abstraktion och modellering Skriftlig tentamen/duggor
Institution	IDA
Antal uppgifter som ingår i tentamen	6
Jour/Kursansvarig Ange vem som besöker salen	Jalal Maleki
Telefon under skrivtiden	1963, eller 070-607 19 63
Besöker salen ca kl.	ca 09:00
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, 2362, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Vilken typ av papper ska användas, rutigt eller linjerat	valfritt
Antal exemplar i påsen	

AID-nummer:	Datum: 2012-08-15
Kurskod: TDDC74	Provkod: TEN1

TDDC74 Programmering: Abstraktion och modellering
Tentamen

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.

Tentan består av sex uppgifter.

Betygsgradering: För betyget 3 krävs minst 12p, för betyget 4 minst 16p och för betyget 5 minst 20p. Högst möjliga antal poäng är 24.

Lycka till!

AID-nummer:	Datum: 2012-08-15
Kurskod: TDDC74	Provkod: TEN1

Uppgift 1

(4 poäng) Den argumentlösa Schemeproceduren `flip` är definierad enligt nedan.

```
(define flip
  (let ((state 0))
    (lambda ()
      (set! state (- 1 state))
      state)))
```

Här är tre anrop till proceduren:

```
 > (flip)
??
 > (flip)
??
 > (flip)
??
```

Vad returneras efter varje anrop? Rita de omgivningsdiagram som fås efter de fyra uttrycken ovan har evaluerats (dvs definitionen på flip och de tre anropen).

Uppgift 2

(4 poäng) Om f är en numerisk funktion och n är ett positivt heltal, då skriver vi

$f(f(\ldots(f(x))\ldots))$

för att visa en funktion som gör samma sak som n-stycken repetitioner av f. Skriv funktionen `repeat` i Scheme som tar `f` och `n` som argument och returnerar en funktion som motsvarar funktionen $f(f(\ldots(f(x))\ldots))$ med `n`-repetitioner av `f`. Följande exempelanrop illustrerar ytterligare hur `repeat` skall fungera.

```
> (define (square x) (* x x))

> ((repeat square 2) 3)
81

> ((repeat square 3) 2)
256

> ((repeat square 1) 9)
81
```

AID-nummer:	Datum: 2012-08-15
Kurskod: TDDC74	Provkod: TEN1

Uppgift 3

(4 poäng) Vad blir värdet av följande uttryck om det evalueras i Scheme?

```
(let ((a 22))
  (let ((a +)
        (b a))
    (a 11 (a b 33))))
```

Skriv om uttrycket så att `let`-uttrycken översätts till motsvarande `lambda`-uttryck.

Uppgift 4

(4 poäng) Skriv Schemeproceduren `xproduct` som tar ett antal mängder (en eller flera) som argument och returnerar en ny mängd som utgör argument-mängdernas kryssprodukt. Denna produkt, som även kallas för Cartesiska produkten, väljer ut ett element åt gången ur en mängd och kombinerar med element ur den Cartesiska produkten av de andra mängderna. Här kommer några exempel som ytterligare förklarar hur `xproduct` skall fungera.

```
> (xproduct '(1 2) '(a b c))
((1 a) (1 b) (1 c) (2 a) (2 b) (2 c))

> (xproduct '(a b c))
((a) (b) (c))

> (xproduct '(1 2) '(a b c) (x y))
((1 a x)(1 a y)(1 b x)(1 b y)(1 c x)(1 c y)
 (2 a x)(2 a y)(2 b x)(2 b y)(2 c x)(2 c y))
```

Uppgift 5

(4 poäng) Följande Scheme-procedur kan användas för att skapa objekt som fungerar som enkla bankkonton.

```
(define (make-account balance)
  (define (withdraw amount)
    (if (>= balance amount)
        (begin (set! balance (- balance amount))
               balance)
        "Insufficient funds"))
  (define (dispatch m)
    (cond ((eq? m 'withdraw) withdraw)
          (else (error "Unknown request -- MAKE-ACCOUNT"
                       m))))
  dispatch)
```

Följande uttryck visar hur ett sådant konto kan användas:

```
> (define acc (make-account 100))

> ((acc 'withdraw) 50)
50

> ((acc 'withdraw) 60)
"Insufficient funds"

> ((acc 'withdraw) 30)
20
```

Komplettera make-account (där det är markerat med ??? nedan) så att man kan även sätta in pengar i kontot.

```
(define (make-account balance)
  (define (withdraw amount)
    (if (>= balance amount)
        (begin (set! balance (- balance amount))
               balance)
        "Insufficient funds"))
  (define (deposit amount)
   ???)
  (define (dispatch m)
    (cond ((eq? m 'withdraw) withdraw)
          ???
          (else (error "Unknown request -- MAKE-ACCOUNT"
                       m))))
  dispatch)
```

Då skulle man kunna skapa konton precis som tidigare och sätta in pengar enligt nedan:

```
> (define acc2 (make-account 100))

> (define acc3 (make-account 200))

> ((acc2 'withdraw) 50)
50

> ((acc2 'withdraw) 60)
"Insufficient funds"

> ((acc2 'deposit) 40)
90

> ((acc3 'deposit) 20)
220

> ((acc2 'withdraw) 60)
30
```

AID-nummer:	Datum: 2012-08-15
Kurskod: TDDC74	Provkod: TEN1

Uppgift 6

(4 poäng) Ändra `make-account` proceduren från förra uppgiften så att den skapar lösenordskyddade konton, dvs den skall ta ytterligare ett argument (en symbol) som krävs sedan vid användning enligt följande exempel:

```
> (define acc4 (make-account 100 'abracadabra))
```

Det resulterande kontot skall sedan kunna hantera uttag och insättningar endast om det lösenord som angivits vid kontots skapande anges.

```
> ((acc4 'abracadabra 'withdraw) 40)
60

> ((acc4 'simsalabim 'deposit) 50)
"Incorrect password"
```