# Försättsblad till skriftlig tentamen vid Linköpings Universitet

| | |
|---|---|
| **Datum för tentamen** | 2013-08-21 |
| **Sal (1)**<br>Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och ringa in vilken sal som avses | TER1 |
| **Tid** | 8-12 |
| **Kurskod** | TDDC74 |
| **Provkod** | TEN1 |
| **Kursnamn/benämning**<br>**Provnamn/benämning** | Programmering - abstraktion och modellering<br>Skriftlig tentamen/duggor |
| **Institution** | IDA |
| **Antal uppgifter som ingår i tentamen** | 6 |
| **Jour/Kursansvarig**<br>Ange vem som besöker salen | Jalal Maleki |
| **Telefon under skrivtiden** | ankn. 19 63 eller 070-607 19 63 |
| **Besöker salen ca kl.** | ja |
| **Kursadministratör/kontaktperson**<br>(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** | |

# Tentamen

## TDDC74 Programmering: Abstraktion och modellering
## Provkod TEN1

Läs alla frågorna först och bestäm dig för den ordning som passar dig bäst.

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

Skriv tydligt för att öka läsbarheten. Använd **väl valda namn** på parametrar och **indentera** din kod. Sammanlagt 2 poäng tilldelas väl-skriven kod, dvs, kod som innehåller bra namngivning, korrekt indentering, användning av dataabstraktion vid behov, och andra principer för bra Scheme-kod.

Det finns sex uppgifter i tentan. Poängen per uppgift/deluppgift anges i samband med varje uppgift.

**Betygsgradering:** För betyget 3 krävs minst 12p, för betyget 4 minst 16p och för betyget 5 minst 19p. Högst möjliga antal poäng är 24.

Lycka till!

# Uppgift 1 - Uttryck och värden

(4 poäng) Antag att x och y är definierade enligt nedan:

```
(define x 3)
(define y 5)
```

Ange värdet som define tilldelar till det namn som definieras. Om du vill kan du använda omgivningsdiagram för att grafiskt visa värdet på procedurer - detta är dock inte nödvändigt om du kan på annat sätt beskriva saken:

1. `(define k (cons x (cons x y)))`
2. `(define m (cons x y))`
3. `(define n (cons (car k) (car (cdr k))))`
4. `(define p (list m (+ x y)))`
5. `(define f (lambda (x y) (cons x (cdr y))))`
6. `(define q (f m n))`
7. `(define r (cons f p))`
8. `(define s (let ((p q) (q p)) (f p q)))`

# Uppgift 2 - Enkel iteration/rekursion

(4 poäng) Skriv en rekursiv funktion `(tabort-tal l)`, som tar bort alla förekomster av tal på alla nivåer i listan `l`. Exempel.

```
> (tabort-tal '(a b 3 (5 c (6)) (4 d w)))
(a b (c ()) (d w))

> (tabort-tal '(9 a 8 b 7))
(a b)
```

Redogör sedan för huruvida din funktion är en rekursiv eller iterativ processlösning? Motivera.

# Uppgift 3 - iteration/rekursion

(4 poäng) Vissa funktioner kan beräknas genom att summera en oändlig serie. Vi skall här skriva en funktion, som summerar en ändlig serie (McLaurin-serien), där vi anger hur *många* termer som skall summeras.

Funktionen *sin* kan beräknas med hjälp av följande approximation:

$$\sin x = \sum_{k=0}^{n} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

Skriv sedan en funktion `(sin n x)` som beräknar $sin\ x$ med hjälp av McLaurin-serien med $n$ termer. Du får gärna använda Schemefunkionen `(expt m n)` som beräknar $m^n$ och naturligtvis fakultetsfunktionen som kan antas existera.

# Uppgift 4 - listhantering

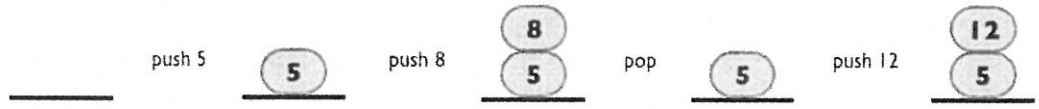
(4 poäng) Skriv en Schemefunktion **`(insert n l)`** med två parametrar där den första är ett heltal och den andra en ordnad lista på heltal. **`insert`** returnerar en ny ordnad lista där första argumentet finns på rätt ställe i listan.

```
> (insert 3 '())
(3)

> (insert 5 '(1 2 3))
(1 2 3 5)

> (insert 2 '(1 2 3))
(1 2 2 3)

> (insert -3 '(1 2 3))
(-3 1 2 3)
```

Använd substitutionsmodellen för att redovisa hur din implementation av insert beräknar `(insert 6 '(1 2 5 8))`.

# Uppgift 5 - omgivningsdiagram och objekt

(4 poäng) En stack (eller en hög) är en struktur som innehåller 0 eller flera element. Nya element läggs ovanför det senaste elementet med hjälp av operationen PUSH, och man kan enbart ta bort (POP) det senast inlagda elementet från stacken. En hög är tom från början. Följande figur visar en sekvens av PUSH och POP operationer.




Ett sätt att implementera stack i Scheme är som objekt med en DISPATCH-procedur som sköter meddelandehanteringen:

```
(define (make-stack)
  (let ((content ()))
    (define (push x)
      (set! content (cons x content)))
    (define (pop)
      (if (null? content)
          (error "Empty stack -- POP")
          (let ((top (car content)))
            (set! content (cdr content))
            top)))
    (define (initialize)
      (set! content '())
      'done)
    (define (empty?)
      (null? content))
    (define (dispatch m)
      (cond ((eq? m 'push) push)
            ((eq? m 'pop) (pop))
            ((eq? m 'initialize) (initialize))
            ((eq? m 'empty?) (empty?))
            (else (error "Unknown request -- STACK" m))))
    dispatch))

(define (pop stack)
  (stack 'pop))

(define (push stack x)
  ((stack 'push) x))
```

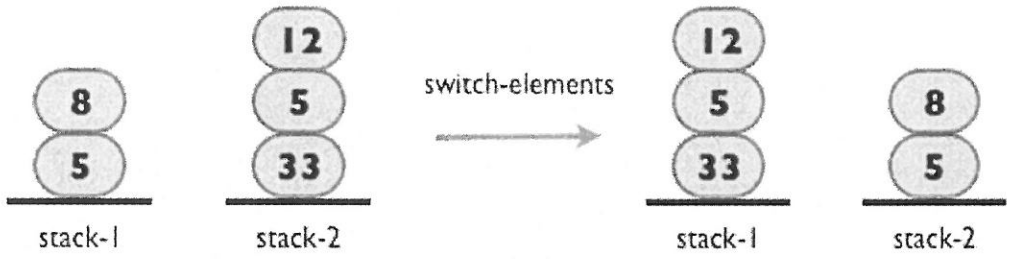
Och här följer hur den är tänkt att användas.

```
> (define s (make-stack))
> (define t (make-stack))
> (push s 5)
> (push s 8)
> (pop s)
8
```

Rita de omgivningsdiagram som avbildar hur ovanstående uttryck evalueras.

# Uppgift 6

(4 poäng) Skriv en procedur `(switch-elements stack-1 stack-2)` som byter innehållet på två stackar enligt förra uppgiften. Följande figur visar exempel innehållen före och efter ett anrop till `switch-elements`:




`stack-1` och `stack-2` antas vara definierade med hjälp av `make-stack` i förra uppgiften.