Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2013-10-23
Sal (1) Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och ringa in vilken sal som avses	TER1
Tid	14-18
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	ankn. 19 63 eller 070-607 19 63
Besöker salen ca kl.	ja
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	

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 - föränderliga liststrukturer

(2 poäng) Rita värdet på k, m, n och p nedan som `mcons`-lådor och pekare.

```
(define x 3)
(define y 6)
(define k (mcons x (mcons x y)))
(set! x 9)
(define m (mcons x y))
(define n (mcons (mcar k) (mcar (mcdr k))))
(set-car! m 12)
(define p (mcons m n))
```

Uppgift 2 - Uttryck och värden

(3 poäng) Skriv om följande `let`-uttryck genom ersätta `let`-uttrycken med motsvarande `lambda`-uttryck:

```
(let ((a 4)
      (b 9))
  (let ((disc (- (* b b) (* 4 (* a c)))))
    (/ (+ (- b) (sqrt disc))
       (* 2 a))))
```

Diskutera vad värdet på detta uttryck blir vid evaluering.

Uppgift 3 - iteration/rekursion

(3 poäng) Följande Scheme-procedur implementerar funktionen *fib*(*n*) och därmed beräknar det n:te Fibonaccitalet.

```
(define (fib n)
  (define (iter a b counter)
    (if (= counter 0)
        b
        (iter (+ a b) a (- counter 1))))
  (iter 1 0 n))
```

Skriv en **iterativ** procedur `sumfib` som tar ett heltal *n* som argument och beräknar följande summa:

fib(*n*)+*fib*(*n*-1)+ ... + *fib*(1) + *fib*(0)

Uppgift 4 - listhantering

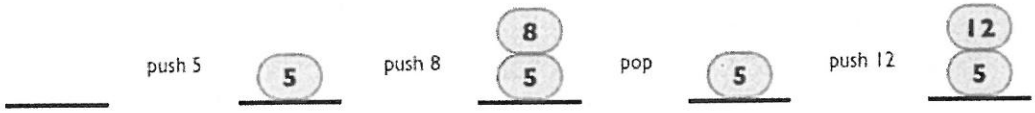
(6 poäng) Medianen är det tal i en mängd som storleksmässigt ligger så att det finns lika många tal som är större än och mindre än medianen. Av talen 1, 7, 9, 10 och 17 är 9 medianen (medan medelvärdet är 8,8). För mängder med ett jämnt antal tal definieras medianen som medelvärdet av de två tal som ligger i mitten.

Skriv proceduren (median lista) som tar en osorterad lista och returnerar medianen av de tal som ingår i listan. Redogör sedan huruvida din procedur skapar en interativ eller en rekursiv processlösning. För att sortera listan först kan du antingen använda dig av den sort-funktion som finns i Scheme eller skriva egen.

```
> (median '(1 7 9 10 34))
9
> (median '(3 5 9 20 49 10)) ; sorteras till 3 5 9 10 20 49
9.5
> (median '(1 6 9))
6
> (median '(2 4))
3
> (median '(2))
2
```

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))
```

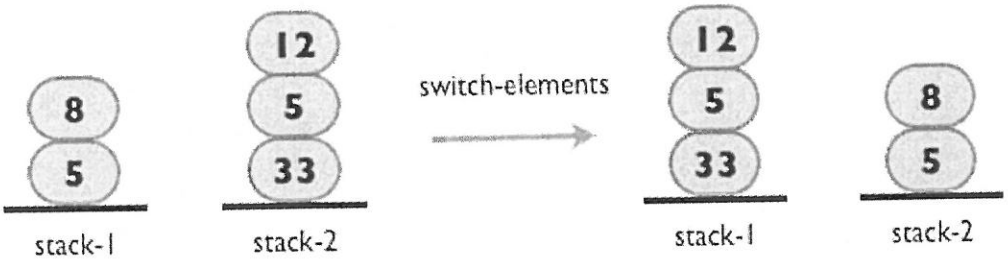
Och här följer hur den är tänkt att användas.

```
> (define s (make-stack))
> (define t (make-stack))
> (push s 23)
> (pop s)
23
> (push t 45)
```

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, så här:

```
> (define stack-1 (make-stack))
> (define stack-2 (make-stack))
> (push stack-1 5)
> (push stack-1 8)
> (push stack-2 33)
> (push stack-2 5)
> (push stack-2 12)
```