Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2015-08-29
Sal (3)	TER1 TER2 TERE
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	5
Jour/Kursansvarig Ange vem som besöker salen	Johannes Schmidt
Telefon under skrivtiden	013-281398 eller. 0725 - 721803
Besöker salen ca klockan	ja kl. 09:00 & 11:45
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, anna.grabska.eklund@liu.se, ankn. 2362
Tillåtna hjälpmedel	Inga
Övrigt	
Antal exemplar i påsen	

Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2015-08-29
Sal (3)	TER1 TER2 TERE
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	5
Jour/Kursansvarig Ange vem som besöker salen	Johannes Schmidt
Telefon under skrivtiden	013-281398 eller. 0725-721803
Besöker salen ca klockan	ja 09:00 & 11:45
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, anna.grabska.eklund@liu.se, ankn. 2362
Tillåtna hjälpmedel	Inga
Övrigt	
Antal exemplar i påsen	

Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2015-08-29
Sal (3)	TER1 TER2 TERE
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	5
Jour/Kursansvarig Ange vem som besöker salen	Johannes Schmidt
Telefon under skrivtiden	013-281398 eller. 0725 -721803
Besöker salen ca klockan	ja kl. 09:00 & 11:45
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund, anna.grabska.eklund@liu.se, ankn. 2362
Tillåtna hjälpmedel	Inga
Övrigt	
Antal exemplar i påsen	

TDDC74 Programmering: Abstraktion och modellering
Tentamen, lördag 29 augusti 2015, kl 8–12

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 skriva en procedur/funktion, så får du skriva ytterligare hjälpfunktioner som kan vara nödvändiga.

Observera att poängavdrag kan ges för onödigt komplicerade eller ineffektiva lösningar.

Betygsgränser:

12 – 15	betyg 3
15,5 – 18,5	betyg 4
19 – 24	betyg 5

Lycka till!

Uppgift 1. Beräkning av uttryck (4 poäng)

Vi ger DrRacket följande uttryck att beräkna (i ordningen som anges).

Vilka värden returneras? Om det blir fel, beskriv varför (vilken typ av fel).

```
> (define x 55)
> x
> (define y 64)
> y
> (define z (set! y (list 42)))
> z
> (define inc (lambda (n) (+ n 1)))
> inc
> (define a (lambda (a b c) (list x y z)))
> a
> (define b (a x y z))
> b
> (define c (inc x))
> c
> (define d (inc y))
> d
> (define e (inc z))
> e
```

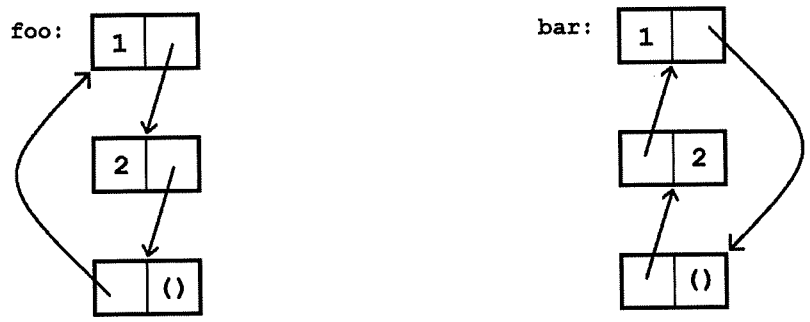
Uppgift 2. Box-pointer diagram (4 poäng)

a) Antag att vi har evaluerat följande Racket-uttryck (i ordning).

```
(define x
  (let ((tmp (cons 2 3)))
    (cons 1 (cons tmp tmp))))
(define y (cons 4 5))
(define z (cons (car (car (cdr x))) y))
(define u (cons (cdr (cdr x)) (cdr x)))
```

Rita box-pointer diagram för strukturerna `x`, `y`, `z`, och `u`. Var noga med pekarna!

b) Skriv Schemekod som skapar dessa två strukturer. Du får bara använda muterbara par när/om det krävs. Du får använda `define` och `let` var du vill.



Uppgift 3. Omgivningsdiagram (4 poäng)

Betrakta koden nedan:

```
(define (make-account balance)
  (lambda (cmd value)
    (cond
      [(eq? cmd 'set)
       (set! balance value)
       balance]
      [(eq? cmd 'add)
       (set! balance (+ balance value))
       balance]
      [else
       (printf "Unknown command\n")])))

(define account1 (make-account 100))
(define account2 (make-account 220))

(account1 'add 0)
(account1 'set 60)
(account2 'add 10)
```

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 skall tydligt framgå vilken kod du hänvisar till.

Uppgift 4. Rekursiva vs iterativa processer (5 poäng)

Vi betraktar funktionen f som tar in ett heltal och returnerar det n:te fibonaccitalet:

$$f(n) = \begin{cases} n \text{ om } 0 \leq n < 2 \\ f(n-1) + f(n-2) \text{ om } n \geq 2 \end{cases}$$

a) **(3 poäng)** Ange två implementationer av f i Racket, en rekursiv processlösning `frek` och en iterativ processlösning `fiter`.

b) **(2 poäng)** Använd subtitutionsmodellen för att beräkna `(frek 3)` och `(fiter 3)`.

Uppgift 5. Data-abstraktion (7 poäng)

Den här uppgiften har tre delar, men dessa kan lösas oberoende av varandra.

Ett polynom (engelska *polynomial*) kan representeras som en följd av koefficienter.

I Racket kan det göras som en lista av koefficienterna.

Till exempel kan vi låta listan `'(5 6 7 8)` stå för polyomet $8x^3 + 7x^2 + 6x + 5$ och listan `'(3 0 1)` för polynomet $x^2 + 3$. Det sista ser vi kanske tydligare om vi skriver $x^2 + 3 = 1x^2 + 0x^1 + 3x^0$.

Polynomets *grad* (engelska *degree*) är den högsta förekommande exponenten. Graden av $8x^3 + 7x^2 + 6x + 5$ är därmed 3 och graden av $x^2 + 3$ är 2.

Vi vill implementera en datatyp `Polynomial` som representerar ett polynom. Datatypen skall ha följande funktionalitet:

- beräkna graden (degree)
- läsa en specifik koefficient
- evaluera polynomet i en given punkt

Här är kod skelettet för klassen `Polynomial%`.

```
(define Polynomial%
  (class object%

    (init-field coefficients)

    (define/public (get-degree)
      ...)

    (define/public (get-coefficient index)
      ...)

    (define/public (evaluate-at value)
      ...)

    (super-new)))
```

Den interna representationen av polynomet skall vara en vanlig lista av koefficienterna `coefficients`.

Så här skall det fungera:

```
> (define pol (make-object Polynomial% '(3 0 1))) ; x^2 + 3
> (send pol get-degree)
2
> (send pol get-coefficient 0)
3
> (send pol get-coefficient 1)
0
> (send pol get-coefficient 2)
1
> (send pol get-coefficient 3)
0
> (send pol get-coefficient 65535)
0

; get highest-index non-zero coefficient
> (send pol get-coefficient (send pol get-degree))
> 1

> (send pol evaluate-at 3)
12
> (send pol evaluate-at 4)
19
```

Uppgifter

a) **(3 poäng)** Implementera `get-degree` och `get-coefficient` så att de fungerar som förväntat. Du får använda `car` och `cdr` för att hantera listan av koefficienterna.
OBS! Observera att den första koefficienten har index 0 och dessutom att `get-coefficient` skall returnera 0 när man anger ett index som är högre än polynomets grad.

b) **(2 poäng)** Betrakta nedanstående implementation av `evaluate-at`.

```
(define/public (evaluate-at value)
  (define (helper coef e)
    (if (null? coef)
        0
        (+ (* (car coef) (expt value e))
           (helper (cdr coef) (+ e 1)))))
  (helper coefficients 0))
```

Som du kan se beror koden på representationen av polynomet som lista. Förbättra koden och skriv den om så att den är oberoende av polynomets representation (d.v.s. använd abstrakioner från a) i stället för `car` och `cdr`).

c) **(2 poäng)** Vi utökar lite funktionalitetet av vår datatyp. Metoden `get-as-function` returnerar en vanlig Racket-funktion som representerar polynomet.

```
(define/public (get-as-function)
  (lambda (value)
    (evaluate-at value)))
```

Hur uppför sig denna metod? Vi kallar `get-as-function` och får polynomet som en vanlig funktion, säg f. Om vi sedan modifierar polynomet (vilket händer om vi t.ex. adderar polynom, eller ändrar en koefficient), så ändras också beteendet av vår funktion f. Det vill säga att funktionen f inte fungerar som en kopia av polynomet utan som en referens till polynomet. Utöka koden så att den returnerade funktionen fungerar som en kopia.

Tips Vad är mer praktiskt? Att ta en kopia av listan av koefficienterna eller av hela polynom-objektet?