Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2014-10-22
Sal (2)	TER1 TERD
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	5
Jour/Kursansvarig Ange vem som besöker salen	Johan Billman
Telefon under skrivtiden	0730539092
Besöker salen ca klockan	ca kl. 16
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund,ankn. 2362, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Antal exemplar i påsen	

Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2014-10-22
Sal (2)	TER1 TERD
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	5
Jour/Kursansvarig Ange vem som besöker salen	Johan Billman
Telefon under skrivtiden	0730539092
Besöker salen ca klockan	ca kl. 16
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund,ankn. 2362, anna.grabska.eklund@liu.se
Tillåtna hjälpmedel	inga
Övrigt	
Antal exemplar i påsen	

TDDC74 Programmering: Abstraktion och modellering
Tentamen, onsdag 22 oktober 2014, kl 14–18

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.

Även om det i uppgiften står att du skall skiva 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 – 15,5	betyg 3
16 – 19,5	betyg 4
20 – 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 a (cons b 'a))
> a
> (define b (cons 'b a))
> b
> (define c (cons 'a 'b))
> c
> (define d (cons c (cons 2 (cons 3 ()))))
> d
> (define f (lambda (a b) (car (cdr d))))
> f
> (define g (lambda (x y z) (+ 55 (f (f x y) (f y z)) z)))
> g
> (define x (g 1 2 3))
> x
> (define y (g (g 1 2 3) 1 (f 1 2)))
> y
```

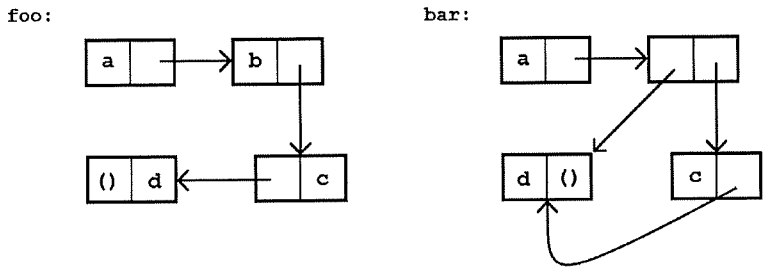
Uppgift 2. Box-pointer diagram (4 poäng)

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

```
(define a (lambda (a b) b))
(define b 4)
(define x (cons (cons a b) (cons 'a 'b)))
(define y (cons (a a 'a) a))
(define z (list a b (cons 'a 'b)))
```

Rita box-pointer diagram för strukturerna `a`, `b`, `x`, `y` och `z`. 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. Evalueringsmodeller (6 poäng)

Betrakta koden nedan:

```
(define (h z)
  (+ z 30))

(define (f x)
  (define (g y)
    (+ y 20))
  (+ (g (+ x 1)) (h (+ x 2))))

(f 1)
```

Uppgift 3a (2 poäng)

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 i procedurkropparna (body), men det ska tydligt framgå vilken kod du hänvisar till.

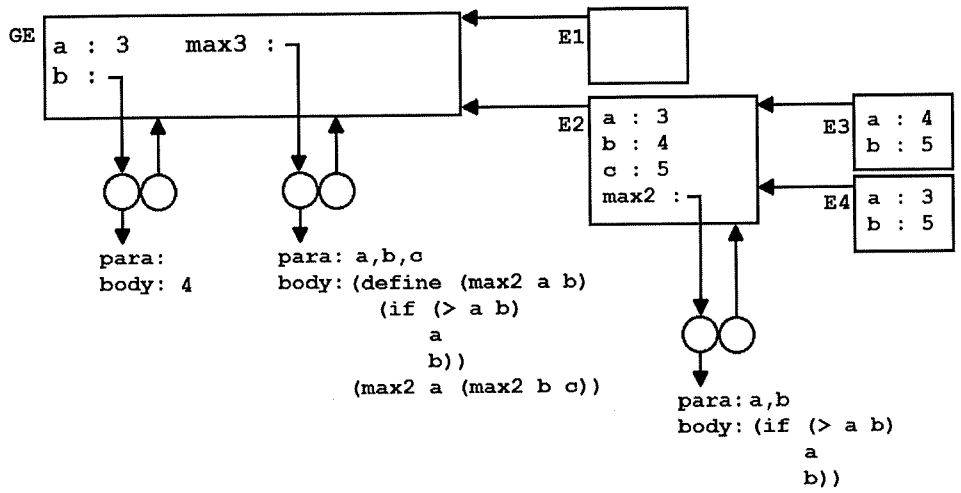
Tips Det kan underlätta att skriva om koden utan syntaktiskt socker (exempelvis för `define`- och `let`-uttryck). Det enda du behöver redovisa är dock omgivningsdiagram.

Uppgift 3b (2 poäng) I det här fallet kan vi även använda substitutionsmodellen, hur kommer det sig?

Uppgift 3c (2 poäng) Använd substitutionsmodellen för att beräkna `(f 1)`.

Uppgift 4. Omgivningsdiagram (4 poäng)

Givet följande omgivningsdiagram:



Skriv kod som ger upphov till detta omgivningsdiagram. Det ovan är hela omgivningsdiagrammet, och inget har "städats bort". Skriv bara precis den kod som behövs.

Uppgift 5. Abstrakt datatyp (6 poäng)

Vi vill implementera ett binärträd (som i föreläsning 6). Binärträd består av noder, och vi behöver ett sätt att representera en nod. En nod består av tre delar: dess vänstra barn (subträd), nodens värde/nyckel, och dess högra barn.

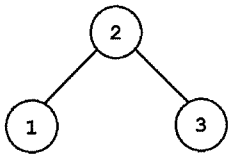
OBS! Läs igenom och försök besvara samtliga deluppgifter, även om du inte har ett fullständigt svar på 5a.

Uppgift 5a - Representation (2 poäng)

Du får själv välja hur den interna representationen av trädet ser ut. Följande ska dock implementeras:

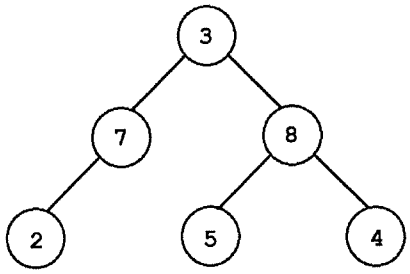
- Konstruktorn `(mkn left value right)`, som skapar en nod.
- Följande "getters": `(get-left node)`, `(get-value node)`, `(get-right node)`.
- `emt`, vilken representerar det tomma trädet. Även här får du välja hur du representerar det internt.
- Predikatet `(empty-tree? node)`, vilket testar hurvida en given nod är det tomma trädet.

Här är ett exempel på hur det ska fungera: `(mkn (mkn emt 1 emt) 2 (mkn emt 3 emt))` kommer att ge följande binära träd:



Uppgift 5b - Konstruktion av träd och definitioner (2 poäng)

Beakta följande binära träd:



Definiera `mytree`, som motsvarar trädet ovan. Använd `mkn` och `emt` som i uppgift 5a.

Är det här ett binärt *sök*träd eller ej? Varför/varför inte?

Uppgift 5c - Procedurer användande gränssnittet (2 poäng)

Implementera predikatet `(present? value tree)`, som returnerar `#t` om värdet finns i trädet och annars returnerar `#f`.

Så här ska det fungera (`mytree` från 5b):

```
> (present? 1 mytree)
> #f
> (present? 2 mytree)
> #t
> (present? 3 mytree)
> #t
> (present? 6 mytree)
> #f
```

Din lösning ska vara oberoende av hur noder representeras. Använd alltså procedurerna som diskuteras i 5a.