Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2017-02-22
Sal (1)	SU-salar, IDA(80)
Tid	14-16
Kurskod	TDDC74
Provkod	KTR1
Kursnamn/benämning **Provnamn/benämning**	Programmering - abstraktion och modellering Frivillig dugga
Institution	IDA
Antal uppgifter som ingår i tentamen	5
Jour/Kursansvarig Ange vem som besöker salen	Jalal Maleki
Telefon under skrivtiden	ankn. 1963
Besöker salen ca klockan	
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Anna Grabska Eklund ankn. 2362 Annelie Almquist , annelie.almquist@liu.se ankn 2934
Tillåtna hjälpmedel	inga
Övrigt	
Antal exemplar i påsen	

TDDC74 Programmering: Abstraktion och modellering
Dugga 1, 2017-02-22 kl 14-16

Läs alla frågorna först och bestäm dig för i vilken ordning du vill lösa uppgifterna. Uppgifterna är inte nödvändigtvis i svårighetsordning.

Använd **väl valda namn** på parametrar och **indentera** din kod. Väl valda namn inkluderar bland annat konsekvent språk. Du behöver inte skriva kodkommentarer, annat än för väldigt svårförklarade hjälpfunktioner (som bör undvikas). Namngivning ska vara tillräcklig (och följa konventioner). Skriv inte onödigt komplicerade lösningar.

Om det är naturligt, definiera gärna hjälpfunktioner! Hjälpfunktioner ska som vanligt lösa tydliga och lättförklarade uppgifter.

Du får använda **alla tillgängliga primitiver** och språkkkonstruktioner i Racket, **om annat inte anges** i uppgiftstexten.

Frågor

Är något otydligt i uppgifterna kan du använda meddelande-funktionen i tentaklienten för att skicka frågor till rättande lärare.

Att lämna in

Skicka in uppgifterna med hjälp av tentaklienten, när du är klar med dem! Du får svar via klienten när din uppgift är rättad. Vänta *inte* på att alla uppgifter är klara med att lämna in. När du har lämnat in en uppgift, fortsätt arbeta på nästa. Du har en inlämning per uppgift (så skicka inte ev a- och b-uppgifter separat). **Följ angivna namn, och testa att alla körexempel i uppgiften fungerar exakt som de är inskrivna!**

Din totalpoäng på duggan ges som summan av lösningar du lämnat in i tid, oavsett om de hinner rättas klart inom skrivtiden eller ej.

Betyg

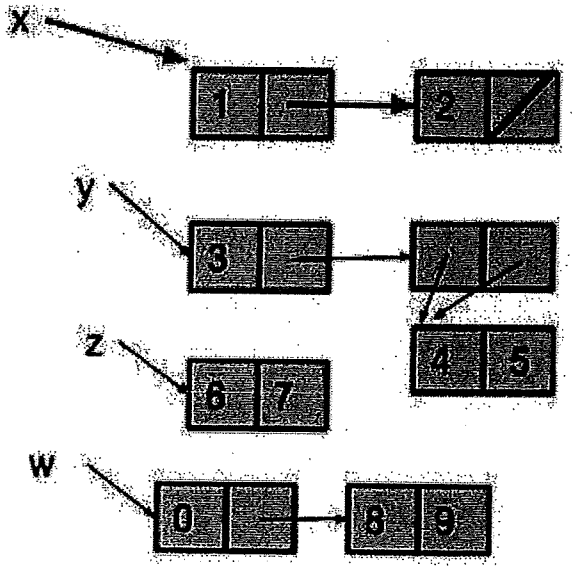
Det finns två duggor i kursen. På varje dugga kan man få som mest 15p. Totalpoängen avgör slutbetyget i kursen. För trea räcker 50% av det totala poängen.

Ett resultat man inte är nöjd med, kan plussas vid tentamenstillfället i juni.

Lycka till!

Uppgift 1, Länkade strukturer (2p)

Skriv kod som ger upphov till följande strukturer och bindningar. Skriv dina lösningar i filen **uppg1.rkt**.



Uppgift 2, Teori och semantik (3p)

I dessa uppgifter svarar ni genom enkel textinlämning. Skriv era svar som kodkommentarer i **uppg2.rkt** och lämna in så. Lämna in uppgift a, b och c (eller de du svarar på) samtidigt, i samma fil.

a) Vi skriver in koden nedan, i exakt den ordning som den står. Det fungerar inte. Varför inte? Svara kortfattat (max en-två meningar).

```
(define n (+ n 1))
```

b) Vi skriver in koden nedan i Racket. Varför fungerar den inte? Svara kortfattat (max en-två meningar).

```
(let ((x 0)
      (y x))
  (+ x y))
```

c) Ger följande procedur upphov till en iterativ eller rekursiv process? Varför? Svara kortfattat (max en-två meningar).

```
(define (f1 n)
  (if (= n 0)
      112
      (f1 (- n 1))))
(f1 100)
```

Uppgift 3, Enkelrekursion (4p)

Skriv dina svar i filen **uppg3.rkt**. Det är viktigt att alla funktioner som anges nedan har exakt samma namn som i uppgiften. Inkludera uttrycket `(provide divisors prime?)` i din fil.

a) Definiera en funktion `divisors` som tar ett heltal $n \geq 0$ och returnerar en lista av talets delare.

Detta är exempel på korrekt utdata:

```
> (divisors 16)
'(1 2 4 8 16)
> (divisors 17)
'(1 17)
```

Det är helt OK att skapa en funktion som returnerar listan i omvänd ordning.

b) Definiera en funktion `prime?` som testar om ett heltal $n \geq 2$ är ett primtal (ett tal som enbart är delbart med sig självt och ett). Du får självklart använda tidigare definierade funktioner.

```
> (prime? 4)
#f
> (prime? 3)
#t
> (prime? 257)
#t
> (if (prime? 257) (printf "257 är ett primtal!~n") (printf "Nej~n"))
257 är ett primtal!
```

Uppgift 4, Dubbelrekursion (3p)

Skriv dina svar i filen **uppg4.rkt**. Det är viktigt att alla funktioner som anges nedan har exakt samma namn som i uppgiften. Inkludera raden `(provide reverse-all)` i din fil.

Skapa en funktion `reverse-all` som tar en lista, potentiellt innehållande underlistor (alla äkta listor, inte allmänna parstrukturer) och vänder den.

Begränsning: Du får inte använda Rackets reverse-funktion.

Så här ska det fungera:

```
> (reverse-all '(1 2 3 4))
'(4 3 2 1)
> (reverse-all '(1 2 (3 4) 5))
'(5 (4 3) 2 1)
> (reverse-all '(1 2 (a b (- /)) 5))
'(5 ((/ -) b a) 2 1)
> (reverse-all '(1 () 5))
'(5 () 1)
```

Uppgift 5, Högre ordningens funktioner (och rekursion) (3p)

Skriv dina svar i filen **uppg5.rkt**. Det är viktigt att alla funktioner som anges nedan har exakt samma namn som i uppgiften. Inkludera raden `(provide dmap)` i din fil.

I Racket finns funktionen `map`, som tar en funktion `f` och en lista `seq` och returnerar en lista som ser likadan ut, med undantag för att `f` applicerats på vart och ett av elementen.

Ett exempel:

```
> (define (square x) (* x x))
> (map square '(1 2 3))
'(1 4 9)
```

Jämför `map-to-each` i laboration 2.

Du får nu in en lista med listor, modell `'((a b c) (d e f))`. Exakt en nivå av listor-i-listor kan förekomma. Till exempel förekommer inte listor som `((1 2 (3 4 5)) (7 8))` eller `(1 2)`.

Definiera en funktion `dmap` som tar en funktion `f`, en lista av listor `sseq` och returnerar en likadan lista av listor, men med `f` applicerat på vart och ett av värdena.

Din kod *måste* använda `map`, och vara på formen

```
(define (dmap f sseq)
  (map
    ?????
    sseq))
```

Så här ska det fungera:

```
> (dmap square '((1 2 3) (4 5 6)))
'((1 4 9) (16 25 36))

> (dmap square '((1 2 3) () (4 5 6)))
'((1 4 9) () (16 25 36))

> (dmap (lambda (x) 'a) '((1 2 3) (4) (5)))
'((a a a) (a) (a))
```