Försättsblad till skriftlig tentamen vid Linköpings Universitet

Duggan skall EJ ske anonymt

(fylls i av ansvarig)

Datum för dugga/tentamen	*10-02-18*
Sal	*TER2*
Tid	*8-10*
Kurskod	*TDDC74*
Provkod	*Del av TEN1 (Dugga 2)*
Kursnamn/benämning	*Programmering-abstraktion och modellering*
Institution	*IDA*
Antal uppgifter som ingår i tentamen	*5*
Antal sidor på tentamen (inkl. försättsbladet)	*5 blad*
Jour/Kursansvarig	*Anders Haraldsson*
Telefon under skrivtid	*Akn. 1403 eller 0705147709*
Besöker salen ca kl.	*9*
Kursadministratör (namn + tfnnr + mailadress)	*Anna Grabska Eklund Ank. 2362, annek@ida.liu.se*
Tillåtna hjälpmedel	*Inga*
Övrigt (exempel när resultat kan ses på webben, betygsgränser, visning, övriga salar tentan går i m.m.)	
Vilken typ av papper ska användas, rutigt eller linjerat	*Duggan lämnas in.*
Antal exemplar i påsen	

Tekniska högskolan vid Linköpings universitet
Institutionen för datavetenskap
Anders Haraldsson

TDDC74 Programmering, abstraktion och modellering

DUGGA 2

Torsdag 18 feb 2010 kl 8-10

Namn: _____

Personnummer: _____

Skriv även ditt namn på varje uppgiftssida.

Uppgifterna löses direkt på denna uppgiftslapp, som lämnas in i sedvanligt tentamensomslag.

Skriv tydligt så att inte dina lösningar missförstås. Använd väl valda namn på parametrar etc. Skriv lagom stort, en del skriver så smått så att vi har svårt att tolka texten och en del skriver så stort så at det inte ryms på raden. Börja raden så långt till vänster som möjligt.

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

På uppgifterna kan halva poäng utdelas.

Betygsgradering: Det är tre duggor. Varje dugga ger 12p, dvs totalt 36p. För att passera en dugga krävs minst 3p på duggan. Totalt skall du på de tre duggorna för betyget 3 ha minst 20p. För betyget 4 minst 25p och för betyget 5 minst 30p.

Lycka till

Uppgift 2. Elementära operationer på listor (4 poäng)

Vi representerar en 2-dimensionell matris som en lista med listor. Varje rad bildar en lista. Vi kan antaga att varje matris alltid har lika långa rader. Exempel: En 3 x 4 dimensionell matris

```
((1 2 3 1)
 (4 5 6 4)
 (7 8 9 7))
```

Funktionerna i de olika deluppgifterna får återanvändas i andra deluppgifter, om så är lämpligt.

2a. (1p) Skriv först en *rekursiv* funktion `(n-element n lista)`, som tar ut det n:te (n > 0) elementet i en lista. Om det ej finns n element i listan skall symbolen `inget-element` returneras.

```
(n-element 2 '(a b c d)) ⇒ b
(n-element 5 '(a b c d)) ⇒ inget-element
```

```
(define (n-element n lista)
```

2b. (1p) Skriv en funktion `(matris-element matris rad kol)`, som givet en matris och rad- och kolumnindex ger motsvarande element. Om rad- eller kolumnindex ligger utanför matrisen returneras `inget-element`.

```
(define en-3x4-matris '((1 2 3 1) (4 5 6 4) (7 8 9 7)))

(matris-element en-3x4-matris 2 3) => 6
(matris-element en-3x4-matris 1 5) => inget-element
(matris-element en-3x4-matris 4 2) => inget-element
```

```
(define (matris-element matris rad kol)
```

[1] Motsvarande Scheme procedure `list-ref` eller motsvarande får ej användas.

Uppgift 3. Godtyckliga strukturer (2 poäng)

Skriv en funktion `(räkna element lista)`, som i en godtycklig lista (som ej innehåller punkterade par) räknar antalet förekomster av ett givet element.

```
(räkna 'a '(a (a b) ((a c) a))) => 4
```

```
(define (räkna element lista)
```

4a. (1,5p) Första uppgiften är att du bestämmer en egen representation av sorteringsträdet och visar detta genom att definiera de primitiva funktionerna för den abstrakta datatypen (gränssnittet).

Hur representeras sorteringsträdet?

Definition av konstruktorfunktionerna:

```
(define (skapa-nod[2] nyckel värde vänster-delträd höger-delträd)


(define (skapa-tomt-träd)
```

Definition av selektorfunktionerna:

```
(define (nod-nyckel nod)
(define (nod-värde nod)
(define (vänster-del nod)
(define (höger-del nod)
```

Definition av jämförelsefunktioner:

```
(define (löv? nod)
(define (tomt-träd? nod)
```

4b. (1,5p) Definiera sedan en funktion `(slå-upp-värde` *`nyckel sort-träd`*`)`, som söker i sorteringsträdet efter nyckeln och returnerar värdet om detta finns, annars returneras symbolen `finns-inget-värde` som värde. Funktionen skall använda de införda primitiverna.

```
(slå-upp-värde 3 test-träd) => tre
(slå-upp-värde 6 test-träd) => finns-inget-värde
```

```
(define (slå-upp-värde nyckel nod)
```

[2] Primitiven `skapa-löv` behöver ej definieras, den använder endast `skapa-nod`.