# Tentamen (del 1) (6 högskolepoäng) i Programkonstruktion och datastrukturer (1DL201)

Lars-Henrik Eriksson
Fredag 21 december 2012, kl. 08:00 – 11:00

**Hjälpmedel:** Inga. ***Inte*** heller elektronisk utrustning.

**Hjälp:** Lars-Henrik kommer att besöka skrivsalen c:a klockan 09:00.

**Anvisningar:** Markera i tabellen nedan ***inte mer än ett*** svar per fråga genom att kryssa över bokstaven för det svarsalternativ som du väljer. ***Lämna bara in denna sida.*** Det är inte meningen att du skall lämna kommentarer till dina svar. Om du tycker att någon fråga är oklar eller felaktig, markera fråganumret med en $\star$ på ***den här*** sidan, och förklara ***på baksidan av detta blad*** vad du menar att problemet är och vilka antaganden du gjort för att kunna svara på frågan.

| | Fråga | Svar | | | | | Fråga | Svar | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| betyg **3** U | 1 | A | B | C | D | E | 2 | A | B | C | D | E |
| | 3 | A | B | C | D | E | 4 | A | B | C | D | E |
| | 5 | A | B | C | D | E | 6 | A | B | C | D | E |
| | 7 | A | B | C | D | E | 8 | A | B | C | D | E |
| | 9 | A | B | C | D | E | 10 | A | B | C | D | E |
| betyg **4** | 11 | A | B | C | D | E | 12 | A | B | C | D | E |
| | 13 | A | B | C | D | E | 14 | A | B | C | D | E |
| | 15 | A | B | C | D | E | | | | | | |
| betyg **5** | | | | | | | 16 | A | B | C | D | E |
| | 17 | A | B | C | D | E | 18 | A | B | C | D | E |
| | 19 | A | B | C | D | E | 20 | A | B | C | D | E |

**Identitet:** Din tentakod (eller namn och personnummer om du saknar kod):

. . . . . .

# Frågeunderlag

En del av frågorna kommer att behandla programmet `fillInForm` som tar en sträng och en stränglista som argument. Strängen skall betraktas som ett "formulär" där text som kan fyllas i representeras av ett dollartecken. Funktionen skall returnera formuläret, men med varje förekomst av ett dollartecken utbytt mot successiva element i listan! Om det finns färre dollartecken än element i listan så ignoreras de extra elementen. Till exempel så skall alltså anropet

```
fillInForm("Bäste $. Jag har glädjen meddela dig att du vunnit $ på vårt
$lotteri.", ["Lars-Henrik", "en semester", "vår"])
```

beräkna strängen

```
"Bäste Lars-Henrik. Jag har glädjen meddela dig att du vunnit en semester på vårt
 vårlotteri."
```

Funktionsspecifikation och programkod för `fillInForm` är:

`fillInForm(form,fill)`
**TYPE:** `string*string list -> string`
**PRE:** `length fill` skall vara minst lika med antalet dollartecken i `form`.
**POST:** `form`, med varje förekomst av dollartecken utbytta mot element i `fill`, i tur och ordning.
**EXAMPLE:** `fillInForm("a $ c $ e",["b","d"]) = "a b c d e"`

```
fun fillInForm(form,fill)  = fillInForm'(form,fill,0)
```

Nedan finns funktionsspecifikation och programkod för hjälpfunktionen `fillInForm'`. Båda är ofullständiga – det finns luckor i dem, markerade `?1?`, `?2?` etc. Din uppgift blir att tala om vad som skall finnas i luckorna.

`fillInForm'(form,fill,pos)`
**TYPE:** `?1?`
**PRE:** `?2?`
**POST:** Delsträngen av `form` från och med position `pos` till strängens slut, med varje förekomst av dollartecken utbytt mot element i `fill`, i tur och ordning.
**EXAMPLE:** `fillInForm'("a $ c $ e",["d"],4) = "c d e"`
**VARIANT:** `?6?`

```
fun fillInForm'(form,fill,pos) =
      if ?3? then
        ""
      else if String.substring(form,pos,1) = "$" then
        hd fill ^ ?4?
      else
        String.substring(form,pos,1) ^ ?5?
```

# Frågor för betyg 3

Om du ger rätt svar på 7 av de 10 frågorna i detta avsnitt så blir du godkänd med minst betyg **3**, annars blir du underkänd (**U**). Du kan inte kompensera ett dåligt resultat i detta avsnitt med poäng från frågorna för betyg **4** eller **5**.

1. Vilken typ skall stå vid `?1?` ovan?
   (A) `string*string list*int -> string` (B) `string*list*int -> string`
   (C) `string*string*int -> string` (D) `string -> string`
   (E) `string*string list*int -> string list`
   **Motivering:**

2. Vilket av följande måste ingå i förvillkoret vid `?2?` ovan? (Det kan alltså behövas ytterligare villkor.)
   (A) `pos = 0`
   (B) `pos >= 0`
   (C) `length fill > 0`
   (D) `length fill > size form`
   (E) `size form > 0`

   **Motivering:** `pos` måste vara $\geq 0$ eftersom den blir positionsargument till `String.substring`. Ingen av de andra kraven är nödvändiga. För ett fullständigt förvillkor krävs dessutom att `pos <= size form` och att `length fill` skall vara minst lika med antalet dollartecken i `form` från och med position `pos`.

3. Vilken kod skall stå vid `?3?` ovan?
   (A) `pos = 0` (B) `pos > 0` (C) `pos = size form` (D) `pos <= size form`
   (E) `form = ""`
   **Motivering:** Basfallet i rekursionen är när positionen `pos` har passerat slutet av strängen `form`, dvs inga tecken finns kvar att arbeta med.

4. Vilken kod skall stå vid `?4?` ovan?
   (A) `fillInForm'(form,fill,pos+1)`
   (B) `fillInForm'(form,tl fill, pos+1)`
   (C) `fillInForm'(form,tl fill, pos+size(hd fill))`
   (D) `fillInForm'(form,fill, pos+size(hd fill))`
   (E) `fillInForm'(String.substring(form,pos,size(hd fill)), tl fill, pos+size(hd fill))`
   **Motivering:**

5. Vilken kod skall stå vid `?5?` ovan?
   (A) `fillInForm'(form,fill,pos+1)`
   (B) `fillInForm'(form,tl fill, pos+1)`
   (C) `fillInForm'(form,tl fill, pos+size(hd fill))`
   (D) `fillInForm'(form,fill, pos+size(hd fill))`
   (E) `fillInForm'(String.substring(form,pos,size(hd fill)), tl fill, pos+size(hd fill))`
   **Motivering:**

6. Vilken variant skall stå vid `?6?` ovan?
   (A) `pos` (B) `size form` (C) `size form - pos`
   (D) `length fill` (E) `length fill - pos`
   **Motivering:** `pos` ökar med ett i varje rekursivt anrop, basfallet är när den blir lika med `size form`.

7. Man vill skriva ett program som hanterar studentregister. Studentregistret skall innehålla ett varierande antal studenter. För varje student skall det finnas uppgift om

   - Namn
   - Personnummer
   - Avklarade kurser och betyg för varje kurs

   Man vill definiera en ny datatyp som skall representera sådana studentregister:

   ```
   datatype studentReg = studentReg of ...
   ```

   Vilket av följande alternativ är ***mest rimligt*** som fortsättning av deklarationen?

   (A) `(string*string*string*int) list`
   (B) `string list*string list*(string*int) list`
   (C) `string list*string list*string list*int list`
   (D) `(string*string*(string*int) list) list`
   (E) `string*string*string*int`

   **Motivering:** Studentregistret skall bestå av en upprepning (lista) av studentuppgifter. Varje studentuppgift består av namn, personnummer och förteckning (lista) över avklarade kurser. Varje avklarad kurs identifieras med namn (t.ex.) och betyg.

8. Vad är värdet av uttrycket `foo [1,2,2,2]` om `foo` är definierad så här?

   ```
   fun foo [] = []
     | foo [x] = [x]
     | foo (first::rest) =
         if first = hd rest then
           foo rest
         else
           first:: foo rest;
   ```

   (A) `[1]` (B) `[1,2]` (C) `[1,2,2]` (D) `[2,2]` (E) `[2,2,2]`

   **Motivering:** Se problem 13 för en förklaring av vad funktionen gör.

9. Vilken är en strikt ("tight") asymptotisk begränsning av körtiden för funktionen `foo` i problem 8? Låt $n$ vara antalet element i funktionens listargument.

   (A) $\Theta(\lg n)$ (B) $\Theta(n)$ (C) $\Theta(n \cdot \lg n)$ (D) $\Theta(n^2)$ (E) $\Theta(2^n)$

   **Motivering:** Antingen använder man teorem 1 med a=1, eller så noterar man helt enkelt att tidsåtgången för alla funktionsanrop i funktionsdefinitionen är oberoende av listans längd *utom* för de rekursiva anropen. Endast ett av de två rekursiva anropen utförs och listans längd minskar med ett. Alltså blir antalet rekursiva anrop lika med listans längd, dvs $n$.

10. Vilken är en strikt (“tight”) asymptotisk begränsning av körtiden för funktionen `intToString` nedan? Antag att $n \geq 0$ och att `digitToString` arbetar i konstant tid. Du behöver i detta fall inte ta hänsyn till tidsåtgången för strängkonkateneringen (`^`).

```
fun intToString n =
      if n < 10 then
        digitToString n
      else
        intToString(n div 10) ^ digitToString(n mod 10)
```

(A) $\Theta(\lg n)$ (B) $\Theta(n)$ (C) $\Theta(n \cdot \lg n)$ (D) $\Theta(n^2)$ (E) $\Theta(2^n)$

**Motivering:** Antingen Master Theorem, fall 2, med $a = 1$ och $b = 10$, eller så noterar man att tidsåtgången för alla funktionsanrop i funktionsdefinitionen är oberoende av listans längd *utom* för det rekursiva anropet. I det rekursiva anropet delas argumentet med 10. Efter c:a $\log_{10} n$ anrop har argumentet blivit $< 10$ och basfallet är nått.

# Frågor för betyg 4

Om du fått minst betyg **3** genom dina svar på de föregående frågorna och dessutom svarar rätt på minst 3 av de 5 frågorna i detta avsnitt så blir du godkänd med minst betyget **4**. Du kan inte kompensera ett dåligt resultat i detta avsnitt med poäng från frågorna för betyg **3** eller **5**.

11. Vilket av följande påståenden om beräkningar i ML är ***felaktigt***?
    (A) I ML beräknas argumenten till en funktion innan funktionen anropas.
    (B) Argumenten till funktioner beräknas i ordning från vänster till höger.
    (C) Resultatet av ett anrop till en funktion (utom inbyggda funktioner) kan beskrivas som att man beräknar funktionskroppen efter att ha bytt ut identifierarna i funktionshuvudet mot motsvarande (delar av) argumenten i funktionsanropet.
    (D) En funktion utan argument beräknas inte utan är ett värde i sig själv.
    (E) I ett if-uttryck beräknas först testet, sedan then-uttrycket och else-uttrycket. Därefter väljs värdet av then-uttrycket eller else-uttrycket beroende på om testet gav true eller false.

    **Motivering:** I ett if-uttryck beräknas först testet, sedan *antingen* then-uttrycket *eller* else-uttrycket, beroende på om testet gav true eller false.

12. Vilket av nedanstående testfall för funktionen `foo` från problem 8 är onödigt för full kodtäckning. Eller behövs allihop?
    (A) `foo[] = []` (B) `foo[1] = [1]` (C) `foo[1,2] = [1,2]`
    (D) `foo[2,2] = [2]` (E) Alla behövs.
    **Motivering:** Både (C) och (D) behövs för att få med båda grenarna i `if`-uttrycket. I rekursionen kommer man till basfallet (B) som alltså är onödigt som separat testfall. Fallet (A) kan man inte komma till med rekursion, så det krävs separat.

13. Titta på funktionen `foo` från problem 8. Låt argumentet kallas `l`.

    Här är fyra förslag till eftervillkor för `foo`. Alla är i någon mening korrekta, men vilka är ***acceptabla*** eftervillkor i en funktionsspecifikation enligt de principer vi gått igenom i kursen?

    (a) Listan `l`, där alla sekvenser av likadana element i följd ersatts med ett enda element.
    (b) Dubbletter borttagna från listan.
    (c) Om listan är tom eller har ett element så ger man tillbaks listan. Om första och andra elementet är samma så gör man rekursion över resten av listan, annars consar man första elementet på rekursion över resten.
    (d) `[]` om `l=[]`. `[x]` om `l=[x]`. Annars anropas `foo` rekursivt med `tl` av `l` som argument. Om första och andra elementet i `l` är olika så bildar man en cons av första elementet och resultatet av rekursionen.

    (A) Bara a (B) a och b (C) a och d (D) a, c och d (E) Alla fyra.
    **Motivering:** (a) är tydlig och svarar på frågan "vad?". (b) är otydlig. (c) och (d) svarar på frågan "hur?".

14. Vilken typ har funktionen `foo` från problem 8?

(A) `int list` (B) `int list -> int list` (C) `''a list`
(D) `''a list -> ''a list` (E) `''a list -> ''b list`
**Motivering:** Funktionen är polymorf, även om exemplen använde heltalslistor. Dubbla apostrofer visar på likhetstyp eftersom funktionen gör likhetsjämförelser på elementen.

15. Programmeraren Putte försöker göra en snabbare sortering och tänker sig en variant av mergesort där man delar listan i tre (i stället för två) lika stora delar, rekursivt sorterar alla tre var för sig och sedan gör en "trevägsmerge" för att sätta ihop resultatet. Frågan är om det lönar sig komplexitetsmässigt i förhållande till vanlig mergesort.

```
fun mergesort3 [] = []
  | mergesort3 [x] = [x]
  | mergesort3 l =
      let
        val (x,y,z) = split3 l
      in
        merge3(mergesort3 x, mergesort3 y, mergesort3 z)
      end
```

Antag att `split3` delar upp listan i tre lika stora delar (så gott som det går) och har linjär tidskomplexitet i längden av sitt argument. Antag också att `merge3` är linjär i summan av längderna av sina tre argument.

Låt $n$ vara antalet element i listargumentet till `mergesort3`. Vilken blir tidskomplexiteten för funktionen `mergesort3`?

(A) $\Theta(n)$ (B) $\Theta(n \cdot \lg n)$ (C) $\Theta(n^{\log_3 2})$ (D) $\Theta(n^{\log_3 2} \cdot \lg n)$ (E) $\Theta(n^{\log_2 3} \cdot \lg n)$
**Motivering:** Master Theorem fall 2 med $a = b = 3$ och $f(x) = \Theta(2n) = \Theta(n)$. Man vinner alltså inte i tidskomplexitet genom att dela upp listan i flera delar.

# Frågor för betyg 5

Om du fått minst betyg **4** genom dina svar på de föregående frågorna och dessutom svarar rätt på minst 3 av de 5 frågorna i detta avsnitt så blir du godkänd med betyg **5**. Du kan inte kompensera ett dåligt resultat i detta avsnitt med poäng från frågorna för betyg **3** eller **4**.

16. Titta på funktionen `flatten`:

```
fun flatten [] = []
  | flatten [x] = x
  | flatten ([]::rest) = flatten rest
  | flatten (first::rest) = (hd first):: flatten((tl first)::rest);
```

`flatten` är en – inte speciellt bra skriven – funktion som "plattar ut" en lista av listor, dvs sätter ihop elementen till en lång lista.
Exempel: `flatten [[1,2],[],[3],[4,5,6]] = [1, 2, 3, 4, 5, 6]`

En viktig fördel med listor som implementeras med cons-celler är att två olika listor kan dela samma cons-celler vilket sparar utrymme i datorn. En nackdel är att programmet under sin körning kan skapa cons-celler "i onödan" – dvs cons-celler som inte blir en del av resultatet.

Exempel:

```
val x = [1,2]
val y = tl(0::1::tl x)
```

`y` binds här till `[1,2]`. Den första cons-cellen i `y` är nykonstruerad, medan den andra cons-cellen delas med `x`. Under beräkningen av `y` skapas ytterligare en cons-cell (med huvudet `0`), vilken inte ingår i värdet `[1,2]`.

Antag att man gör anropet `flatten [[1,2],[3,4]]`, vilket ger resultatet `[1,2,3,4]`. Frågan är nu: hur många cons-celler delar resultatet med argumentet till `flatten` och hur många cons-celler skapar `flatten` som ***inte*** ingår i resultatet?

(A) 0 resp. 0 (B) 0 resp. 4 (C) 2 resp. 0 (D) 2 resp. 1 (E) 2 resp. 2
**Motivering:** Cons-cellerna i argumentets sista element ( [3,4]) delas med resultatet. Cons-cellerna i det rekursiva anropet kommer inte med i resultatet.

17. Titta på funktionen `bar` som utför division av positiva heltal med hjälp av upprepad subtraktion:

```
fun bar(x,y) = if x<y then 0 else 1+bar(x-y,y);
```

Vilket är det *minst restriktiva* förvillkor (det som tillåter flest olika argument) som funktionen `bar` kan ha om den inte skall fastna i en oändlig rekursion? (Vi bryr oss inte om ifall resultatet blir meningsfullt eller inte.)

(A) $y \neq 0$ (B) $y > 0$ (C) $x \geq 0$ ***och*** $y > 0$ (D) $x < y$ ***eller*** $y > 0$ (E) Något annat.
**Motivering:** Om $x < y$ så terminerar uppenbarligen funktionen. I annat fall måste $x$ kunna minska tills det är mindre än $y$, och det är bara möjligt om $y > 0$

18. Man matar in följande deklarationer till ett ML-system (i denna ordning):

```
val x = 1
val y = 2
fun fie x =
   let
     val x = y+1
   in
     x+y
   end
val y = 3
```

Därefter ger man – i tur och ordning – ML-systemet uttrycken `fie 0` och `x` att beräkna. Vilka värden beräknas?
(A) 7 och 1 (B) 7 och 4 (C) 5 och 3 (D) 5 och 1 (E) 4 och 1
**Motivering:** Bindningen av $x$ inuti $fie$ påverkar inte värdet av $x$ utanför. Den andra bindningen av $y$ påverkar inte värdet av $y$ inuti $fie$, där den första bindningen av $y$ används.

19. Antag att `f` och `g` är funktioner som båda beräknar samma sak, men med olika algoritmer. Låt $n$ beteckna storleken på indata till funktionerna.Tidskomplexiteten för `f` är $\Theta(n \cdot \lg n)$ i både det genomsnittliga och värsta fallet. Tidskomplexiteten för `g` är $\Theta(n^2)$ i värsta fallet. I genomsnittliga fallet har även `g` tidskomplexitet $\Theta(n \cdot \lg n)$.

Man vill välja endera av `f` eller `g` för att få den kortaste körtiden. Vilket av dessa påståenden är korrekt?

(A) Det är alltid bättre att använda `f`.
(B) För indata med lämpliga egenskaper är det tänkbart att `g` är bättre att använda oavsett storlek på indata.
(C) I genomsnitt spelar det ingen roll om man använder `f` eller `g`.
(D) Det enda som avgör om man skall använda `f` eller `g` är den förväntade storleken på indata. `f` är alltid bättre om indata är tillräckligt stort.
(E) Det enda som avgör om man skall använda `f` eller `g` är den förväntade storleken på indata. `g` är alltid bättre om indata är tillräckligt stort.

**Motivering:** Att `f` och `g` har olika tidskomplexitet i det värsta fallet betyder att *för tillräckligt stora* indata så går `f` snabbare än `g`. För små indata kan det vara tvärtom. (A) är alltså fel. I det genomsnittliga fallet har funktionerna samma komplexitet, men den ena kan ändå vara snabbare än den andra (men inte *asymtotiskt* snabbare. (C) är alltså fel. Samma sak gäller om man väljer indata så att man undviker de fall där `g` har kvadratisk tidskomplexitet. (D) och (E) är alltså fel, men (B) rätt.

20. En inordertraversering (“inorder walk”) av ett binärt sökträd ger en lista där informationen i trädet kommer i stigande ordning. Ett sätt att sortera en lista vore alltså att lägga in varje element i listan i ett (från början tomt) binärt sökträd och sedan göra en inordertraversering. Om vi förutsätter att man använder ett balanserat binärt sökträd och en algoritm för traverseringen som tar linjär tid, vilken blir tidskomplexiteten i värsta fallet för en sådan “trädsortering”? ($n$ är storleken på listan som sorteras.)

(A) $\Theta(n)$ (B) $\Theta(n \cdot \lg \lg n)$ (C) $\underline{\Theta(n \cdot \lg n)}$ (D) $\Theta(n \cdot \lg n \cdot \lg n)$ (E) $\Theta(n^2)$

**Motivering:** Den naturliga funktionen för att lägga in element i trädet är

```
fun build [] = empty
| build(x::l) = insert(build l,x)
```

Rekursionsformeln för tidsåtgången av `build` blir $T(n) = T(n-1) + \Theta(\lg n - 1) = T(n-1) + \Theta(\lg n)$ eftersom insättning i ett balanserat binärt sökträd alltid går att göra i tid $\Theta(\lg n)$. Med induktion visas nu att $T(n) = \Theta(n \cdot \lg n)$. Lägger vi till tidskomplexiteten för traverseringen som är $\Theta(n)$, så får vi $\Theta(n \cdot \lg n) + \Theta(n) = \Theta(n \cdot \lg n)$. Detta gäller både i genomsnittliga och värsta fallet.