

# TENTAMEN I CD5100 FUNKTIONELL PROGRAMMERING

Onsdagen den 17 januari 2007, kl 14:30 – 19:30

Kurslitteratur är inte tillåten, och inte eller andra hjälpmedel som på något sätt kan ersätta kurslitteraturen (t.ex. egna anteckningar, andra böcker i ämnet, kopior av OH-bilder, datorer eller räknare med dito lagrad information). Endast generella hjälpmedel är tillåtna, som räknare utan lagrad information av betydelse för kursen, ordbok, allmän formelsamling och liknande. För godkänt krävs 15 poäng, max är 30 poäng. Resultatet offentliggörs senast onsdagen den 7 februari 2007.

Vänligen observera följande:

- Motivera alltid dina svar. Bristande motivering kan ge poängavdrag. Omvänt kan även ett felaktigt svar ge poäng, om det framgår av motiveringen att tankegången ändå är riktig.
- Skriv tydligt!
- Varje blad skall vara försedd med namn, personnummer, uppgiftsnummer och bladnummer.
- Endast en uppgift på ett och samma blad.
- Skriv enbart på ena sidan av ett blad.
- Uppgifterna är inte nödvändigtvis sorterade i svårighetsgrad. Om du kör fast kan det löna sig att gå vidare till nästa uppgift.
- Lösningförslag kommer att finnas på kursens hemsida efter att tentan är slut.

Jour: Björn Lisper på 021-151709.

---

---

## UPPGIFT 1 (8 POÄNG)

Definiera en funktion `minmax` som tar en lista som argument och returnerar paret av det minsta och största elementet i listan. Du ska definiera funktionen på två sätt:

a) Med hjälp av någon eller några av Haskell's inbyggda högre ordningens funktioner. Din deklaration ska vara så kort och lättförståelig som möjligt. (3p)

b) Så effektivt som möjligt. Speciellt så får din funktion inte gå igenom listan mer än en gång. (3p)

c) Om du inte ger några egna typdeklarationer, vad bör `minmax` ges för typ av Haskell's typsystem? Du behöver inte pretera en fullständig typhärledning, men du måste ge åtminstone en kort motivering. (2p)

Du får ett av göra följande antaganden: antingen behöver `minmax` bara fungera på listor som inte är tomma, eller så är typen för elementen i listan en instans av Haskell's inbyggda klass `Bounded`. Denna klass har som metoder två konstanter `minBound` och `maxBound`, vilka står för det minsta respektive största elementet i typen.

**Ledning:** Haskell har två funktioner `foldr1`, `foldl1 :: (a -> a -> a) -> [a] -> a` som är varianter av `foldr` och `foldl`. Skillnaden är att de inte fungerar för den tomma listan, och de behöver då inte ta som argument det element som ska returneras för den tomma listan.

## UPPGIFT 2 (5 POÄNG)

a) Definiera en egen datatyp för binära träd, där man kan lagra data av en viss typ både i de interna noderna och i löven! Din datatyp ska vara polymorf. (2p)

b) Definiera en IO action som tar ett träd av din typ och skriver ut dess innehåll på skärmen i *inorder*. (En inordergenomgång av ett binärt träd går först rekursivt ned i vänster delträd, sen tar man rotenoden, och sen höger delträd.) Du får anta att typen på de data som lagras i trädet är en instans av klassen *Show*. (3p)

### UPPGIFT 3 (3 POÄNG)

Betrakta följande funktionsdeklarationer:

```
f x 0 = 0
```

```
f x 1 = x
```

```
g y = y + g (y + 1)
```

Vad blir resultaten i Haskell av följande funktionsanrop:

- `f (g 1) 0`

- `f (g 1) 1`

Motivera ditt svar!

### UPPGIFT 4 (6 POÄNG)

Det kan vara trevligt att ha en funktion som talar om hur stor plats i minnet ett datum tar. Man vill gärna att en sådan funktion ska fungera på olika typer: alltså kan det vara läge att använda Haskell's klass-system för detta.

Deklarera en klass *Sized*, med en metod *size* som tar ett datum av typen ifråga och returnerar dess storlek i minnet, räknat i antalet bitar, som en *Int*! Gör sen typen *Int* själv till en instans av *Sized*. Gör slutligen `[a]` till en instans av *Sized*, under förutsättning att *a* är en instans av *Sized*.

För att lösa uppgiften måste man förstås känna till eller anta en del detaljer om hur ens Haskell-implementering representerar data. Du får anta att: en *Int* tar 32 bitar, att `[]` tar 32 bitar, samt att en cons-cell tar 64 bitar (två pekare à 32 bitar vardera). Redovisa vilka övriga antaganden du har gjort om datarepresentationer.

### UPPGIFT 5 (3 POÄNG)

Vilka av Haskell's inbyggda typer (som vi gått igenom i kursen) är *Inte* instanser av klassen *Eq*? Förklara varför!

### UPPGIFT 6 (5 POÄNG)

Bevisa att `map f . tail = tail . map f`!

**Ledning:** `map` och `tail` får antas vara definierade på följande sätt:

```
map f [] = []
```

```
map f (x:xs) = f x : map f xs
```

```
tail [] = error "tail of empty list"
```

```
tail (x:xs) = xs
```

Lycka till!

*Björn*