TENTAMEN I DVA 229 FUNKTIONELL PROGRAMMERING MED F#

Torsdagen den 25 mars 2021, kl 14:30 – 18:30

Detta är en hemtentamen. Därför gäller lite andra regler än vanligt. Samarbete med eller hjälp från andra personer är *inte* tillåtet. I övrigt är alla hjälpmedel tillåtna (VisualStudio, MSDN, kursmaterial, etc.) Man får dock inte basera sin lösning på kod som är plankad från t.ex. nätet (OBS att vi kan kontrollera t.ex. med urkund). Tentan lämnas ut via Canvas 14:30 och inlämning sker via Canvas i form av en pdf-fil innan 18:30 den 25 mars. För godkänt krävs 15 poäng, max är 30 poäng. Resultatet offentliggörs senast torsdagen den 15 april 2021.

Vänligen observera följande:

- Om inte annat sägs ska lösningarna vara rent funktionella, dvs. sånt som muterbara variabler och sidoeffekter får inte förekomma.
- 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.
- Ge klara och tydliga förklaringar! Oklara och röriga förklaringar kan ge avdrag.
- Märk dina lösningar med namn och personnummer. (P.g.a. omständigheterna går det inte att ha en anonym tenta.)
- Endast en uppgift på ett och samma sida. Markera uppgiftsnumret tydligt. Se också till att sidorna är numrerade.
- Uppgifterna är inte nödvändigtvis sorterade i svårighetsgrad. Om du kör fast kan det löna sig att gå vidare till nästa uppgift.
- Lösningar som inkommer efter 18:30 kommer inte att bli rättade. Undantag kommer bara att göras om orsaken är något som helt klart ligger utanför er kontroll, t.ex. om nätet går ned.

Frågor på själva tentan: Björn Lisper på 0739-607199. Tekniska frågor (Canvas mm): Jean Malm på 070-9658574.

UPPGIFT 1 (6 POÄNG)

När man gör signalbehandling *normaliserar* man ofta tidssignaler. Detta innebär att man först bildar medelvärdet av tidssignalen och sen drar medelvärdet från vart och ett av elementen i serien. Resultatet blir en tidsserie som är balanserad runt 0, vilket kan ha en del fördelar i den fortsatta signalbehandlingen.

a) Deklarera en funktion som tar en tidsserie, representerad som en lista av flyttal (`float`), och returnerar dess balanserade version enligt ovan! Använd direkt rekursion. Förklara i detalj hur din lösning fungerar, speciellt rekursionen! Ange också vad din funktion får för typ, och motivera varför funktionen får den typen. *(4p)*

b) Gör en alternativ deklaration av funktionen i *a)* som använder sig av någon eller några inbyggda högre ordningens listfunktioner i F# på ett vettigt sätt. Förklara hur din lösning fungerar och framförallt hur de högre ordningens funktionerna används! *(2p)*

UPPGIFT 2 (4 POÄNG)

Balanserad summering av en array går till på följande sätt:

- om arrayen är tom, returnera noll,
- om arrayen har ett enda element, returnera det elementet,
- annars dela arrayen i två ungefär lika stora delar, summera delarna rekursivt och returnera summan av delsummorna.

Deklarera en funktion som summerar arrayer av flyttal (`float []`) med balanserad summering! För full poäng får inte din lösning vara onödigt slösaktig med minne.

UPPGIFT 3 (4 POÄNG)

a) Har F# lat eller ivrig evaluering vid funktionsanrop? *(1p)*

b) Förklara hur lat resp. ivrig evaluering vid funktionsanrop fungerar. Ge ett exempel där de ger olika resultat. *(3p)*

UPPGIFT 4 (4 POÄNG)

Deklarera en version av `List.fold` som utåt beter sig precis som den vanliga versionen, men internt använder en muterbar referenscell för att ackumulera resultatet. Förklara i detalj hur din lösning fungerar. Lista också de funktioner och operatorer i din lösning som opererar på muterbara referensceller och förklara för var och en av dem vad den operatorn/funktionen gör.

Ledning: det är tillåtet att använda itererade loopar om man så vill.

UPPGIFT 5 (2 POÄNG)

Följande funktionsdeklarationer är ett försök att deklarera två ömsesidigt rekursiva funktioner `f` och `g`:

```
let rec f x = if x = 0 then 1 else x + g (x-1)
let rec g x = if x = 0 then 1 else x + f (x-1)
```

a) Dessa deklarationer har dock ett problem. Vilket? *(1p)*

b) Gör om deklarationerna så de fungerar som avsett! *(1p)*

UPPGIFT 6 (6 POÄNG)

a) Deklarera en F#-datatyp för träd enligt följande:

- en nod i trädet kan vara antingen ett löv eller en intern nod,
- en intern nod han ha två eller fyra söner, samt
- alla noder, både löv och interna noder, ska innehålla data av någon typ `'a` där `'a` är en typvariabel.

Förklara hur din datatyp representerar dessa träd. *(2p)*

b) Deklarera en funktion som räknar antalet noder med fyra söner i ett träd av typen ovan. Förklara klart och tydligt hur din funktion fungerar och framförallt hur trädet traverseras. *(4p)*

UPPGIFT 7 (4 POÄNG)

Visa att den algebraiska lagen

```
List.map f >> List.tail = List.tail >> List.map f
```

gäller. Du behöver inte göra ett helt formellt induktionsbevis, det räcker med den informella typen av bevis som vi använt på föreläsning.

Ledning: du behöver specialbehandla fallet att funktionerna appliceras på den tomma listan.

Lycka till! *Björn*