# TENTAMEN I DVA 229 FUNKTIONELL PROGRAMMERING MED F#

*Tisdagen den 7 juni 2022, kl 14:30 – 18:30*

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 torsdagen den 14 april 2022.

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 uppgiftsnummer och bladnummer.
- Endast en uppgift på ett och samma blad.
- Skriv enbart på ena sidan av ett blad.
- Om inte annat sägs ska lösningarna vara rent funktionella, utan sidoeffekter. Det betyder att ni får använda mutables och liknande endast om det uttryckligen är tillåtet.
- Uppgifterna är inte nödvändigtvis sorterade i svårighetsgrad. Om du kör fast kan det löna sig att gå vidare till nästa uppgift.
- Lösningsförslag kommer att finnas på kursens hemsida efter att tentan är slut.

Frågor: Björn Lisper på 0739-607199.

---

**UPPGIFT 1** (6 POÄNG)

Deklarera en funktion `f` som tar en lista `l` av typen `int list` och returnerar en lista som bildas ur `l` på följande sätt:

- negativa element i `l` tas inte med
- för varje icke-negativa element i `l` tas dess värde plus ett med.

T.ex. ska gälla att `f [1;-1;0;-3] = [2;1]`.

*a)* Deklarera en version av `f` som använder rekursion! *(4p)*

*b)* Gör en alternativ deklaration av `f` som använder sig av någon eller några inbyggda högre ordningens listfunktioner i F# på ett vettigt sätt. *(2p)*

**UPPGIFT 2** (4 POÄNG)

Deklarera en funktion som summerar en array av `float` med *balanserad summering*. En sådan summering går till på följande sätt:

- om bara ett tal ska summeras så returnera det. Annars:
- dela upp arrayen i två lika stora delar,

- summera delarna,
- lägg ihop summorna av delarna och returnera.

För enkelhets skull får du anta att arrayens storlek är en jämn tvåpotens, så att det alltid går att dela arrayen i två lika stora delar. Du får också anta att arrayen inte är tom. För full poäng får lösningen inte vara onödigt ineffektiv.

**UPPGIFT 3** (2 POÄNG)

Betrakta följande anrop till funktionerna `List.take` och `Seq.take`:

```
List.take 1 [1 .. 10000000]
Seq.take 1 {1 .. 10000000}
```

Vilket av anropen kommer att gå snabbast, och varför?

**UPPGIFT 4** (4 POÄNG)

Deklarera en funktion som tar en lista och ett predikat som argument och som returnerar antalet element i listan för vilka predikatet blir sant. Din lösning ska använda direkt rekursion och antalet element ska ackumuleras i en muterbar referenscell. Du får i denna uppgift *inte* använda dig av standardfunktioner som `List.filter` and `List.length`.

**UPPGIFT 5** (2 POÄNG)

Låt `s` vara en sträng. Förklara vad `s.[i]` och `s.[i..i]` är! Skiljer de sig åt och i så fall hur? Eller är de kanske bara två sätt att skriva samma sak?

**UPPGIFT 6** (2 POÄNG)

Betrakta följande kod:

```
let g x = f x
let f x = x
```

Kompilatorn kommer dock inte att godkänna detta. Förklara varför!

**UPPGIFT 7** (6 POÄNG)

Ibland vill man ha träd där en intern nod kan ha ett godtyckligt antal söner. Ett sätt att implementera detta är att låta varje intern nod innehålla en array som i sin tur innehåller delträden. En variant av sådana träd ser ut som följer:

- noder i trädet är antingen löv eller interna noder,
- ett löv innehåller ett data av någon typ `'a`, men inga delträd, samt
- en intern nod innehåller en array av delträd enligt ovan (men inga data).

*a)* Deklarera en polymorf datatyp för denna typ av träd! *(2p)*

*b)* Deklarera en funktion som tar ett träd enligt ovan, som innehåller heltal av typ `int`, och returnerar summan av alla hel talen i trädet! *(4p)*

**UPPGIFT 8** (4 POÄNG)

Betrakta följande funktionsdefinition:

```
let rec f x = let y = 1 in y + f x
```

Härled en typ för `f`! För full poäng ska den härledda typen vara den mest generella. Ordentlig motivering krävs. *(4p)*

Lycka till! *Björn*