1. (4 p)

   En viktig egenskap vid lat evaluering är terminering.

   a) (1 p) Förklara och visa med ett exempel när det inte terminerar oavsett evalueringsstrategi.

   b) (3 p) Förklara varför och visa med ett exempel när det terminerar endast med lat evaluering! OBS! Förklara också varför det inte terminerar då man använder en annan evalueringsstrategi än den för lat evaluering.

Lösning:

a) (1 p) Då funktionen `addOne = addOne + 1` anropas resulterar det i en oändlig antal rekursiva anrop oavsett evalueringsstrategi. Dvs den terminerar inte.

b) (3 p) call-by-name resulterar i att `fst` utförs först och därmed utförs inte alls anropet `addOne`. Vid Lat evaluering används call-by-name och därmed terminerar anropet `fst(0, addOne)`.

Då call-by-value används anropas först funktionen `addOne` vilket enligt a)-uppgiften inte terminerar.

Rättning:

a) (1 p) Ett poäng om rätt svar (exemplet behöver inte vara kod), 0 annars.

b) (3 p) 3 poäng om korrekt.
-1 poäng om man inte exemplifierat eller förklarat innebörden av call-by-name.
-1 poäng om sambandet mellan lat evaluering och call-by-name inte framgår
-1 poäng om man inte exemplifierat eller förklarat innebörden av call-by-value.
Man behöver inte bokstavligen ha nämnt call-by-name och call-by-value, men koncepten ska gå att känna igen i lösningen

2. (8 p)

Följande kod kan används för att implementera matriser i Haskell.

```
data Matris a = M
        Int -- Antal rader
        Int -- Antal kolumner
        [a] -- matriselementen
        deriving (Show, Eq)
```

a) (2 p) För att skapa 2×2-matrisen $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ görs anropet nedan vars resultat lagras i `m1`. Motivera varför resultaten av anropen som lagras i `m2` och `m3` också fungerar, dvs inte ger körningsfel.

```
m1 = M 2 2 [1,2,3,4]
m2 = M 2 2 [[1,2],[1,3],[5,1],[7,8]]
m3 = M 1 1 "qwe"
```

b) (3 p) Med funktionen `matrisOperation` nedan kan man t.ex. addera och subtrahera två matriser. Hur ska anropet se ut för att addera `m1` med sig själv?

c) (3 p) Skriv om raden `elem = [operation el | el <- (elements l1 l2)]` utan att använda listomfattning och så att funktionens funktionalitet bibehålles.

Lösning:

a) Typ-kraven som finns är uppfyllda dvs både antalet rader och kolumner är heltal och matriselementen är i båda fallen en lista. NB! Strängar är en lista med `Char`!

b) `matrisOperation m1 m1 (\(a,b) -> a+b)`

c) `elem = map operation (elements l1 l2)` Man kan också skriva en egen rekursiv funktion.

Rättning:

a) Två poäng för förklaring att matriselementen kan vara en lista av godtycklig typ. -1 poäng om man inte nämner att strängar är listor av Char.

b) Tre poäng om korrekt svar
−2 poäng för om man använder en operator som inte funkar på tupler (t.ex. "`(+)`" eller "`(sum)`")
−3 poäng för om man använder en operator som gör något helt annat (t.ex. "`(++)`") Inget avdrag för struntfel t.ex. "`/`" istället för "\", "`m2`" istället för "`m1`", etc.

c) Tre poäng om korrekt svar.
−1 poäng om man glömmer applicera `elements` på `l1 l2`.
−3 poäng för varianter där man gör `map` två gånger, eller använder `zip`. Inget avdrag för struntfel t.ex. glömma "`elem =`".

3. (8 p)

Lösning:

a) Initialt tilldelas variabler värden t.ex.
resultat = 1 och $n = 4$

Beräkningssteg 1: variablerna uppdateras
resultat = $1 \cdot 4$ och $n = 4 - 1$
Beräkningssteg 2: variablerna uppdateras
resultat = $4 \cdot 3$ och $n = 3 - 1$
Beräkningssteg 3: variablerna uppdateras
resultat = $12 \cdot 2$ och $n = 2 - 1$
Beräkningssteg 4: variablerna uppdateras
resultat = $24 \cdot 1$ och $n = 1 - 1$

Resultatet 24 returneras.
Minnesanvändningen är konstant.

b) Anropet fakultet 4 utvecklas till en kedja med rekursiva anrop:
4 * (fakultet 3) Stacken byggs på
4 * 3 * (fakultet 2)
4 * 3 * 2 * (fakultet 1)
4 * 3 * 2 * 1 * (fakultet 0)
4 * 3 * 2 * 1 * 1 Basfall!
4 * 3 * 2 * 1 Successiv avveckling
4 * 3 * 2
4 * 6
24
24 Resultatet retuneras

Minnesanvändningen är linjär.

c) Imperativ paradigm använder variabler som uppdateras. Upprepning kan utföras med hjälp av iteration, ingen anrops- kedja byggs upp.

Funktionell paradigm har inga variabler. Upprepning sker med rekursion, man får en anropskedja på stacken som allokerar minne tills man når basfallet, först då börjar kedjan avvecklas och minnesutrymmet lämnas tillbaka successivt. Värden skickas med det rekursiva anropet.

Rättning:

a) 1 poäng för beskrivning av beräkningsstegen.
1 poäng för minnesanvändningen.
1 poäng för förklaring hur variabler uppdateras.

b) 1 poäng för beskrivning av anropen.
1 poäng för minnesanvändningen.
1 poäng för förklaring att multiplikationerna utförs sist (successiv avveckling).

c) 2 poäng för korrekt svar.
-1 poäng om bara ena paradigmet beskrivet korrekt, eller om ofullständig förklaring av skillnaden.