Skriftlig Eksamen
Programmeringssprog (DM22)

Institut for Matematik & Datalogi
Odense Universitet

Fredag den 7. januar 2000 kl. 9–13

Alle sædvanlige hjælpemidler (lærebøger, notater etc.) samt brug af lommeregner er tilladt.

Eksamenssættet består af 5 opgaver på 4 nummererede sider (1–4). Fuld besvarelse er besvarelse af alle 5 opgaver. De enkelte opgavers vægt ved bedømmelsen er angivet i procent.

Der må gerne refereres til resultater fra lærebogen. Henvisninger til andre bøger end lærebogen accepteres ikke som led i besvarelsen af et spørgsmål.

Bemærk, at hvis der er et spørgsmål i en opgave, man ikke kan besvare, må man gerne (så vidt det er muligt) besvare de efterfølgende spørgsmål og blot antage, at man har en løsning til de foregående spørgsmål.

Opgave 1 (20%)

De enkelte spørgsmål i denne opgave kan besvares uafhængigt af hinanden.

Spørgsmål a: Betragt Haskell-funktionerne

```
map2 = map map
mapd = map . map
```

Hvad er den mest generelle type for henholdsvis `map2` og `mapd`? Vis de enkelte skridt i udledningen af typerne. □

Spørgsmål b: Betragt følgende Prolog-program:

```
p(1).
p(2) :- !.
p(3).
```

For hver af følgende forespørgsler skal der gøres omhyggeligt rede for *alle* Prolog's svar (svarende til, at der i den interaktive kørsel skrives ";", indtil der ikke er flere svar).

(i) `p(X).`

(ii) `p(X),p(Y).`

(iii) `p(X),!,p(Y).`

□

Opgave 2 (15%)

En **pythagoræisk triple** er defineret som tre positive heltal (x, y, z), der opfylder

$$x^2 + y^2 = z^2.$$

Eksempler er (3,4,5) og (5,12,13). Det vides, at der eksisterer uendeligt mange pythagoræiske tripler.

Spørgsmål a: Skriv ved hjælp af *list comprehensions* en Haskell-funktion

```
triads :: Int -> [(Int,Int,Int)]
```

som opfylder, at `triads n` returnerer en liste indeholdende alle pythagoræiske tripler med elementer mindre end eller lig `n`, ordnet lexicografisk og uden gentagelser. □

Spørgsmål b: Vi ønsker at slippe af med den begrænsning, der ligger i parametren `n`, idet vi vil erstatte `triads` med en (lazy) liste,

```
alltriads :: [(Int,Int,Int)]
```

som kan producere samtlige pythagoræiske tripler "på anfordring". Et oplagt løsningsforslag ville være simpelt hen at udelade `n` fra generatorerne i de anvendte *list comprehensions*. Vil det virke i det konkrete tilfælde? Hvis ikke, beskriv med ord, hvorledes beregningerne kan omstruktureres. Der kræves ikke et egentligt Haskell-program, men en omhyggelig analyse af problemet. □

Opgave 3 (25%)

I denne opgave vil en bestemt slags binære træer blive realiseret i Haskell. Træerne er elementer i en datatype med definitionen

```
data Htree a = Null | Fork a (Htree a) (Htree a)
```

Vi ønsker at konstruere et `Htree` ud fra en ikke-tom liste af `a`-elementer, `[x0, x1, ..., xn]`, så træet bliver balanceret i følgende betydning: I roden anbringes `x0`, på næste niveau `x1` og `x2`, dernæst de fire elementer `x3` til `x6` (hvis der er så mange), og så videre. Det nederste niveau fyldes op fra venstre.

Spørgsmål a: Første skridt i konstruktionen af træet består i at opdele listen i underlister med længder, som er voksende potenser af 2, altså

```
[[x0],[x1,x2],[x3,x4,x5,x6],...]
```

hvor den sidste underliste muligvis ikke når op på den fulde længde. Funktionen, der konstruerer denne opdeling, er specificeret ved

```
levels :: [a] -> [[a]]
levels = levelsWith 1
```

Skriv funktionen `levelsWith`. □

Spørgsmål b: Funktionen `mkHtrees` tager en liste af lister, som returneret af `levels`, og bygger en liste af træer:

```
mkHtrees :: [[a]] -> [Htrees a]
mkHtrees = foldr addLayer [Null]
```

Hvis funktionen `addLayer` anvendes på en liste af `a`-elementer, `[y0,y1,...]`, og en liste af træer `[t0,t1,...]`, producerer den en ny liste af træer:

```
[Fork y0 t0 t1, Fork y1 t2 t3, ...]
```

Hvis listen af træer ikke er lang nok, fyldes der op med et passende antal tomme træer (`Null`). Skriv funktionen `addLayer`. □

Spørgsmål c: Brug ovenstående funktioner til at definere

```
mkHtree :: [a] -> Htree a
```

som bygger et `Htree` ud fra en liste af `a`-elementer. □

Opgave 4 (20%)

I denne opgave vil en bestemt slags binære træer blive realiseret i Prolog. En knude i træet repræsenteres ved et term `n(`v`,`l`,`r`)`, hvor v angiver en ikke-negativ, heltallig værdi, mens l og r står for henholdsvis venstre og højre undertræ. Tomme (under-)træer repræsenteres som `[]`.

Givet et træ i denne repræsentation ønsker vi at skrive en Prolog-funktion, som konstruerer et nyt træ med præcis samme struktur, i hvilket hver eneste værdi er erstattet med den største værdi, som forekommer nogetsteds i det oprindelige træ. Ud fra træet

```
n(3,n(1,n(4,[],[]),n(1,[],[])),
    n(5,n(9,n(2,[],[]),[]),n(6,[],n(5,[],[]))))
```

laves således

```
n(9,n(9,n(9,[],[]),n(9,[],[])),
    n(9,n(9,n(9,[],[]),[]),n(9,[],n(9,[],[]))))
```

Spørgsmål a: Skriv en Prolog-funktion, `maxtree(InTree,OutTree)`, som udfører det ønskede ved hjælp af *to* rekursive gennemløb af træet: et til at finde den største værdi og et til at konstruere det ny træ. □

Spørgsmål b: Man kan udføre konstruktionen ved hjælp af blot ét rekursivt gennemløb af det oprindelige træ, hvis man benytter et passende antal *akkumulatorer* og et "*hul*", analogt med dem man bruger i forbindelse med differensstrukturer. Skriv en ny udgave af `maxtree/2`, som benytter denne teknik. □

Opgave 5 (20%)

De enkelte spørgsmål i denne opgave kan besvares uafhængigt af hinanden. Spørgsmålene er formuleret i sædvanlig prædikatlogiknotation. Om I vil bruge denne eller den mere "Prolog-agtige", benyttet i Clocksin & Mellish kapitel 10, er op til jer.

Spørgsmål a: Omskriv dette logiske udtryk til konjunktiv normalform:

$$\neg(\forall X)(p(X) \Rightarrow ((\forall Y)(p(Y) \Rightarrow p(f(X,Y))) \wedge \neg(\forall Z)(q(X,Z) \Rightarrow p(Z))))$$

□

Spørgsmål b: Vis ved hjælp af *lineær inputresolution*, at nedenstående mængde af clauses er inkonsistent.

$$\neg p \vee \neg q \vee r$$
$$\neg s \vee t$$
$$\neg t \vee p$$
$$s$$
$$\neg r$$
$$\neg s \vee u$$
$$\neg u \vee q$$

□