Ordinær eksamen i Funktionsprogrammering, blok 1 2008

Nærværende dokument udgør opgavesættet for den ordinære eksamen i kurset "Funktionsprogrammering", blok 1 2008.

Dokumentet offentliggøres mandag 27. oktober kl. 9.00 via KUs kursusadministrationssystem ISIS. Besvarelsen skal afleveres senest onsdag 29. oktober kl. 9.00 — se afsnit 2 nedenfor.

Besvarelsen evalueres som bestået/ikke-bestået. Eksamensresultaterne vil findes på ISIS senest tre uger efter eksamens afslutning, og vil findes i det Naturvidenskabelige Fakultets eksamensprotokol umiddelbart derefter.

Eksamenssættet består af 10 opgaver, der hver giver op til 10 point. Af disse gives 8 point for korrekt besvarelse og 2 point for kommentarer, indrykning, variabelnavngivning og generel læsbarhed.

Opgavernes rækkefølge i sættet er uafhængig af deres sværhedsgrad. Besvarelsen vurderes bestået, hvis den vurderes til at have opnået 50 point eller mere (bemærk, at delvist færdige løsninger til enkeltopgaver giver point — man bør således aflevere sin besvarelse *under alle omstændigheder*).

Opgavesættet er berammet til at kunne løses med mindst 50% korrekt besvarelse med 16 timers koncentreret arbejdsindsats af enhver studerende, som har opnået mindst 4,5 point i kursets obligatoriske opgaver.

Hvis der opstår tvivl om selvstændighed eller kvalitet i besvarelsen, kan studerende blive indkaldt til en supplerende mundtlig eksamen fredag 7. november 2008. Studerende vil blive indkaldt til mundtlig eksamen per brev til deres KU-konti torsdag 6. november 2008 kl. 12.00, hvorfor alle studerende bedes efterse deres konti på denne dato. Udeblivelse fra mundtlig eksamen vil resultere i karakteren "ikke-bestået" eller, hvis der er tale om indkaldelse grundet mistanke om eksamenssnyd, i indberettelse af den studerende til dekanen.

I tilfælde af uklarheder i opgaveteksten er det op til eksaminanden selv at specificere løsningens forudsætninger; se dog afsnit 3 nedenfor.

1 Selvstændighed i besvarelser og eksamenssnyd

Opgavesættet skal besvares *individuelt*. Det er tilladt at diskutere opgave*formuleringerne* med andre studerende eller udenforstående, herunder at stille opklarende spørgsmål til opgavernes fortolkning. De studerende opfordres kraftigt til at stille sådanne spørgsmål på kursets ISIS-forum, så andre studerende kan drage nytte af svarene.

Det er *ikke* tilladt at diskutere *besvarelse* af opgaverne med andre personer, herunder løsningsmetoder, algoritmer eller konkret programtekst. Hvis i tvivl: Man må diskutere, *hvad* de efterspurgte SML-funktioner skal beregne, men ikke *hvordan* de skal beregne.

Specifikt er følgende *ikke* tilladt i eksamensperioden, og enhver overtrædelse vil resultere i indkaldelse til mundtlig overhøring samt overdragelse af sagen til dekanen for Det Naturvidenskabelige Fakultet til behandling under gældende regler for eksamenssnyd:

- At vise enhver del af sin besvarelse til andre, herunder specielt personer, som følger kurset.

- At diskutere eller afskrive (dele af) *besvarelse* af opgaver fra eksamenssættet med andre.
- At vise enhver del af opgavesættet til personer, som ikke er tilknyttet kurset. Herunder at lægge (dele af) opgaveformuleringer online (fora og chatrooms inklusive) andetsteds end kursets ISIS-forum.

Det indskærpes, at alle besvarelser vil blive underlagt både elektronisk og menneskelig plagiatkontrol.

2 Aflevering

Besvarelsen skal afleveres elektronisk via ISIS senest onsdag 29. oktober kl. 9.00 efter følgende procedure:

På kursets ISIS-hjemmeside findes menupunktet "Eksamen", hvorunder et opgavepunkt med titlen "Aflevering af eksamen" forefindes. Under dette punkt skal besvarelsen af eksamenssættet uploades efter samme procedure som aflevering af de obligatoriske opgaver på kurset.

I tilfælde af, og *kun* i tilfælde af, at ISIS ikke fungerer i tidsrummet 29. oktober, kl.7.00–9.00, kan besvarelser sendes til `simonsen@diku.dk`; der vil umiddelbart efter opgavens modtagelse på denne mailadresse blive sendt et tidsstemplet svar tilbage til den adresse, som opgaven er afsendt fra.

I tilfælde af, og *kun* i tilfælde af, at hverken ISIS eller DIKUs mailsystem fungerer 29. oktober i tidsrummet 7.00–9.00, kan besvarelsen gemmes på blivende medium (f.eks. en USB-nøgle) og overdrages til den kursusansvarlige frem til 29. oktober kl. 11.00 i DIKUs lokaler på Universitetsparken 1.

Det er den studerendes eget ansvar at gøre sig bekendt med, om ISIS, hhv. DIKUs mailsystem fungerer på afleveringstidspunktet[1]. Der kan forekomme mild overbelastning, hvis mange forsøger at aflevere eksakt samtidigt—den studerende opfordres derfor til at uploade sin besvarelse i så god tid som muligt.

Alle opgaverne skal afleveres i én fil navngivet "efternavn.fornavne.sml". Hedder man f.eks. "Jakob Grue Simonsen", skal filen således navngives "Simonsen.JakobGrue.sml". Bemærk, at man således skal angive sit *fulde* navn.

Filens indhold skal kunne afvikles under MosML 2.01 på DIKUs system vha. kommandoen `mosml -P full`.

Det er afgørende, at filens indhold kan afvikles, og at funktioner i filen har de navne og typer, der er specificeret i opgaveteksten; i modsat fald kan man risikere, at hele eksamensbesvarelsen vil blive betragtet som ukorrekt. Hvis man har en delvis løsning til en delopgave, som ikke kan afvikles, skal denne indsættes i SML-kommentarer `(* ... *)`.

I opgaver, hvor man bliver bedt om at skrive tekst (f.eks. forklaringer), der ikke kan afvikles under MosML, må denne tekst tillige indsættes i SML-kommentarer.

Alle funktioner i besvarelsen forventes kommenteret i henhold til god kommentarskik, se evt. FP-2 afsnit 5.3.3, og al programtekst skal opstilles pænt med passende indrykning.

Modsat de obligatoriske opgaver kræves der *ikke* afprøvning af SML-funktionerne, som udarbejdes i forbindelse med eksamensbesvarelsen. Eksaminanden opfordres dog kraftigt til at afprøve sine funktioner for at sikre korrekte besvarelser.

Bemærk, at det er muligt for studerende at uploade mere end én fil. Den *senest rettidigt afleverede* fil—og kun den—vil blive anset for den endelige eksamensbesvarelse.

[1] Det bemærkes, at ISIS stedse har fungeret upåklageligt i alle eksamensperioder.

3 Støtte i eksamensperioden

De studerende skyndes til at benytte kursets diskussionsforum på ISIS til spørgsmål eller diskussion af uklarheder i opgaveteksten samt til spørgsmål om formalia i forbindelse med eksamen. Kursets forelæsere og instruktorer vil læse og besvare spørgsmål på forummet i tidsrummet 9.00–22.00, både mandag 27. oktober og tirsdag 28. oktober 2008.

Spørgsmål af administrativ eller procedural karakter kan tillige rettes personligt til den kursusansvarlige, Jakob Grue Simonsen, tlf. 35 32 14 39, e-mail: simonsen@diku.dk gennem hele eksamensperioden.

Der kan tillige rettes personlig henvendelse til den kursusansvarlige, lokale 24.5.46 i Københavns Universitets lokaler på Amager: Njalsgade 126, bygning 24, 5. sal i perioden mandag 27. oktober 14.00-15.30, hhv. tirsdag 28. oktober 10.00-11.30.

De studerende opfordres til at stille spørgsmål, hvis svar kunne have interesse for andre studerende, på kursets ISIS-forum.

4 Eksamensopgaver

Opgave 1 Ved en *forlængelse* af en liste `[`x_1`, ..., `x_n`]` vil vi i denne opgave forstå en ny liste af form `[SOME` x_1`, ..., SOME` x_n`, `$\underbrace{\texttt{NONE, ..., NONE}}_{m}$`]` for et helt tal $m \geq 0$.

Konstruer en SML-funktion `udfyld : 'a list list -> 'a option list list`, der for en liste `[`xr_1`, ..., `xr_k`]` af lister forlænger (om nødvendigt) de k indgående lister, så de alle bliver lige lange. Listerne må ikke forlænges mere end nødvendigt. Funktionens værdi for den tomme liste af lister er uden betydning.

Eksempel: `udfyld [["a", "b"], ["c", "d", "e"], ["f"]]` skal returnere
`[[SOME "a", SOME "b", NONE], [SOME "c", SOME "d", SOME "e"], [SOME "f", NONE, NONE]]`.

Opgave 2 I en liste siges en værdi x at danne *et løb af længde* n (for et positivt helt tal n), hvis listen indeholder mindst en ubrudt delsekvens af mindst n på hinanden følgende forekomster af x.

Konstruer en SML-funktion `loeb : ''a list * int -> ''a option`, så `loeb (`xr`,`n`)` returnerer `SOME` x, hvis xr indeholder et løb af x'er af længde n, og `NONE`, hvis xr ikke indeholder noget løb af længde n. Hvis flere forskellige elementer danner løb af længde n, skal funktionen blot returnere `SOME` y for et af disse elementer y.

Eksempler: `loeb ([1,2,1],2)` skal returnere `NONE`; `loeb ([3,4,4,3,3,3],2)` kan returnere `SOME 3` eller `SOME 4`.

Opgave 3 Et *spil* gennemløber et antal *tilstande*, idet der i hver tilstand (pånær sluttilstande) er nogle mulige valg af *træk*, og et valgt træk fører fra en tilstand i spillet til en ny tilstand.

Under forudsætning af, at der allerede foreligger typer til beskrivelse af mængden af tilstande og mængden af mulige træk, kan de mulige udviklinger i et spil derfor beskrives i SML af et træ fra følgende datastruktur:

```
datatype ('tilst,'traek) spiltrae
         = T of 'tilst * ('traek * ('tilst,'traek) spiltrae) list;
```

Intentionen er, at træet `T (`t`,[(`d_1`,`T_1`), ..., (`d_n`,`T_n`)])` skal illustrere, at der i tilstanden t kan foretages de n lovlige træk $d_1, \ldots, d_n$, og at et valg af trækket d_1 vil føre til spiltræet T_1 (der igen har en tilstand og en række mulige videreudviklinger), og så videre, trækket d_n ville føre fra t til spiltræet T_n.

1. Konstruer en SML-funktion

```
danSpiltrae : ('tilst -> 'traek list) * ('tilst * 'traek -> 'tilst)
  -> 'tilst -> ('tilst,'traek) spiltrae
```

   sådan at hvis `lovlige` t danner listen af lovlige træk i spillet i tilstanden t (for sluttilstande en tom liste) og `naeste (`t`,`d`)` danner den nye tilstand, hvis man fra t vælger trækket d, så vil `danSpiltrae (lovlige,naeste)` t illustrere de mulige udviklinger i spillet fra udgangstilstanden t.

   Eksempel: I NIM skiftes to spillere til at fjerne tændstikker fra en række tændstikbunker. I hvert træk skal man vælge en bunke og fjerne mindst én tændstik fra den, eller to, eller tre, og så videre, men højst hele bunken (og altså kun tændstikker fra samme bunke). En tilstand kan repræsenteres af en heltalsliste (antallet af tændstikker i de forskellige bunker) og et træk af et talpar til angivelse af bunken og antallet af tændstikker, det er valgt at fjerne. Bunkerne antages at være nummereret fra 0.

2. Skriv funktionerne `lovlige` og `naeste` for NIM, og afprøv den konstruerede funktion `danSpiltrae` på dette eksempel.

Opgave 4 Lad a og b være heltal, hvor $a \leq b$. Vi bruger notationen $a..b$ til at betegne mængden af heltal $\{a, a+1, \ldots, b\}$. F.eks. er $3..7 = \{3,4,5,6,7\}$ og $5..5 = \{5\}$. En matematisk funktion f fra heltal til heltal siges at være *voksende* på intervallet $a..b$, hvis (og kun hvis) det for alle $x, y \in a..b$ gælder at $x < y \Rightarrow f(x) \leq f(y)$. Den siges at være *strengt voksende* hvis (og kun hvis) $x < y \Rightarrow f(x) < f(y)$. (En strengt voksende funktion er dermed også voksende. Det modsatte behøver ikke at være tilfældet.)

Skriv en SML-funktion `vokstest : int * int -> (int -> int) -> bool * bool`, så kaldet `vokstest (`a`,`b`)` f returnerer et par af boolske værdier `(`v`,` sv`)`, der angiver om f er henholdsvis voksende og strengt voksende på intervallet $a..b$. `vokstest` kan antage at $a \leq b$, og at f er defineret uden undtagelser på alle heltal i intervallet $a..b$, men ikke nødvendigvis uden for dette.

Eksempelvis skal kaldet `vokstest (1,3) (fn x => x*x)` returnere `(true,true)`, mens kaldet `vokstest (1,3) (fn x => x div 4` skal returnere `(true,false)`.

Opgave 5 Lad der være givet 4 heltal a_0, a_1, b_0 og b_1. Vi kan så danne en uendelig følge af heltal $(s_0, s_1, s_2, \ldots)$ ved følgende rekursionsligninger:

$$\begin{aligned} s_0 &= a_0 \\ s_1 &= a_1 \\ s_{i+2} &= b_0 \cdot s_i + b_1 \cdot s_{i+1} \quad (\text{for } i \geq 0) \end{aligned}$$

Hvis vi f.eks. sætter $a_0 = 0$, $a_1 = 1$ og $b_0 = b_1 = 1$, får vi den velkendte Fibonacci-følge $(0, 1, 1, 2, 3, 5, 8, \ldots)$, hvor hvert tal i følgen (på nær de to første) er summen af de to foregående.

Skriv en funktion `reklign : int * int -> int * int -> int -> int`, så kaldet

`reklign (`a_0`,` a_1`) (`b_0`,` b_1`)` n (hvor $n \geq 0$)

returnerer s_n. Eksempelvis skal kaldet `reklign (0,1) (1,1) 4` returnere værdien `3` (idet `3` er element nr. 4 i Fibonacci-følgen, når nummereringen starter fra 0).

Køretiden skal være $O(n)$; specielt bør s_{42} for Fibonacci-følgen kunne beregnes på væsentligt under et sekund.

(De næste to opgaver refererer begge til følgende scenarie)

I FP Banks kundesystem opereres der med en abstrakt type af indskudskonti, beskrevet ved følgende signatur:

```
signature KONTO =
sig
  type konto
  val aabn : int -> konto
  val trans : konto * int -> konto option
  val saldo : konto -> int
  val tilskriv : konto -> konto
end;
```

De enkelte operationer har følgende betydninger:

- `aabn` n opretter en ny konto med den angivne startsaldo. Hvis denne er negativ, kastes undtagelsen `Domain`.
- `trans (`k`,` n`)` repræsenterer transaktionen, der indsætter n kroner på kontoen k, hvis $n \geq 0$. Hævning repræsenteres som indsætning af et negativt beløb. Hvis transaktionen går godt, returneres `SOME` k', hvor k' er den resulterende kontotilstand; hvis transaktionen bliver afvist (hvilket sker hvis man prøver at hæve et større beløb end den aktuelle saldo), returneres `NONE`.
- `saldo` k returnerer den aktuelle saldo (som altid vil være ikke-negativ) for kontoen k.
- `tilskriv` k tilskriver periodiske renter for kontoen k (ud fra kontospecifikke regler, beskrevet nedenfor) og returnerer den resulterende kontotilstand k'.

Opgave 6 Banken tilbyder i øjeblikket to slags konti, der kun adskiller sig ved rentereglerne:

GribKonto. Ved hver tilskrivning indsættes 2% (afskåret *nedad* til et helt kronebeløb) af den *mindste* saldo siden sidste tilskrivning (eller kontoens åbning, hvis der ikke tidligere er tilskrevet renter).

HajKonto. Ved hver tilskrivning indsættes 4% (igen afskåret nedad) af saldoen *umiddelbart* efter sidste tilskrivning (eller kontoens åbning), forudsat at der i perioden ikke er blevet hævet fra kontoen. Er der hævet på kontoen, tilskrives 0 kr. for hele perioden.

Skriv to strukturer `GribKonto :> KONTO` og `HajKonto :> KONTO`, der implementerer ovenstående regler.

Opgave 7 FP Bank tilbyder også, at man kan binde to vilkårlige konti sammen i en *kombinationskonto*, sådan at transaktioner så vidt muligt benytter den ene (kaldet *lønkontoen* i det følgende), men at indestående ud over en specificeret maksimumssaldo (max) periodisk overføres til den anden (*opsparingskontoen*). Kombinationskontoen beskrives også ved en struktur med signatur `KONTO`, hvor de fire kontooperationer nu fortolkes som følger:

- Ved åbning oprettes begge konti. Hele indskudsbeløbet bruges til at åbne lønkontoen, mens opsparingskontoen starter tom.
- Ved indskudstransaktioner indsættes hele beløbet på lønkontoen. Ved hævning tages beløbet så vidt muligt fra lønkontoen, men hvis der ikke er fuld dækning på denne, tages det manglende fra opsparingskontoen. Hvis den samlede saldo heller ikke er tilstrækkelig, afvises transaktionen, og der hæves ikke noget fra hverken den ene eller den anden konto.

- Saldoen for kombinationskontoen er den samlede saldo for de to konti.
- Rentetilskrivning sker på de to konti ud fra hver deres regler. Hvis saldoen på lønkontoen herefter overstiger max, hæves det overskydende beløb fra lønkontoen og indsættes på opsparingskontoen.

Skriv en funktor

```
functor KombiKonto (structure LK : KONTO
                    structure OK : KONTO
                    val max : int) :> KONTO = ...
```

der implementerer ovenstående regler, sådan at man herefter f.eks. kan skrive:

```
structure GHKonto = KombiKonto (structure LK = GribKonto
                                structure OK = HajKonto
                                val max = 1000);
```

De to konti `LK` og `OK` kan godt vælges ens (f.eks. begge som `GribKonto`), og behøver ikke at være valgt blandt de to kontoformer fra foregående opgave.

Opgave 8 Skriv en funktion `composelist : ('a -> 'a) list -> 'a -> 'a`, således, at hvis `[`f_1`, ..., `f_n`]` er en liste af funktioner af typen `'a -> 'a` og x er en værdi af typen `'a`, så returnerer kaldet `composelist [`f_1`, ..., `f_n`]` x værdien $f_1(f_2(\cdots f_{n-1}(f_n(x))))$. I specialtilfældet, hvor $n = 0$ (dvs. listen af funktioner er tom), skal `composelist []` x returnere x.

Eksempelvis skal kaldet

```
composelist [fn y => y*y, fn z => z+1, fn w => w div 2] 4
```

resultere i værdien `9` ($= ((4$ `div` $2) + 1) * ((4$ `div` $2) + 1)$).

Opgave 9 Et *ord* er en værdi af type `string`, som udelukkende indeholder store eller små bogstaver i det latinske alfabet (A–Z, a–z). Således ses bort fra danske bogstaver og øvrige specialtegn i denne opgave. To ord er *ens*, hvis de består af samme bogstaver i samme rækkefølge, uagtet om der bruges store bogstaver eller ej. Dermed er ordene "Alfabet" og "alfabet" ens, mens ordene "Alfabet" og "Alphabet" ikke er ens.

En værdi `st` af typen `string` består af en eller flere *linjer*, defineret på oplagt måde efter tilstedeværelsen af specialtegnet `\n`. Tekstens første linje nummereres "1", og de følgende linjer fortløbende. Således består teksten

```
"See the dog run.\nSee Spot run.\n Run, Spot, run!\n "
```

af linjerne

```
"See the dog run."
"See Spot run."
" Run, Spot, run!"
" "
```

med linjenumrene 1, 2, 3, og 4, henholdsvis.

Ved et *index* for en værdi `st` af typen `string` forstås i denne opgave en liste af par `(t,xs)`, hvor `t` er et ord (skrevet med småt) og `xs` er en liste af numre på de linjer i teksten `st`, i hvilke ordet `t` forekommer.

Eksempelvis er følgende indexet for ovenstående tekst:

```
[("dog",[1]),("run",[1,2,3]),("see",[1,2]),("spot",[2,3]),("the",[1])]
```

Indexet skal være sorteret i alfabetisk orden efter `t`, og for hvert ord `t` forventes listen `xs` i det tilhørende par `(t,xs)` at være uden dubletter og sorteret i stigende nummerorden.

Konstruer en funktion `index : string -> (string * int list) list` som givet en tekst `st` returnerer indexet for `st`.

Opgave 10 Betragt følgende erklæringer:

```
exception Empty;

fun swapscure ([x] : int list) = []
  | swapscure (x :: y :: xr)   = if x < y then x :: swapscure(y :: xr)
                                 else y :: swapscure(x :: xr)
  | swapscure _                = raise Empty;

fun obscure1 [x] = x
  | obscure1 (xr as _ :: _ :: _) = obscure1(swapscure xr)
  | obscure1 _                   = raise Empty;
```

1. Forklar kort, hvad kaldet `obscure1 xs` beregner, når `xs` er en ikke-tom liste.

   (Vink: Prøv evt. at udføre kaldene `swapscure [5,2,6,7,1]`, `swapscure (swapscure [5,2,6,7,1])`, samt `obscure1 [5,2,6,7,1]`.)

2. Angiv køretidskompleksiteten for `obscure1 xs` som funktion af længden, n, af listen `xs`. Svaret ønskes angivet i O- eller Θ-notation. Argumentér kort for svaret.

   (Vink: Find først køretidskompleksisteten for `swapscure xs` som funktion af længden af listen `xs`. Overvej dernæst hvor lang den liste er, som `swapscure xs` returnerer, i forhold til længden af `xs`.)

3. Skriv en funktion `obscure2 : int list -> int`, som beregner det samme som `obscure1`, men således at `obscure2 xs` har køretidskompleksitet $O(n)$, hvor n er længden af listen `xs`. Argumentér kort for køretidskompleksiteten.

(Opgavesættet slut)