Ordinær eksamen i Introduktion til programmering, blok 1 2009

Nærværende dokument udgør opgavesættet for den ordinære eksamen i kurset "Introduktion til programmering", blok 1 2009.

Dokumentet offentliggøres onsdag 28. oktober kl. 9.00 via KUs kursusadministrationssystem Absalon. Besvarelsen skal afleveres senest fredag 30. oktober kl. 9.00 — se afsnit 2 nedenfor.

Besvarelsen bedømmes efter 7-trinsskalaen. Eksamensresultaterne vil findes på Absalon senest tre uger efter eksamens afslutning, og vil findes i det Naturvidenskabelige Fakultets eksamensprotokol snart 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, god programmeringsstil, og generel læsbarhed.

Opgavernes rækkefølge i sættet er uafhængig af deres sværhedsgrad. Bemærk, at delvist færdige løsninger til enkeltopgaver giver point; endvidere fordrer deltagelse i evt. reeksamen, at man har deltaget i den ordinære eksamen — man bør således aflevere sin besvarelse *under alle omstændigheder* (eller, i værste fald, aflevere blankt).

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

Hvis der opstår tvivl om selvstændighed i besvarelsen, kan studerende blive indkaldt til en supplerende mundtlig eksamen fredag 13. november 2009. Studerende vil blive indkaldt til mundtlig eksamen per e-mail til deres KU-konti torsdag 12. november 2009 kl. 12.00, hvorfor alle studerende bedes efterse deres konti på denne dato. Udeblivelse fra mundtlig eksamen vil resultere 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 Dikutal-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 det.

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 Dikutal-forum.

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

2 Aflevering

Besvarelsen skal afleveres elektronisk via Absalon senest fredag 30. oktober 2009 kl. 9.00 efter følgende procedure:

På kursets Absalon-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 Absalon ikke fungerer i tidsrummet 30. 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 Absalon eller DIKUs mailsystem fungerer 30. 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 30. oktober kl. 11.00 i DIKUs lokaler på Universitetsparken 1.

Det er den studerendes eget ansvar at gøre sig bekendt med, om Absalon, 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.

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. IP-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.

3 Støtte i eksamensperioden

De studerende skyndes til at benytte kursets diskussionsforum på Dikutal til spørgsmål eller diskussion af uklarheder i opgaveteksten samt til spørgsmål om formalia i forbindelse med eksamen.

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

Kursets forelæsere og instruktorer vil læse og besvare spørgsmål på forummet i tidsrummet 9.00–22.00, både onsdag 28. oktober og torsdag 29. oktober 2009.

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 onsdag 28. oktober 14.00-15.30, hhv. torsdag 29. oktober 10.00-11.30.

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

4 Eksamensopgaver

Opgave 1 Erklær en funktion `grupper : int -> 'a list -> 'a list list`, således at et kald `grupper n xs` returnerer listen af lister der fås ved at danne lister af `n` af elementerne i `xs` ad gangen (indtil der ikke er flere elementer i listen).

Hvis `n` er større end længden af `xs` skal listen `[xs]` returneres.

Eksempler:

Kaldet `grupper 2 [3, 1, 4, 1, 5, 9]` skal returnere `[[3, 1], [4, 1], [5, 9]]`.

Kaldet `grupper 4 [3, 1, 4, 1, 5, 9]` skal returnere `[[3, 1, 4, 1], [5, 9]]`

Kaldet `grupper 42 [3, 1, 4, 1, 5, 9]` skal returnere `[[3, 1, 4, 1, 5, 9]]`.

Opgave 2 En *permutation* af en værdi, `v`, af typen `string` er en anden værdi, `w`, af typen `string`, således at hvert tegn i `v` forekommer præcis en gang i `w` for hver gang det forekommer i `v`.

Eksempelvis er teksten `"nørd"` en permutation af teksten `"drøn"`, og teksten `"nebelwerfer"` er en permutation af teksten `"lebenferwer"`. Teksten `"drøøn"` er *ikke* en permutation af teksten `"nørd"`.

Erklær en funktion `permtkst : string * string -> bool`, således at et kald `permtkst(s,t)` returnerer `true` netop hvis `t` er en permutation af `s`.

Opgave 3 Betragt følgende tre funktionserklæringer:

```
fun findlige []         = []
  | findlige (x :: xs)  = if x mod 2 = 0 then x :: findlige xs else findlige xs

fun sumlige []        = 0
  | sumlige (x :: xs) = if x mod 2 = 0 then x div 2 + sumlige xs else sumlige xs

fun collatz []        = []
  | collatz (x :: xs) = if x mod 2 = 0 then x div 2 :: collatz xs
                                       else 3*x + 1 :: collatz xs
```

Skriv de tre ovenstående erklæringer om, så de benytter en eller flere passende *polymorfe højereordensfunktioner*.

Du må enten selv erklære de polymorfe højereordensfunktioner (evt. i en lokal erklæring), eller benytte standardfunktionerne fra kurset.

Opgave 4 Betragt følgende datatypeerklæring:

```
datatype bit = O | Z
```

Vi ønsker nu at betragte lister af værdier af typen `bit`. F.eks. er `[O,O,Z]` og `[Z,Z,O,O,Z]` sådanne lister.

Erklær en funktion `allbitlists : int -> bit list list`, således at hvis `n` er en værdi af typen `int`, således at `n` ≥ 0, da returnerer kaldet `allbitlists n` en liste, som indeholder alle værdier af typen `bit list`, som har længde præcis `n`.

F.eks. skal kaldet `allbitlists 0` returnere værdien `[]`, mens kaldet `allbitlists 2` kan returnere værdien `[[Z,Z], [Z,O], [O,Z], [O,O]]`.

Det anses for korrekt, hvis din funktion returnerer en liste, som har elementerne i en anden rækkefølge end ovenstående.

Vink: Hvis du har listen af alle værdier af type `bit list` af længde $n-1$, hvordan fås så den tilsvarende liste af værdier af type `bit list` af længde n?

Opgave 5 Vigenère-koden er en klassisk metode til at kryptere data. Metoden er en lidt vanskeligere variation af den velkendte Cæsar-kode. Denne opgave handler om at kryptere heltal med (en variant af) Vigenère-koden.

I (vores variant af) Vigenère-koden vælges en (kort, ikke-tom) liste `n` af heltal ("nøglen"); for at kryptere heltallet `k` lægger man tallene fra `n` til hvert ciffer i `k` modulo 10, *idet man starter fra det mest betydende ciffer*—det ciffer, som står længst til venstre i den almindelige måde at skrive tal op på. Hvis man løber tør for tal i `n`, begynder man forfra.

Hvis `n` for eksempel er listen `[4,25,2]` og `k` er `54153`, da bliver cifrene i det krypterede tal som følger:

$((5+4) \bmod 10)$ $((4+25) \bmod 10)$ $((1+2) \bmod 10)$ $((5+4) \bmod 10)$ $((3+25) \bmod 10)$

Og det krypterede tal bliver derfor `99398`.

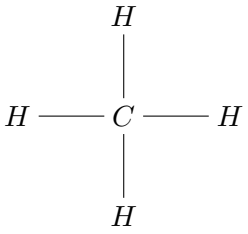
Erklær en funktion `vig : int list -> int -> int`, således at et kald `vig n k` returnerer det heltal, som fås ved at kryptere heltallet `k` med nøglen `n`. Nøglen `n` kan antages at være en ikke-tom liste, og der kan gennem hele opgaven ses bort fra problemer med evt. foranstillede nuller. Endvidere kan kan der i opgaven ses bort fra kryptering af negative tal.

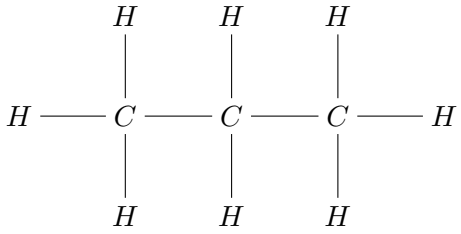
Opgave 6 En *alkan* er en kemisk forbindelse, som består af carbon- og hydrogenatomer, hvori der kun optræder enkeltbindinger.

En alkan skal opfylde følgende:

- Hver alkan skal indeholde mindst et carbonatom.
- Hvert carbonatom har netop fire bindinger, og hvert hydrogenatom netop en binding.
- Hvert carbonatom *skal* være forbundet til *præcis* fire andre atomer, og hvert hydrogenatom *skal* være forbundet til *præcis* et andet atom.
- Carbonatomer kan være forbundet til andre carbonatomer og til hydrogenatomer. Hydrogenatomer kan kun være forbundet til carbonatomer; således kan to hydrogenatomer ikke være forbundet til hinanden.
- Alkanen skal være sammenhængende og må ikke indeholde kredse.

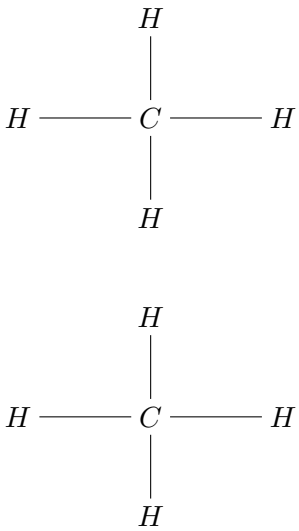
Eksempler på alkaner (for de interesserede: methan til venstre, propan til højre):



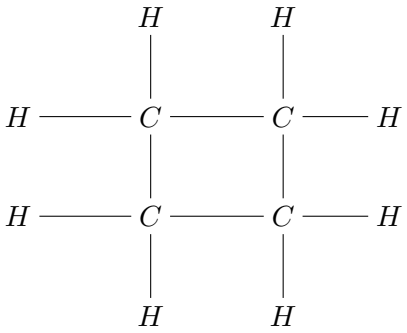


Eksempler på forbindelser, som *ikke* er alkaner (til venstre: *Ikke*-sammenhængende. Til højre: Indeholder en kreds).

Eks. 1



Eks. 2



Bemærk, at i eksemplet til venstre ovenfor forekommer der to alkaner—men de udgør ikke *tilsammen* en alkan.

Erklær en datatype `alkan` til at repræsentere alkaner; datatypen *skal* indeholde en værdikonstruktør, som repræsenterer et carbonatom. Det er i orden, hvis den samme alkan kan repræsenteres af flere forskellige værdier af datatypen. Til gengæld må det kun være muligt i datatypen at erklære værdier, som repræsenterer en alkane.

Vink I: Kravet om, at en alkan skal være sammenhængende og ikke må indeholde kredse vil automatisk være overholdt, medmindre du kaster dig ud i en meget avanceret løsning.

Vink II: Tænk på et carbonatom som om, det har fire "børn".

Erklær dernæst en funktion `atomantal : alkan -> int * int`, således at et kald `atomantal a` returnerer parret `(c,h)`, hvor `c` og `h` er antallet af carbonatomer, hhv. hydrogenatomer i alkanen `a`.

For methan og propan i det ovenstående, skal `atomantal a` altså returnere henholdsvis `(1,4)` og `(3,8)`.

Opgave 7 Kald en fil *velformet*, hvis den indeholder navne på folk og deres respektive e-mail-adresser som følger:

- En e-mail-adresse indeholder altid netop et snabel-a (`#"@"`).
- E-mail-adresser kan ikke indeholde nogle blandt tegnene `#"\n", #"\r", #" ", #"\t"`, men kan indeholde ethvert andet tegn.
- Navne kan ikke indeholde nogle blandt tegnene `#"\n", #"\r", #"\@"`
- En linje består af en endelig sekvens af tegn på formen "s`\n`", hvor s er en sekvens af tegn, som ikke indeholder tegnet `#"\n"`.
- E-Mail-adresser står altid sidst på en linje.
- Navne og e-mail-adresser er altid adskilt af mellemrum.

Der kan være et vilkårligt stort antal linjer i filen.
F.eks. kan følgende være indholdet af en velformet fil kaldet `a.txt`:

```
Jakob Grue Simonsen simonsen@diku.dk
Ebenezer Scrooge moralturpitude@hotmail.com
Silver, Long John warshiplover54@gmail.com
```

(Hvor vi i ovenstående har fremvist tegnet `#"\n"` ved at skifte linje).
Erklær en funktion `adresser : string -> string list`, således at hvis `f` er en værdi af typen `string`, da returnerer et kald `adresser f` listen af alle e-mailadresser, som forekommer i filen med navn `f`. *Filen kan antages velformet.*
F.eks. skal kaldet `adresser "a.txt"` returnere listen
`["simonsen@diku.dk", "moralturpitude@hotmail.com", "warshiplover54@gmail.com"]`
Erklær dernæst en funktion `dommail : string -> string -> string list`, således at et kald `dommail d f` returnerer en liste af alle e-mail-adresser fra den velformede fil med navn `f`, som indeholder teksten `d` i den del af adressen, som forekommer efter `@`.
F.eks. skal kaldet `dommail "diku" "a.txt"` returnere listen `["simonsen@diku.dk"]`.
Kaldet `dommail "" "a.txt"` skal returnere listen
`["simonsen@diku.dk", "moralturpitude@hotmail.com", "warshiplover54@gmail.com"]`.

Opgave 8 Et *snit* af en liste `xs` er et par `(ys,zs)` af lister, således at `ys @ zs = xs`. Snittet siges at være *ægte*, hvis hverken `ys` eller `zs` er den tomme liste. I snittet `(ys,zs)` siges `ys` at være snittets *forende* og `zs` at være snittets *bagende*.
Erklær en funktion `snit : 'a list -> ('a list * 'a list) list`, således at et kald `snit xs` returnerer en liste, som indeholder alle ægte snit af `xs`.
F.eks. kan kaldet `snit [4,7,1]` returnere listen `[([4],[7,1]), ([4,7],[1])]`.
Kaldet `snit [9,34]` skal returnere listen `[([9],[34])]`
Det anses for korrekt, hvis din funktion returnerer en liste med elementerne i en anden rækkefølge end ovenstående.

Erklær dernæst en funktion `kombisnit : 'a list -> 'a list -> 'a list list`, således at et kald `kombisnit xs ys` returnerer en liste, som består af alle de mulige lister, som kan fås ved at sammenhægte en forende fra `xs` med en bagende fra `ys`.
F.eks. kan kaldet `kombisnit [4,7,1] [9,34]` returnere listen
`[[4,34], [4,7,34]]`.

Kaldet `kombisnit [9,34] [4,7,1]` kan returnere listen `[[9,7,1], [9,1]]`.

Det anses for korrekt, hvis din funktion returnerer en liste med elementerne i en anden rækkefølge end ovenstående.

Opgave 9 For længe siden i en fjern galakse ønskede et større containertransportfirma at sende det gode rumskib "SS Totalentreprise" til en række destinationer uden for det lokale solsystem.

Ledelsen i firmaet var bekymret over, at de selv måske ville ældes i et andet tempo end rumskibets besætning, når rumskibet foretog rejsen fra punkt A til punkt B. Ledelsen var ligeglad med detaljer som acceleration, energi, henførselssystemer og den slags: Man ville have et enkelt svar, og man ville have det med det samme.

Ved overenskomst med besætningens fagforening blev man enige om at måle afstanden, L, tilbagelagt af SS Totalentreprise, og skibets fart, v, set fra besætningens synspunkt.

Man hyrede tre konsulenter: Dr. Newton, Dr. Einstein og Dr. Skywalker, hver med sit syn på sagen.

- Dr. Newton sagde: "Hvis SS Totalentreprise tilbagelægger afstand L med fart v, og lysets hastighed er c, så går der L/v tid, både for ledelsen og for besætningen. Og jeg vil blæse på lysets hastighed"
- Dr. Einstein sagde: "Hvis SS Totalentreprise tilbagelægger afstand L, med fart v, og lysets hastighed er c, så går der L/v tid for besætningen, men

  $$\frac{L}{v \cdot \sqrt{1.0 - \frac{v^2}{c^2}}}$$

  tid for ledelsen. I øvrigt er lysets hastighed 300000 kilometer per sekund."
- Dr. Skywalker sagde: "Hvis SS Totalentreprise tilbagelægger afstand L, med fart v, og lysets hastighed er c, da går der L/c^v tid for besætningen og L/v tid for ledelsen, men lysets hastighed er kun 6 kilometer per sekund."

Ledelsen var forvirret, og kunne i øvrigt kun finde ud af heltal. Man bad derfor de tre konsulenter om at runde alle de fundne *tider* ned til nærmeste heltal (ved hjælp af `MLs floor`-funktion). Konsulenterne skulle dog regne så præcist som muligt (ved hjælp af værdier af typen `real`), indtil tiderne blev fundet.

Man besluttede sig for at erklære følgende signatur:

```
signature Totalentreprise =
sig
  val besaetningstid : int -> int -> int
  val ledelsestid    : int -> int -> int
end
```

De to funktioner `ledelsestid` og `besaetningstid` tager *først* afstanden L (i kilometer), *dernæst* hastigheden (i kilometer per sekund) som argument, og returnerer hhv. tiden for ledelsen og tiden for besætningen målt i sekunder.

Erklær tre strukturer: `Newton :> Totalentreprise`,

`Einstein :> Totalentreprise` og `Skywalker :> Totalentreprise`, som matcher signaturen `Totalentreprise` og som implementerer de tre konsulenters syn på sagen.

Hvis f.eks. $L = 1000000000$ og $v = 260000$, da skal kaldene `besaetningstid L v` og `ledelsestid L v` returnere følgende:

- (Newton) `3846` hhv. `3846`
- (Einstein) `3846` hhv. `7709`
- (Skywalker) `0` hhv. `3846`

For at opnå fuldt pointtal skal følgende krav være overholdt: Hvis L og v begge er positive heltal, som kan repræsenteres i `ML`, da skal `besaetningstid L v` i alle tre strukturer returnere en værdi af typen `int` (der skal altså ikke kastes undtagelser, som ikke bliver fanget behørigt).

Opgave 10 I denne opgave benytter vi igen følgende datatype:

```
datatype bit = O | Z;
```

Betragt en liste `xs` af værdier af typen `bit`; antag, at `xs` har længde n, at antallet af `O`er i listen er n_O, og at antallet af `Z`er i listen er n_Z. Da siges `xs` at være *1-tilfældig*, hvis $|n_O/n - 1/2| < 1/\lfloor\sqrt{n}\rfloor$ og $|n_Z/n - 1/2| < 1/\lfloor\sqrt{n}\rfloor$.

I ovenstående er / division af brudne tal (svarende til SMLs operator `/`), $|\cdots|$ er absolutværdi (svarende til SMLs funktion `abs`), og $\lfloor\cdots\rfloor$ runder ned til nærmeste heltal (svarende til SMLs funktion `floor`).

Erklær en funktion `alletilf : int -> int`, således at et kald `alletilf n` returnerer *antallet* af alle 1-tilfældige lister af længde n.

F.eks. skal kaldet `alletilf 4` returnere `14`, mens `alletilf 20` skal returnere `1005176`.

For at få fuldt pointtal for opgaven kræves, at `alletilf n` terminerer inden for ganske få sekunder, selv for "store" værdier af `n`, f.eks. `25`. Man må se bort fra returværdier, som er for store til at kunne repræsenteres af værdier af typen `int` i SML.

Vink: Regn først ud, hvor mange `O`'er, der *mindst* skal være, hhv. *højst* må være, for at en liste af længde `n` er 1-tilfældig.

(Opgavesættet slut)