Ordinær eksamen i Introduktion til programmering, blok 1, 2010

1. November 2010

Dette dokument udgør opgavesættet for den ordinære eksamen i kurset "Introduktion til programmering", blok 1, 2010. Det består af 13 nummererede sider.

Dokumentet offentliggøres mandag den 1. november kl. 9:00 via KU's kursusadministrationssystem Absalon. Besvarelsen skal afleveres senest onsdag den 3. november kl. 9:00 — se afsnit 2 nedenfor.

Besvarelsen bedømmes efter 7-trinsskalaen ud fra en samlet bedømmelse af, hvorvidt læringsmålene for kurset er opfyldt (se kursusbeskrivelsen).

Eksamensresultaterne vil findes på Absalon senest tre uger efter eksamens afslutning og vil findes i det Naturvidenskabelige Fakultets eksamensprotokol kort tid derefter.

Eksamenssættet består af 9 opgaver. For alle opgaver kræves sigende kommentarer (herunder begrundelser for eller forklaringer af løsninger i det omfang, det ikke allerede fremgår af koden) og variabelnavngivning, god programmeringsstil og generel læsbarhed, herunder ved passende indrykning.

Opgavernes rækkefølge i sættet er uafhængig af deres sværhedsgrad. Læs hele sættet grundigt igennem, før du begynder at programmere.

Bemærk, at delvist færdige løsninger til enkeltopgaver kan give point.

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 den 19. november 2010, kl. 13. Studerende vil blive indkaldt til mundtlig eksamen med elektronisk brev til deres KU-konti senest torsdag den 18. november 2010 kl. 12:00, hvorfor alle studerende skal efterse deres konto den 18. november 2010 om eftermiddagen. Udeblivelse fra mundtlig eksamen vil resultere i indberettelse til studienævnet.

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

1 Selvstændighed i besvarelsen 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 forum på `dikutal.dk`, så andre studerende kan drage nytte af svarene og alle eksaminander er ligestillet.

Det er *ikke* tilladt at diskutere *besvarelse* af opgaverne med andre personer, herunder afprøvningstilfælde, 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 studienævnet 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 eller hele *besvarelser* af opgaver fra eksamenssættet.
- 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.
- At efterlyse løsninger.
- At bruge i øvrigt tilladeligt skriftligt eller mundtligt materiale ud over kursets undervisningsmateriale uden henvisning til kilden (f.eks. oplysninger fra Wikipedia, Google Scholar eller lignende).

Brugen af *skriftligt* materiale fra offentligt tilgængelige kilder er *tilladt* under forudsætning af, at kilden angives i besvarelsen.

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

2 Aflevering

Besvarelsen skal afleveres elektronisk via Absalon senest onsdag den 3. november 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 3. november kl. 7:00–9:00, kan besvarelser sendes til `henglein@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. Eksaminander, som *ikke* er registreret som deltagere på IP-hjemmesiden på Absalon, skal ligeledes aflevere ved email til `henglein@diku.dk`.

I tilfælde af, og *kun* i tilfælde af, at *hverken* Absalon *eller* DIKUs elektroniske postsystem, fungerer 3. november i tidsrummet 7:00–9:00, skal besvarelsen gemmes på USB-nøgle og overdrages til Jette Møller, lokale 3-2-18 på DIKU, frem til den 3. november kl. 11:00. (USB-nøglen returneres i givet fald efter eksamensbedømmelsen.)

Det er den studerendes eget ansvar at gøre sig bekendt med, om Absalon, henholdsvis DIKUs postsystem 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 er et korrekt SML-program, der kan afvikles under Moscow ML 2.00 eller Moscow ML 2.01 ved hjælp af kommandoen `mosml -P full`.

[1] Det er værd at bemærke, at Absalon indtil nu har fungeret upåklageligt i alle eksamensperioder.

Det er tillige et krav, at funktioner i filen har *præcis de navne og typer*, der er specificeret i opgaveteksten, også hvad angår små og store bogstaver. 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 (for eksempel 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, og hver linje må være maksimum 80 tegn lang inklusive indrykning.

Eksaminanden opfordres kraftigt til at afprøve sine funktioner for at sikre korrekte besvarelser. Medmindre det er eksplicit forlangt i opgaven, behøves afprøvningen dog ikke inkluderes i besvarelsen.

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

Meddelelser fra kursets undervisere på kursets diskussionsforum på Dikutal (også tilgængelig fra kursets hjemmeside) gælder som supplerende oplysninger om eksamenen. Det er den enkelte studerendes ansvar at holde sig ajour med disse oplysninger i eksamensperioden.

Generelt tilskyndes eksaminanderne til at benytte kursets diskussionsforum til spørgsmål eller diskussion af uklarheder i opgaveteksten, samt til spørgsmål om formalia i forbindelse med eksamen.

Kursets undervisere vil både mandag den 1. november og tirsdag den 2. november 2010 læse og besvare spørgsmål på forummet i tidsrummet kl. 9:00–22:00.

Kun spørgsmål og meddelelser med fortroligt indhold rettes direkte til den kursusansvarlige, Fritz Henglein, tlf. 35 32 14 63, e-mail henglein@diku.dk, lokale 3-2-17.

4 Eksamensopgaver

De følgende sider indeholder de 9 eksamensopgaver, som skal løses og hvis besvarelse skal afleveres i henhold til ovenstående instruktioner.

Opgave 1 Funktionen `indsaet` defineret herunder indsætter et element på en angiven plads i en liste:

```
fun indsaet x (xr, n) = List.take (xr, n) @ x :: List.drop (xr, n)
```

(a) Erklær i SML en funktion `fjern : ''a -> ''a list -> ''a list * int`, som i følgende forstand har den omvendte virkning af `indsaet`: Et kald af formen `fjern` x xr skal returnere parret `(`yr`,` n`)`, hvor forekomsten længst til venstre af x i xr er på plads n (med nummerering, der begynder fra 0), og hvor yr er den liste, som vil fremkomme ved at fjerne denne forekomst af x fra xr. Dermed vil `indsaet` x `(fjern` x xr`)` altid returnere xr (forudsat, at x forekommer i xr). Der stilles intet krav til virkningen af `fjern` x xr i de tilfælde, hvor x ikke forekommer i xr.

Som et eksempel skal `fjern #"t" [#"k", #"l", #"a", #"t", #"r", #"e", #"t"]` returnere `([#"k", #"l", #"a", #"r", #"e", #"t"], 3)`.

(b) Hvilke betingelser skal x, n og xr opfylde, for at `fjern` x `(indsaet` x `(`xr`,` n`))` vil returnere `(`xr`,` n`)`?

Opgave 2 Funktionen `udvaelg : 'a list -> int list -> 'a list` skal for en liste xr og en liste af indices returnere listen af de pågældende elementer fra xr.

Kaldet `udvaelg [#"t", #"o", #"n", #"e"] [2,3,0,0,1]` skal for eksempel returnere `[#"n", #"e", #"t", #"t", #"o"]`.

(a) Erklær funktionen `udvaelg`.

(b) Angiv en erklæring af funktionen på formen

`val udvaelg2 =` ☐ `o` ☐ ☐

hvor de tre rammer er udfyldt med funktioner fra biblioteket `List` (lærebogen af H&R Appendix D.4 (Table D.17)) og funktionen `curry` defineret ved

```
fun curry f x y = f (x, y)
```

Obs: Ved en korrekt besvarelse af delopgave b vil det være tilstrækkeligt at besvare delopgave a med programlinjen

```
fun udvaelg xr = udvaelg2 xr
```

Opgave 3 I noterne IP-2 (afsnit 8.4 og 11.3.2) defineres `mergesort : real list -> real list` og hjælpefunktionerne `splitAt` og `merge`:

```
fun splitAt ([], _) = ([], [])
  | splitAt (xr, 0) = ([], xr)
  | splitAt (x :: xr, n) = let val (yr, zr) = splitAt (xr, n - 1)
                           in (x :: yr, zr) end
fun merge ([], yr) = yr
  | merge (xr, []) = xr
  | merge (xxr as (x : real) :: xr, yyr as y :: yr)
    = if x <= y then x :: merge (xr, yyr) else y :: merge (xxr, yr)
fun mergesort (xr as _ :: _ :: _)
    = let val (yr, zr) = splitAt (xr, length xr div 2)
      in merge (mergesort yr, mergesort zr) end
  | mergesort xr = xr
```

(a) Erklær en funktion `sortPerm : real list -> int list * real list`, der ud over at sortere sit argument også viser, hvilken permutation af argumentet der er foretaget. Med andre ord skal det være sådan, at hvis `sortPerm` *xr* returnerer `(`*nr*`,` *yr*`)`, så er *yr* listen *xr* i sorteret rækkefølge, og *nr* er elementernes positioner (nummereret fra 0) i den oprindelige liste *xr* (således at `udvaelg` *xr nr* `=` *yr*, hvor `udvaelg` er funktionen fra opgave 2).

Som et eksempel skal `sortPerm [3.4, 1.7, 6.9, 2.1]` returnere `([1, 3, 0, 2], [1.7, 2.1, 3.4, 6.9])`.

[Vink: `sortPerm` kan dannes ved modifikation af de viste funktioner `splitAt`, `merge` og `mergesort`.]

(b) Køretiden for `mergesort` har størrelsesorden $\mathcal{O}(n \log n)$, hvor n er længden af argumentlisten. Har køretiden for `sortPerm` samme størrelsesorden?

Opgave 4 Vi betragter heltalslister, som enten er den tomme liste eller har hoved forskellig fra nul. En sådan liste `[`a_n`,` a_{n-1}`, ...,` a_2`,` a_1`,` a_0`]` af $n+1$ hele tal, $a_n \neq 0$, opfattes som repræsentation af *polynomiet*

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0 \tag{1}$$

af *grad* n. Den tomme liste `[]` opfattes som repræsentation af *nulpolynomiet* 0 (som tillægges grad $-\infty$).

Som et eksempel opfattes SML-listen `[2, 0, 0, 1, ~18, ~3]` altså som repræsentation af femtegradspolynomiet $2x^5 + x^2 - 18x - 3$.

(a) Erklær en funktion `evalPoly : int list -> int -> int`, så kald af formen `evalPoly` *ar* k (forudsat at *ar* er tom eller har hoved forskelligt fra 0) vil returnere værdien af det af *ar* repræsenterede polynomium for $x = k$. Som et eksempel skal `evalPoly [2, 0, 0, 1, ~18, ~3] 2` returnere `29`.

(b) Erklær også en funktion `visPoly : int list -> string`, der angiver det repræsenterede polynomium som en tekst på følgende form: For $n \geq 2$ skal potenser af x skrives `x^`n; x^1 skal kun skrives som `x`, og x^0 skal ikke skrives. Nulpolynomiet skal skrives som `0`, men ellers skal led med koefficient 0 helt udelades, og de øvrige led skal forbindes med `+` og `-`, som man plejer at gøre. Der skal ikke være blanktegn i resultatet, koefficienten 1 skal kun skrives, når den er polynomiets konstantled, og der skal ikke være noget multiplikationstegn mellem koefficienterne og de efterfølgende potenser af x.

Som et eksempel skal `visPoly [2, 0, 0, 1, ~18, ~3]` returnere `"2x^5+x^2-18x-3"`.

Virkningen af `visPoly` på argumentlister med hovedet 0 er uden betydning.

(c) Hvis man lader `evalPoly [`a_n`,` a_{n-1}`, ...,` a_2`,` a_1`,` a_0`]` k beregne sit resultat ved en metode, der svarer til direkte indsættelse i formlen (1), bliver der brug for $\mathcal{O}(m)$ multiplikationer til beregning af hvert k^m og $\mathcal{O}(n^2)$ multiplikationer i alt. Udtryk beregningen ved en foldning, så antallet af multiplikationer kun bliver $\mathcal{O}(n)$.

[Obs: En korrekt besvarelse af delopgave c vil samtidig besvare delopgave a.]

Opgave 5 I denne opgave vil vi ved en *kalenderfil* forstå en tekstfil, hvis linjer enten er blanke (det vil sige kun indeholder blanktegn `#" "` og tabuleringstegn `#"\t"`) eller indledes med tre positive tal til angivelse af dato, måned og år. Efter hvert tal er der mellemrum (dannet af et eller flere blanktegn og/eller tabuleringstegn), og der kan også eventuelt være mellemrum foran det forreste tal. Resten af linjen, efter de tre tal, angiver den daterede begivenhed. Hvis årstallet er tocifret, skal det forstås som et år mellem 1920 og 2019; ellers er årstallet fircifret.

En kalenderfil kunne for eksempel indeholde:

```
   11 08 2011     Jane & Svends sølvbryllup

23 10 10     middag hos Aase
09  11   2010  Dansk   Datahistorisk   Forening
11 11   10  Møde med JS
```

Erklær en funktion `hentKalender : string -> (int * int * int) * string`, sådan at hvis f er stinavnet til en kalenderfil, vil kaldet `hentKalender` f returnere en liste af de angivne begivenheder, hvor hver begivenhed har form `((`*dd*`,` *mm*`,` *aaaa*`),` *tekst*`)`, idet datoen er angivet i rækkefølgen dag (*dd*), måned (*mm*) og år (*aaaa*), hvor årstallet om nødvendigt er kompletteret til fire cifre, hvor *tekst* er teksten fra resten af den pågældende linje i filen (uden det afsluttende linjeskift), og hvor listeelementerne er ordnet efter stigende dato.

Hvis `aftaler2010.txt` for eksempel er navnet på en fil med det ovenfor angivne indhold, skal kaldet `hentKalender "aftaler2010.txt"` returnere

```
[((2010, 10, 23), "middag hos Aase"),
 ((2010, 11, 9), "Dansk Datahistorisk Forening"),
 ((2010, 11, 11), "Møde med JS"),
 ((2011, 8, 11), "Jane & Svends sølvbryllup")]
```

Noter: Hvor tekst ude til højre i indgangslinjerne (til beskrivelse af begivenheder) indeholder mellemrum (et eller flere blanktegn og/eller tableringstegn), skal uddata også have mellemrum, men ikke nødvendigvis med samme antal blank- og tabuleringstegn.

Der er ikke i denne opgave noget krav om validering af, at kombinationerne af dag, måned og år angiver korrekte kalenderdatoer.

Opgave 6 En *formel* i *udsagnslogik* er et udtryk, som er bygget fra *udsagnsvariable* samt de logiske *konnektiver* $\neg$ ("ikke"), $\wedge$ ("og"), $\vee$ ("eller"); samt konstanter *tt* ("sand") og *ff* ("falsk"). Vi repræsenterer en formel i SML som en værdi af typen `prop`:

```
datatype prop
         = VAR of string
         | NOT of prop
         | AND of prop * prop
         | OR of prop * prop
         | TT
         | FF
```

Bemærk, at udsagnsvariable kan være givet med vilkårlige tekster; `NOT, AND, OR` står for de logiske konnektiver $\neg, \wedge, \vee$; og `TT, FF` repræsenterer henholdsvis *tt* og *ff*. Et eksempel på en erklæring af en formel er

```
val prop1 = OR (NOT (VAR "The pope sleeps"), VAR "The pope snores")
```

En *valuering* er en funktion af typen `string -> bool`, som tilordner en sandhedsværdi til hver udsagnsvariabel. For eksempel udtrykker valueringen

```
fun tassign1 "The pope sleeps" = true
  | tassign1 _ = false
```

at paven sover, men at han ikke snorker, og i øvrigt er alle andre udsagnsvariable også falske i `tassign1`.

En formel *har sandhedsværdien* `true` *under valuering* E, hvis:

- den har form `AND` (Φ, Ψ), og både Φ og Ψ har sandhedsværdi `true` under E; eller
- den har form `OR` (Φ, Ψ), og mindst en af Φ, Ψ har sandhedsværdi `true` under E; eller
- den har form `NOT` Φ, og Φ har sandhedsværdi `false` under E; eller
- den har form `TT`; eller
- den har form `VAR s`, og $E(s) =$ `true`.

I alle andre tilfælde har den sandhedsværdi `false`.

(a) Erklær en funktion `eval : prop -> (string -> bool) -> bool`, som beregner sandhedsværdien af en formel under en valuering; f.eks. skal `eval prop1 tassign1` returnere `false`.

(b) Erklær en funktion `implies: prop * prop -> prop`, som har følgende egenskab: `eval (implies (Φ, Ψ)) E = not (eval Φ E) orelse eval Ψ E`. Afprøv `implies` for at sandsynliggøre, at den har denne egenskab.

(c) To formler er *ækvivalente*, hvis de har samme sandhedsværdi under alle valueringer.

Erklær en funktion `simplify : prop -> prop` som returnerer en formel, der er ækvivalent med argumentet, og som har så få forekomster af `TT` og `FF` som muligt.

[Vink: Formel $\Phi \wedge ff$ er ækvivalent med *ff*, og $\Phi \wedge tt$ er ækvivalent med Φ. Formlen $\Psi \vee ff$ er ækvivalent med Ψ, og $\Psi \vee tt$ er ækvivalent med *tt*. Læg desuden mærke til, at $\wedge$ og $\vee$ er kommutative: $\Phi \wedge \Psi$ er ækvivalent med $\Psi \wedge \Phi$, og $\Phi \vee \Psi$ er ækvivalent med $\Psi \vee \Phi$.]

Opgave 7 En *klassedeling* af en mængde S er en mængde af parvis disjunkte ikke-tomme delmængder af S, hvis foreningsmængde er lige med S. Følgende er for eksempel klassedelinger af $\{1, 2, 3\}$:

- $\{\{1\}, \{2\}, \{3\}\}$,
- $\{\{1\}, \{2, 3\}\}$,
- $\{\{2\}, \{1, 3\}\}$,
- $\{\{3\}, \{1, 2\}\}$,
- $\{\{1, 2, 3\}\}$.

Der er ikke andre klassedelinger. Dermed har $\{1, 2, 3\}$ samlet 5 forskellige klassedelinger.

(a) Erklær en funktion `klassedelinger: 'a list -> 'a list list list`, som returnerer alle klassedelinger af argumentet.

Her skal mængder repræsenteres ved dubletfri lister. (Elementernes rækkefølge i listerne er uden betydning.) Kaldet `klassedelinger [1, 2, 3]` kunne for eksempel returnere

`[[[1], [2], [3]], [[1], [2, 3]], [[1, 2], [3]], [[2], [1, 3]], [[1, 2, 3]]]`,

men lister og elementer kunne også stå i en anden rækkefølge.

(b) Antallet af klassedelinger af en mængde med n elementer afhænger kun af n. Det hedder *Bell*-tallet B_n. B_0 er 1, og for $n \geq 0$ kan B_{n+1} bestemmes ved følgende formel:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

Erklær en funktion `bell : int -> int` til beregning af Bell-tallet.

(c) En medstuderende påstår, at der gælder

$$\texttt{bell (}n + m\texttt{) mod } m \;=\; \texttt{(bell (}n + 1\texttt{) + bell } n\texttt{) mod } m$$

for alle $m > 1$ og $n > 1$. Det stiller du dig dog tvivlende overfor.

Afprøv din medstuderendes påstand.

Opgave 8 Lad `KLIST` være en signatur for *konkatenerbare lister*, som vi kalder *k-lister*:

```
signature KLIST =
   sig
   (* Typen af endelige k-lister med elementer af type 'a *)
      type 'a klist
   (* Den tomme k-liste *)
      val tom : 'a klist
   (* Et-elements k-liste *)
      val singleton : 'a -> 'a klist
   (* Konkateneringen af to k-lister *)
      val konkat : 'a klist * 'a klist -> 'a klist
   (* Omsætning til almindelig liste *)
      val tilListe : 'a klist -> 'a list
   (* Længden af en k-liste, det vil sige antallet af elementer i den *)
      val laengde : 'a klist -> int
   end
```

Et element af typen `'a klist` kan opfattes som en almindelig liste, men med andre primitive operationer end den i SML indbyggede type `'a list`.

(a) Erklær en struktur `TrivKList :> KLIST` ved hjælp af den indbyggede listetype: `type 'a klist = 'a list`.

(b) Erklær en funktion `fraListe : 'a list -> 'a klist` som konstruerer en k-liste fra en SML-liste. Hvordan afprøver du korrektheden af din implementation? (Hvilken egenskab skal du afprøve?)

(c) Strukturen `TrivKList` ovenfor implementerer k-lister ved hjælp af SML-lister. Find en asymptotisk mere effektiv implementering `KList :> KLIST`, som har følgende egenskaber: Alle funktioner i `KLIST` med undtagelse af `tilListe` kører i konstant tid ("$\mathcal{O}(1)$"), og `tilListe` kører i tid $\mathcal{O}(n)$, hvor n er længden af argumentet.

Opgave 9 For et positivt helt tal n vil vi i denne opgave ved en *permutation over* n forstå en liste, der netop en gang indeholder hvert af tallene $0, 1, \ldots, n-1$. Som bekendt er der $n!$ forskellige permutationer over n. Funktionen `nrPerm` defineret nedenfor (hvor `fjern` er funktionen beskrevet i opgave 1) knytter til hver permutation over et naturligt tal n et entydigt løbenummer mellem 0 og $n! - 1$:

```
fun nrPerm [] = 0
  | nrPerm nr
    = let val n = length nr
          val (mr, m) = fjern (n-1) nr
      in n * nrPerm mr + m end
```

(a) Konstruer en funktion `permNr : int -> int -> int list`, så `permNr` n k for $0 \leq k < n!$ returnerer permutationen over n med løbenummer k. For $k < 0$ eller $k \geq n!$ er virkningen af `permNr` n k uden betydning.

For $0 \leq k \leq n! - 1$ vil der altså gælde

$$\begin{aligned} \texttt{length (permNr } n\ k\texttt{)} &= n \\ \texttt{nrPerm (permNr } n\ k\texttt{)} &= k. \end{aligned}$$

5 Rettelser og forklaringer

Status: 2. november 2010, kl. 22:00 (endelig)

Følgende indeholder rettelser samt yderligere forklaringer til opgaveteksten i afsnit 4, som sammenfatter alle *væsentlige* rettelser og forklaringer fra IP-forummet.

5.1 Rettelser

- Opgave 5, "Erklær en funktion `hentKalender : string -> (int * int * int) * string`, ...": Det skal erstattes med "Erklær en funktion `hentKalender : string -> ((int * int * int) * string) list`, ...".

- Opgave 5: Paragrafen

  "Hvis `aftaler2010.txt` for eksempel er navnet på en fil med det ovenfor angivne indhold, skal kaldet `hentKalender "aftaler2010.txt"` returnere

  ```
  [((2010, 10, 23), "middag hos Aase"),
   ((2010, 11, 9), "Dansk Datahistorisk Forening"),
   ((2010, 11, 11), "Møde med JS"),
   ((2011, 8, 11), "Jane & Svends sølvbryllup")]"
  ```

  erstattes med:

  "Hvis `aftaler2010.txt` for eksempel er navnet på en fil med det ovenfor angivne indhold, skal kaldet `hentKalender "aftaler2010.txt"` returnere

  ```
  [((23, 10, 2010), "middag hos Aase"),
   ((9, 11, 2010), "Dansk Datahistorisk Forening"),
   ((11, 11, 2010), "Møde med JS"),
   ((11, 8, 2011), "Jane & Svends sølvbryllup")]"
  ```

5.2 Yderligere forklaringer

- Opgave 4(b): `visPoly` skal undertrykke koefficienten, hvis den er -1 eller 1. F.eks. skal det returnere `"2x^2-x+3"`, ikke `"2x^2-1x+3"`.

- Opgave 7(a): Funktionen `klassedelinger` skal acceptere lister af vilkårlig længde, ikke kun lister af længde 3.

- Opgave 3(a): Hvis der er flere ens elementer i indata, kan `sortPerm` returnere dem i vilkårlig indbyrdes rækkefølge. F.eks. er både `([0, 2, 1, 3], [1.3, 1.3, 2.9, 4.7])` og `([2, 0, 1, 3], [1.3, 1.3, 2.9, 4.7])` tilladelige resultater på kaldet `sortPerm [1.3, 2.9, 1.3, 4.7]`.

- Opgave 8(c): Vink: Erklær `'a klist` i strukturen `KList` som datatype:

```
structure CList :> CLIST =
  struct
  datatype 'a clist =
    ...
  ...
  end
```

- Opgave 2: Hvis funktionen `udvaelg` løses som angivet under "Obs", indsættes erklæringen af `udvaelg` efter erklæringen af `udvaelg2`.

- Opgave 2(b): Typen af `udvaelg2` vil give "value polymorphism" advarslen på grund af restriktion af værdi-polymorfi typereglen i Standard ML. Det er *uundgåeligt* og er en *fuldt korrekt* løsning af opgaven, hvis `udvaelg2` i øvrigt opfylder de stillede krav i opgaven.

  Ligeledes vil erklæringen af `udvaelg` ved hjælp af `fun udvaelg xr = udvaelg2 xr` ikke være polymorf efter første anvendelse på et argument. Det er *uundgåeligt* og er en *fuldt korrekt* løsning af opgaven, hvis `udvaelg2` i øvrigt opfylder de stillede krav i opgaven.

(Opgavesættet slut)