Ordinær eksamen i Introduktion til programmering, blok 1, 2011

2. november 2011
Version 4 (endelig): 2011-11-03, 13:00

[VIGTIGT: Se alle opdateringer (afsnit 5) til den oprindelige opgave (afsnit 1-4)!]

Dette dokument udgør opgavesættet for den ordinære eksamen i kurset "Introduktion til programmering", blok 1, 2011. Det består af 18 nummererede sider.

Dokumentet offentliggøres onsdag den 2. november kl. 9:00 på kursets hjemmeside via KU's kursusadministrationssystem Absalon. Besvarelsen skal afleveres senest fredag den 4. november kl. 9:00 — se dog afsnit 2 nedenfor hvis du sideløbende er indskrevet på kurset MatIntro.

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 KU's studieadministrative system kort tid derefter.

Eksamenssættet består af 8 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 studerende, som har opnået mindst 5 point i kursets obligatoriske afleveringer.

Hvis der opstår tvivl om selvstændighed eller opnåelse af læringsmålene ud fra besvarelsen, kan studerende blive indkaldt til en supplerende mundtlig eksamen fredag den 18. november 2011, kl. 13. Studerende vil blive indkaldt til mundtlig eksamen med elektronisk brev til deres KU-konti senest torsdag den 17. november 2011 kl. 12:00, hvorfor *alle studerende skal efterse deres konto den 17. november 2011 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 også afsnit 3 nedenfor.

1 Selvstændighed i besvarelsen og eksamenssnyd

Opgavesættet skal besvares *individuelt.* Det er tilladt at diskutere opgave*formuleringerne* herunder at stille opklarende spørgsmål til opgavernes fortolkning, til instruktorerne under instruktorvagtordningen og på kursets elektroniske diskussionsforum.

Det er *ikke* tilladt at diskutere *besvarelse* af opgaverne med andre personer, herunder afprøvningstilfælde, løsningsmetoder, algoritmer eller konkret programtekst. 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 diskussionsforum.
- 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 efter følgende regler.

2.1 Afleveringsfrist

Den *ordinære* afleveringsfrist er **fredag, den 4. november, kl. 9:00**. Hvis og *kun hvis* (!) du

- sideløbende er *indskrevet* på kurset MatIntro og
- *deltager* i forelæsninger og øvelser på MatIntro torsdag, den 3. november, kl. 9:00-17:00

så er din afleveringsfrist fredag, den 4. november, kl. 17:00. Denne *ekstraordinære* afleveringsfrist finder kun anvendelse for MatIntro-studerende for at sikre lige vilkår for alle eksamensdeltagere i deres respektive eksamensperioder. Hvis du er MatIntro-studerende er det *ikke* nødvendigt at søge om dispensation.[1]

2.2 Procedure

Eksamenssættet uploades efter samme procedure som aflevering af de obligatoriske opgaver på kurset. På kursets Absalon-hjemmeside findes menupunktet "Eksamen", hvorunder opgavepunkter med titlerne "Aflevering af eksamen (ordinær frist)" og "Aflevering af eksamen (ekstraordinær frist)" forefindes. Du skal anvende førstnævnte, hvis din afleveringsfrist er kl. 9:00; sidstnævnte, hvis den er kl. 17:00. I tilfælde af—og *kun* i tilfælde af—at Absalon ikke fungerer i *hele* timen inden din afleveringsfrist, udskydes afleveringsfristen med en time til kl. 10:00, hhv. 18:00. Skulle Absalon heller ikke fungere i denne periode, skal besvarelsen umiddelbart efter udløb af fristen sendes med email til `henglein@diku.dk`.

Det er den studerendes eget ansvar at gøre sig bekendt med, om Absalon fungerer på afleveringstidspunktet.[2] Der kan forekomme mild overbelastning, hvis mange forsøger at aflevere eksakt samtidigt — du opfordres derfor til at uploade din besvarelse i så god tid som muligt.

[1] Er du indskrevet på et andet for din uddannelse væsentligt *kursus* med *forpligtende* kursusaktiviteter i den ordinære IP-eksamensperiode fra 2. november, kl. 9:00, til 4. november, kl. 9:00, kan du tilsvarende søge om forlænget eksamenstid ved fremlæggelse af dokumentation af: indskrivning, de forpligtende aktiviteter og din deltagelse i dem. Dette skal ske ved email til `henglein@diku.dk` *inden* udløb af den *ordinære* eksamensfrist.

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

2.3 Format

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 skal angive sit *fulde* navn. 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.

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`.

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 eller visning af evalueringstrin), der ikke kan afvikles under Moscow ML, skal 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, hvor et tabulatortegn tæller som 6 mellemrumstegn. (Det er dog bedst at undgå brugen af tabulatortegn.)

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.

3 Støtte i eksamensperioden

Spørgsmål om eventuelle uklarheder i opgaveteksten samt spørgsmål om formalia i forbindelse med eksamen kan stilles til instruktorvagten i DIKUs kantine den 3. og 4. november i perioden 9:00-18:00 samt på kursets elektroniske diskussionsforum under Absalon. Opdaterede versioner af opgavesættet med eventuelle rettelser og besvarelser af relevante spørgsmål til instruktorerne og på diskussionsforummet lægges på Absalon under menupunktet "Eksamen" på følgende tidspunkter:

- onsdag, 2. november, kl. 12:00
- onsdag, 2. november, kl. 16:00
- torsdag, 3. november, kl. 12:00

Disse og kun disse gælder som supplerende oplysninger om eksamenen. Det er den enkelte studerendes ansvar at holde sig ajour med disse opdateringer. Indlæg på diskussionsforummet er ikke autoritative og kan i værste fald være misvisende. Da væsentlige spørgsmål, rettelser og kommentarer fra diskussionsforummet medtages i opdateringerne, er det således ikke nødvendigt løbende at følge diskussionsforummet.

Kun spørgsmål og meddelelser med fortroligt indhold rettes direkte til den kursusansvarlige, Fritz Henglein, tlf. 30589576, e-mail henglein@diku.dk, lokale 3-2-17.

4 Eksamensopgaver

De følgende sider indeholder de 8 eksamensopgaver, som skal løses og hvis besvarelse skal afleveres i henhold til ovenstående instruktioner.

Opgave 1 *Stirlingtallet* $\left\{ {n \atop k} \right\}$ *af anden art* er antallet af måder, en mængde af n elementer kan deles i k ikke-tomme delmængder. $\left\{ {n \atop k} \right\}$ kan defineres rekursivt således:

$$\begin{aligned}
\left\{ {0 \atop 0} \right\} &= 1 \\
\left\{ {n \atop 0} \right\} &= 0 \qquad (n > 0) \\
\left\{ {0 \atop k} \right\} &= 0 \qquad (k > 0) \\
\left\{ {n+1 \atop k} \right\} &= k \cdot \left\{ {n \atop k} \right\} + \left\{ {n \atop k-1} \right\} \qquad (n \geq 0, k > 0)
\end{aligned}$$

(a) Erklær en funktion `stirling : int * int -> int` i SML, således at `stirling (`n`,`k`)` returnerer $\left\{ {n \atop k} \right\}$ hvis $n \geq 0, k \geq 0$ og $\left\{ {n \atop k} \right\}$ kan repræsenteres som element af typen `int` i SML. Hvis $n \geq 0$ og $k \geq 0$, men $\left\{ {n \atop k} \right\}$ er for stort et tal til at kunne repræsenteres som element af `int`, skal `stirling(`n`,`k`)` kaste undtagelsen `Overflow`. Hvis $k < 0$ eller $n < 0$, så skal `stirling(`n`,`k`)` kaste undtagelsen `Domain`.

Enhver anvendelse af `stirling` skal *helst* tage maksimum 1 sekund at evaluere på en almindelig pc. [Vink: Overvej hvad værdien af $\left\{ {n \atop k} \right\}$ er for $n \leq k$.]

(b) Vis evalueringstrinnene (analog til Hansen & Rischel, side 8, hvor evalueringstrinnene for `fact 4` er vist) tekstuelt i en SML-kommentar for følgende udtryk:

- `stirling (4,2)`
- `stirling (~3,0)`
- `stirling (~2,~1)`
- `stirling (30,28)`

For `stirling (30,28)` behøver du kun at vise maksimum 5 evalueringstrin.

Du må gerne forkorte `stirling` til `s` i evalueringstrinnene i delopgave b, dog *skal* funktionen i delopgave a hedde `stirling`!

Opgave 2 I denne opgave vil vi ved en *nedre trekant af orden m* forstå en SML-værdi af type `'a list list`, der er en liste af m lister, for hvilke den første liste har længde en, den anden længde to og så fremdeles indtil den sidste af de m lister, der skal have længde m.

Værdien `trekant` herunder er eksempel på en nedre trekant af orden 4:

```
val trekant = [[1.2], [~3.5, 4.1], [5.2, 6.0, 0.7], [3.9, ~1.4, 2.0, 8.8]]
```

En værdi er en *nedre trekant*, hvis den er en nedre trekant af orden m for et eller andet naturligt tal m. Som et udartet tilfælde regnes den tomme liste `[]` også for en nedre trekant (nemlig af orden 0).

(a) Definer en funktion `erNedreTrekant : 'a list list -> bool`, som netop returnerer `true`, hvis dens argument er en nedre trekant, og ellers `false`.

(b) *Diagonalen* af en nedre trekant `t` af orden m er listen bestående af elementet i første liste af `t`, andet element af anden liste i `t`, og så videre op til sidste element af den sidste liste i `t`.

Definer en funktion `diagonal : 'a list list -> 'a list`, så `diagonal t` for en nedre trekant `t` danner diagonalen af `t`. Virkningen af et kald af form `diagonal t`, når `t` ikke er en nedre trekant, er uden betydning.

For eksempeltrekanten ovenfor skal gælde

```
diagonal trekant;
val it = [1.2, 4.1, 0.7, 8.8] : real list
```

(c) Definer en funktion `vend : 'a list list -> 'a list list`, sådan at hvis `t` er en nedre trekant af orden m, vil `vend t` også blive en liste af m lister, men den første af disse vil være alle hovederne af listerne i `t`, den anden de $m-1$ elementer på anden plads af listerne i `t`, og så videre indtil den sidste liste, der kun vil have et element, nemlig det sidste element af den sidste liste i `t`.

For eksempeltrekanten ovenfor skal gælde

```
vend trekant;
val it = [[1.2, ~3.5, 5.2, 3.9], [4.1, 6.0, ~1.4], [0.7, 2.0], [8.8]]
         : real list list
```

For værdier `t`, som ikke er en nedre trekant, er det uden betydning, hvad et kald af form `vend t` fører til.

Opgave 3 Herunder defineres tre funktioner `f`, `g` og `h`. Forklar i almindeligt sprog for hver af de tre funktioner, hvad resultatet af at kalde dem vil være. (Forklaringerne skal ikke referere til rekursive kald eller definitionernes struktur, men gøre rede for, hvilken *værdi* der til syvende og sidst vil komme ud af funktionskaldene.) Omformuler også hver af de tre definitioner, så der på passende måde benyttes *polymorfe højereordensfunktioner* i definitionerne.

(a)
```
local fun kth (y :: _) 1 = y
          | kth (_ :: yr) n = kth yr (n-1)
in fun f _ nil = nil
     | f xr (n :: nr) = kth xr n :: f xr nr
end
```

(b)
```
fun g xs nil = xs
  | g xs (f :: fs) =
      let fun loop nil = nil
            | loop (y :: ys) = if f y then y :: loop ys else loop ys
      in g (loop xs) fs
      end
```

(c)
```
fun h [x : real] = (x,x)
  | h (x :: xr) = let val (frst,sdst) = h xr
                  in if x < frst then (x,sdst)
                     else if x > sdst then (frst,x) else (frst,sdst)
                  end
```

(d) Lad `f` og `h` være funktionerne defineret i delopgaverne a og c (*ikke* dine omformuleringer). Antag, at `xr : real list` har længde n, og angiv i en SML-kommentar tidskompleksiteten (som funktion af n) af henholdsvis funktionskald af form `f xr (upto (1,`n`))` og af form `h xr` (idet `upto` er funktionen fra afsnit 5.2.1 i lærebogen af Hansen & Rischel).

Opgave 4 En $m \times n$ *matrix* (flertal: matricer) for $m, n \geq 1$ er en to-dimensional opstilling af m *rækker* af elementer, hvor hver række har n elementer, illustrereret således:

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ & \vdots & \\ a_{m1} & \dots & a_{mn} \end{pmatrix}$$

Her er en 2×3 matrix A_1 hvis *indgange* (elementer) er heltal:

$$\begin{pmatrix} 4 & 1 & 3 \\ 2 & 0 & 5 \end{pmatrix}$$

Den har 2 rækker og 3 *søjler*.

I denne opgave forudsætter vi, at matricer er repræsenteret som en liste af rækker, hvor hver række er repræsenteret som en liste med dens elementer:[3]

```
type 'a matrix = 'a list list (* alle rækker skal have samme længde *)
```

For eksempel er ovenstående 2×3 matrix repræsenteret således:

```
val a1 = [ [4, 1, 3], [2, 0, 5] ]
```

(a) En værdi af typen `'a list list` repræsenterer kun en matrix, hvis alle rækker har samme længde. Erklær en funktion `isMatrix : 'a list list -> bool`, som returnerer `true` hvis argumentet repræsenterer en matrix, ellers `false`.

(b) Erklær en funktion `dim : 'a matrix -> int * int`, som returnerer (m, n) hvis argumentet repræsenterer en $m \times n$ matrix. Det kan forudsættes, at argumentet er en matrix.

(c) Den *transponerede* A^T af en matrix A er en anden matrix, der dannes ved at lave rækker til søjler og omvendt. For eksempel er den transponerede A_1^T af A_1, ovenstående 2×3 matrix, følgende 3×2 matrix:

$$\begin{pmatrix} 4 & 2 \\ 1 & 0 \\ 3 & 5 \end{pmatrix}$$

Erklær en funktion `transpose : 'a matrix -> 'a matrix` som transponerer sit argument. Det kan antages, at argumentet er en matrix.

(d) Matrixmultiplikation er defineret på følgende måde. Givet $m \times k$ og $k \times n$ matricer

$$A = \begin{pmatrix} a_{11} & \dots & a_{1k} \\ & \vdots & \\ a_{m1} & \dots & a_{mk} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & \dots & b_{1n} \\ & \vdots & \\ b_{k1} & \dots & b_{kn} \end{pmatrix}$$

så er indgangene i deres *matrixprodukt* $A \cdot B$ en $m \times n$ matrix

$$\begin{pmatrix} c_{11} & \dots & c_{1n} \\ & \vdots & \\ c_{m1} & \dots & c_{mn} \end{pmatrix}$$

[3] Det kaldes række-baseret repræsentation (eng. row major order).

defineret ved

$$c_{ij} = \sum_{\ell=1}^{\ell=k} a_{i\ell} \cdot b_{\ell j} \qquad (= a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} + \ldots + a_{ik} \cdot b_{kj})$$

Her forudsættes, at elementerne kan *adderes* og *multipliceres*. Vi bruger heltalsaddition og -multiplikation i denne opgave. For eksempel, er $A_1 \cdot A_1^T$ følgende 2×2 matrix:

$$\begin{pmatrix} 26 & 23 \\ 23 & 26 \end{pmatrix}$$

I dette eksempel er indgangen c_{12} (værdien i række 1 og søjle 2) resultatet af følgende udregning:

$$c_{12} = 4 \cdot 2 + 1 \cdot 0 + 3 \cdot 5 = 8 + 0 + 15 = 23.$$

Erklær en funktion `multiply : int matrix * int matrix -> int matrix`, som returnerer matrixproduktet af inputargumenterne. Her kan det antages, at begge argumenter er matricer. Hvis deres dimensioner ikke passer sammen, skal `multiply` kaste undtagelsen `Domain`.

Opgave 5 *Burrows-Wheeler Transformationen (BWT)* er en funktion, som permuterer tegnene i en tekststreng således, at resultatet kan komprimeres bedre end den oprindelige tekststreng. BWT er brugt i datakomprimering, hvor en høj kompressionsfaktor er krævet (f.eks. i bzip2) og er over de sidste 5 år blevet et vigtigt værktøj i effektiv gensekvensanalyse i bioinformatik. I denne opgave skal du implementere BWT og dens inverse funktion InvBWT i Standard ML.

Specifikation af BWT og InvBWT. Hvis man flytter 0, 1 eller flere tegn fra slutningen til begyndelsen af en tekststreng, kaldes det en *rotation*. BWT kan nu specificeres uformelt således: Givet en tekststreng s,

1. konstruer alle rotationer af s;
2. sorter rotationerne alfabetisk; og
3. returner det sidste tegn fra hver rotation, sammensat til en tekststreng.

For eksempel, BWT af `"azzcy"` beregnes således:

1. Rotationerne af `"azzcy"` er `["azzcy", "yazzc", "cyazz", "zcyaz", "zzcya"]`.
2. Deres alfabetiske sortering resulterer i listen `["azzcy", "cyazz", "yazzc", "zcyaz", "zzcya"]`.
3. Det sidste tegn fra rotationerne i sorteret rækkefølge bliver `[#"y", #"z", #"c", #"z", #"a"]`. Sammensat til en tekststreng giver det `"yzcza"` som resultat.

Den *inverse Burrows-Wheeler Transformation (InvBWT)* tager en tekststreng t og et tegn ω, som forekommer præcist en gang i t, som input og konstruerer den tekststreng s, hvis BWT er t og som har ω som sluttegn. InvBWT kan implementeres på følgende måde:

1. Start med en liste $\vec{s}_0$ bestående af n tomme strenge `["", ..., ""]`, hvor n er længden af t. For $i = 1, \ldots, n$, gør følgende:
   (a) Sorter $\vec{s}_{i-1}$ alfabetisk og kald resultatet $\vec{s}_i^{(*)}$.
   (b) Konstruer $\vec{s}_i$ ved at lyne (eng. *zip*) t med $\vec{s}_i^{(*)}$: Hvis $t = t^{(1)} \ldots t^{(n)}$ og $\vec{s}_i^{(*)} = [s_i^{(1)}, \ldots, s_i^{(n)}]$, så er $\vec{s}_i = [t^{(1)} s_i^{(1)}, \ldots, t^{(n)} s_i^{(n)}]$.
2. $\vec{s}_n$ indeholder alle rotationer af s. Returner den streng $s_n^{(j)}$ i s_n som ender med ω.

For eksempel, hvis vi har $t =$ `"yzcza"` og $\omega =$ `#"y"`, så beregnes dens InvBWT i følgende trin.

1. Vi beregner $\vec{s}_1^{(*)}, \vec{s}_1, \vec{s}_2^{(*)}, \vec{s}_2, \vec{s}_3^{(*)}, \vec{s}_3, \vec{s}_4^{(*)}, \vec{s}_4, \vec{s}_5^{(*)}, \vec{s}_5$:

$$
\begin{array}{lcl}
\vec{s}_1^{(*)} & = & [\text{"", "", "", "", ""}] \\
\vec{s}_1 & = & [\text{"y", "z", "c", "z", "a"}] \\
\vec{s}_2^{(*)} & = & [\text{"a", "c", "y", "z", "z"}] \\
\vec{s}_2 & = & [\text{"ya", "zc", "cy", "zz", "az"}] \\
\vec{s}_3^{(*)} & = & [\text{"az", "cy", "ya", "zc", "zz"}] \\
\vec{s}_3 & = & [\text{"yaz", "zcy", "cya", "zzc", "azz"}] \\
\vec{s}_4^{(*)} & = & [\text{"azz", "cya", "yaz", "zcy", "zzc"}] \\
\vec{s}_4 & = & [\text{"yazz", "zcya", "cyaz", "zzcy", "azzc"}] \\
\vec{s}_5^{(*)} & = & [\text{"azzc", "cyaz", "yazz", "zcya", "zzcy"}] \\
\vec{s}_5 & = & [\text{"yazzc", "zcyaz", "cyazz", "zzcya", "azzcy"}]
\end{array}
$$

2. Resultatet er `"azzcy"`, da det er den tekststreng i $\vec{s}_5$, som ender med sluttegnet `#"y"`.

Delopgaver:

(a) Definer i SML en funktion `BWT : string -> string`, som implementerer BWT.

[Vink: Brug `Listsort.sort` eller erklær din egen sorteringsfunktion. Før du kan bruge `Listsort.sort` med `mosml`, skal du udføre `load "Listsort";`.]

(b) Definer i SML en funktion `invBWT : string * char -> string`, som implementerer den inverse BWT.

(c) Afprøv om din implementering af `invBWT` er den inverse funktion af `BWT`. For eksempel skal der gælde

```
val testBWT1 = invBWT (BWT "azzcy", #"y") = "azzcy" handle _ => false;
val testBWT1 = true : bool
```

Erklær passende hjælpefunktioner i `local`-erklæringer i a og b. Der er ikke noget krav til typerne af hjælpefunktionerne, men `BWT` og `invBWT` *skal* have de angivne typer.

Opgave 6 Betragt binære træer med information knyttet til sine forgreningspunkter, defineret på følgende måde

```
datatype 'a btree = Blad | Gren of 'a btree * 'a * 'a btree
```

De i undervisningen beskrevne omformninger fra `'a btree` til "lineær" form som `'a list` (preorder, inorder, postorder) er ikke *enentydige (injektive*, eng. *one-to-one, injective)*: Det er normalt ikke muligt at gendanne et træ fra sin lineære form. Med henblik på at afhjælpe denne mangel defineres for hver type τ en "udvidet" type τ `udvidelse`:

```
datatype 'a udvidelse = % | << | == of 'a | >>
(* % repræsenterer et blad, << og >> fungerer som henholdsvis venstre- og *)
(* højreparentes. == er bare med, fordi der skal optræde en værdikonstruktør på det sted. *)
```

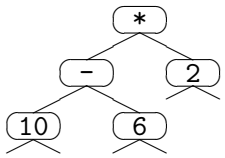
Med denne nye type til rådighed kan man linearisere et binært træ men samtidig bevare strukturen:

```
fun lineariser Blad = [%]
  | lineariser (Gren (t1,x,t2)) = << :: lineariser t1 @ == x :: lineariser t2 @ [>>]
```

For træet `udtr : string btree` defineret af

```
val udtr = Gren (Gren (Gren (Blad,"10",Blad),"-",Gren (Blad,"6",Blad)),
                 "*",
                 Gren (Blad,"2",Blad))
```

(illustreret med figuren herunder) vil `lineariser udtr : string udvidelse list` for eksem-



Figur 1: Visualisering af `udtr : string btree`

pel blive

```
[<<,<<,<<,%,=="10",%,>>,=="-",<<,%,=="6",%,>>,>>,=="*",<<,%,=="2",%,>>,>>]
```

(Opgaveformulering på næste side)

(a) Definer i SML en funktion `gendan : 'a udvidelse list -> 'a btree`, som gendanner et binært træ fra sin linearisering: For alle `t :: 'a btree` skal det være sådan, at man får `t` tilbage som værdi af `gendan (lineariser t)`.

Det er klart, at ikke alle værdier af type `'a udvidelse list` repræsenterer binære træer; blandt andet skal antallet af `<<`'er og `>>`'er være det samme, og gennemløbes listen fra venstre, må antallet af `>>`'er aldrig overstige antallet af `<<`'er. Hvorledes `gendan x` virker, når `x` ikke er en værdi af form `lineariser t`, er uden betydning.

[Vink: Konstruer en hjælpefunktion `analyser : 'a udvidelse list -> 'a btree * 'a udvidelse list`, som danner et træ af en indledende del af sit argument og returnerer dette træ og den ubenyttede del af argumentet. Med andre ord skal `analyser` kun kaldes med argumenter, der kan skrives på formen `lineariser t @ ur` og fra et sådant kald returnere `(t,ur)`.

Følges dette vink, løses opgaven af et program på formen

```
fun analyser (%  :: ur) = (Blad,ur)
  | analyser (<< :: ur) = (* skriv mig! *)
fun gendan ur = let val (t,[]) = analyser ur in t end
```

]

Opgave 7 Programmet `wc`, som står for *word count*, er et nyttigt hjælpeprogram, der er blevet indført i begyndelsen af 70erne i den første version af operativsystemet UNIX, foreløberen for alle på nuværende tidspunkt udbredte *desktop*-operativsystemer. I denne opgave skal du implementere funktioner i SML svarende til — men ikke (!) nødvendigvis identisk med – funktionaliteten af `wc`.

(a) *Ordene* i en tekststreng er ikke-tomme tegnfølger, som er adskilt fra hinanden ved blanktegn og ikke selv indeholder blanktegn. (Et tegn `c` er et *blanktegn*, hvis `Char.isSpace c` returnerer `true`.)

Erklær en funktion `wordNum : string -> int`, som returnerer antallet af ord i en tekststreng. For eksempel, skal der gælde

```
wordNum " blib blob c1d -45$3 4Ad\n";
val it = 5 : int
```

(b) En *linje* er en muligvis tom tegnfølge uden forekomst af *linjeskifttegnet* `#"\n"`, efterfulgt af en enkelt forekomst of linjeskifttegnet. Desuden gælder en ikke-tom tegnfølge uden efterfølgende linjeskifttegn også som linje, hvis det udgør slutningen af inputtet.

Erklær en funktion `wc1 : string -> int * int * int`, som returnerer antallet af linjer, ord og tegn (i denne rækkefølge) i en tekstfil. Hvis åbningen af filen som tekstfil ikke lykkes, skal `wc1` kaste undtagelsen `Io`.

[Vink: Se D.10 i Hansen & Rischel og brug `wordNum` fra delopgave a.]

(c) Erklær en funktion `display : int -> int -> string`, som konverterer tallet i det andet argument til dets decimalrepræsentation med 0 eller flere forekomster af *mellemrumstegnet* `#" "`. Resultatet skal have mindst det antal tegn, der er angivet i det første argument, men ellers ikke flere end nødvendigt. For eksempel skal der gælde

```
val d1 = display 1 24;
val d1 = "24": string
val d2 = display 2 24;
val d2 = "24": string
val d3 = display 3 24;
val d3 = " 24": string
val d8 = display 8 24;
val d8 = "      24": string
```

(Der er 1 mellemrumstegn i `d3` og 6 mellemrumstegn i `d8`.)

[Vink: Brug `Int.toString` og tilføj det fornødne antal foranstillede mellemrumstegn.]

(d) Erklær en funktion `wc : string list * string option -> unit`, som har en liste af inputfilnavne og et outputfilnavn som argumenter. Hvis intet outputfilnavn er specificeret (dvs. det andet argument er `NONE`), så skal uddata skrives til `TextIO.stdOut` i stedet for.

Funktionen `wc` skal opfylde følgende egenskaber:

- For hver inputfil skrives antal linjer, ord og tegn i inputfilen samt navnet af inputfilen til outputfilen. Disse 4 oplysninger skal være adskilt fra hinanden med mindst et mellemrumstegn og afsluttet med linjeskifttegnet. Antallet af mellemrumstegn skal sikre, at de 3 tal hvis muligt er *højrejusteret* på hhv. 8., 16. og 24. position på linjen.

- Hvis en inputfil ikke kan åbnes som tekstfil, skal filnavnet efterfulgt af

  `" cannot be opened\n"`

  skrives til outputfilen.
- Funktionen `wc` skal *helst* bruge en hjælpefunktion `procSingleFile : string -> unit`, som udskriver uddata for inputfilnavnet og som er defineret i kroppen af `wc`.

  [Vink: Brug `wc1` og `display`. Bemærk desuden, at `procSingleFile` kan anvendes på en liste af filer ved hjælp af `List.app`.]

Lad os antage, at filen `test01.txt` indeholder `" blib blob c1d -45$3 4Ad\n\nd\rz44\n"` og filen `test02.txt` indeholder `"dfj\n"`. Så skal der gælde

```
wc (["test01.txt", "test02.txt"], NONE);
       3       7      32 test01.txt
       1       1       4 test02.txt
val it = () : unit
```

Her er 7 mellemrumstegn før første forekomst af *3* i første outputlinje, før forekomsten af *7* i første outputlinje og før begge forekomster af *1* i anden outputlinje. Der er 6 mellemrumstegn før forekomsten af *32* i første outputlinje. Der er et enkelt mellemrumstegn før filnavnene i begge outputlinjer.

Opgave 8 Example 17.2 i Hansen & Rischel viser med definitionen af `power2`, hvordan opløftning til n-te potens i stedet for at bruge $\mathcal{O}(n)$ multiplikationer kan benytte gentagne kvadreringer og derved udføres med kun $\mathcal{O}(\log n)$ multiplikationer.

Nøjagtig den samme metode kan benyttes for andre typer end tal og for andre operationer end multiplikation; det kræves blot, at den type, der skal arbejdes med, har en binær operation, der kan erstatte multiplikation, og et såkaldt "neutralelement" eller "etelement", der kan erstatte `1.0` i eksemplet.

Signaturen `MONOID` stiller dette krav:

```
signature MONOID
= sig type t
      val mul : t * t -> t (* mul(a,mul(b,c)) = mul(mul(a,b),c) *)
      val one : t          (* mul(a,one) = mul(one,a) = a *)
  end
```

Signaturen `MONOIDWPOW` dannes ved udvidelse med en potensopløftningsfunktion:

```
signature MONOIDWPOW
= sig include MONOID
      val pow : t * int -> t (* pow(a,n) = mul(mul(...mul(one,a)...,a),a) *)
                             (*            n forekomster af a             *)
      end
```

Example 17.2 arbejder med multiplikation inden for typen `real`, svarende til strukturen `RealMult`:

```
structure RealMult : MONOID
= struct type t = real
         val mul = op *
         val one = 1.0
  end
```

(a) Definer en funktor `RepSquare` med hovedet

```
functor RepSquare (structure S : MONOID) : MONOIDWPOW
= struct
        (* skriv mig! *)
  end
```

som konstruerer sin potensopløftningsfunktion ud fra `S.one` og `S.mul` med metoden fra Example 17.2.

Når definitionen er gennemført, vil lærebogens eksempel `power2(2.0,10)` kunne gengives via

```
structure RealPow = RepSquare (structure S = RealMult);
RealPow.pow(2.0,10)
```

(b) Definer også en struktur `StringConc` for sammenkædning af tekststrenge, der kan bruges som argument til `RepSquare`, sådan at for eksempel en seks-dobbelt gentagelse som `"ja!"^"ja!"^"ja!"^"ja!"^"ja!"^"ja!"` kan beregnes med kun fire `^`-operationer.

5 Opdateringer

5.1 Opdatering (2011-11-02, 12:00)

- Der er en fejl i eksemplet i opgave 4(d). Resultatmatricen skal være,

$$\begin{pmatrix} 26 & 23 \\ 23 & 29 \end{pmatrix}$$

og *ikke*, som angivet i opgaveformuleringen,

$$\begin{pmatrix} 26 & 23 \\ 23 & 26 \end{pmatrix}.$$

5.2 Opdatering (2011-11-02, 17:00)

- Angående opgave 2: Uddybende forklåring af, hvad en nedre trekant er.

  En nedre trekant er en liste af lister, hvor den første liste har længde 1, den anden liste har længde 2, etc. Det kan visualiseres således:

```
[[ 1.2],
 [~3.5,  4.1],
 [ 5.2,  6.0,  0.7],
 [ 3.9, ~1.4,  2.0,  8.8]]
```

  Bemærk at imens

```
[[ 1.2,  0.0,  0.0,  0.0],
 [~3.5,  4.1,  0.0,  0.0],
 [ 5.2,  6.0,  0.7,  0.0],
 [ 3.9, ~1.4,  2.0,  8.8]]
```

  er en *nedre-trekantsmatrix* i normale sammenhænge, så er den *ikke* en nedre trekant i opgavens sammenhæng.

- Angående brugen af Moscow ML funktionen `Listsort.sort`:

  Lad os prøve at kigge på typen. I MosML:

```
- Listsort.sort;
> val 'a it = fn : ('a * 'a -> order) -> 'a list -> 'a list
```

  Altså: det er en funktion som givet "noget" returnerer en operation på lister. Vi ved fra dokumentationen at den operation er sortering. Så hvad er "noget"? Typen er `'a * 'a -> order`, så lad os kigge på `order`. Det er en datatype i standardbiblioteket som er defineret således:

```
datatype order = LESS | EQUAL | GREATER
```

  Man kan vel gætte betydningen. Lad os som eksempel definere typen af tal som indeholder både `real` og `int`:

```
datatype tal = Real of real | Int of int
fun fun add (Int a, Int b) = Int (a + b)
  | add (Real a, Real b) = Real (a + b)
  | add (Int a, Real b) = Real (real a + b)
  | add (Real a, Int b) = Real (a + real b)

...
```

Hvis nu vi vil kunne sortere tal skal vi definere en funktion med type `tal * tal -> order`:

```
fun compare (Int a, Int b) = Int.compare (a, b)
  | compare (Real a, Real b) = Real.compare (a, b)
  | compare (Int a, Real b) = Real.compare (real a, b)
  | compare (Real a, Int b) = Real.compare (a, real b)
```

Nu kan vi erklære vores sorteringsfunktion. I MosML:

```
- val talsort = Listsort.sort compare;
> val talsort = fn : tal list -> tal list
- val a = Int 42
  val b = Real 7.0;
> val a = Int 42 : tal
  val b = Real 7.0 : tal
- talsort [a, b];
> val it = [Real 7.0, Int 42] : tal list
```

Og til sidst et vink: Kend dit standardbibliotek (`http://www.standardml.org/Basis/`) og MosMLs bibliotek (`http://bit.ly/mosmllib`).

- Angående opgave 3: De omformulerede funktioner skal hedde `f`, `g` og `h`, lige som de oprindelige funktioner, som ikke skal afleveres.
- Angående opgave 4(b): At `dim` har den angivne type betyder, at SML uden fejlmelding accepterer en indgangslinje af form

  ```
  dim : 'a matrix -> int * int
  ```

  I sine typesvar benytter SML imidlertid ikke det indførte typenavn, så "ekkoet"fra den viste linje vil blive

  ```
  val it = fn : 'a list list -> int * int
  ```

5.3 Opdatering (2011-11-03, 13:00)

- Ogavebesvarelsen skal affattes i SML samt på enten engelsk eller dansk.
- Hvis kravet til en funktion er, at den skal have en bestemt type, så behøver den ikke være mere polymorf, selv hvis det skulle være muligt at finde sådan en løsning.
- Hvis man vil referere til en funktion i en struktur uden brugen af dot-notationen og uden at åbne hele strukturen, er dette muligt ved en `val`-erklæring; f.eks.

```
val toString = Int.toString
```

- Det er tilladt at anvende funktioner fra gruppe/individuelle ugeopgaver til løsning af examenssættet, så længe dette eksplicit angives i besvarelsen.

- Som del af bedømmelsen bliver den funktion, man afleverer, afprøvet af instruktorerne. Man behøver ikke at aflevere egne testeksempler, medmindre det er et eksplicit krav i opgaven. Korrekt udformet afprøvning efterladt i filen ved aflevering tæller ikke ned, såfremt koden er korrekt udformet MosML. Dette vil bl.a. sige, at IO-afprøvning i forbindelse med opgave 7 skal udkommenteres.

- Angående opgave 1:

  Ligesom binomialkoefficienterne er Stirlings tal af anden art nogle tal, der afhænger af to parametre, men det er nogle andre tal end binomialkoefficienterne — derfor den anderledes skrivemåde med krøllede parenteser, hvor man plejer at bruge rund parentes i binomialkoefficienter. Der findes også andre notationer for Stirlings tal af anden art; nogle steder skrives de for eksempel $S_{n,k}$. Det væsentlige er, at de afhænger af de to parametre på den måde, det er beskrevet i opgaven.

  Man skal vise de første 5 evalueringstrin, hvis antallet af alle trin er større eller lige med 5. Hvis dette antal er mindre end 5, så skal man vise alle evalueringstrin. Simplifikation af aritmetiske udtryk gælder ikke som særskilt evalueringstrin. Dette betyder for eksempel, at, istedet for at skrive `17 - 1`, skal man skrive `16`, når man skriver resultatet af et trin ned.

- Angående opgave 5: En forklaring af Burrows Wheeler Transformationen findes på Wikipedia: `http://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform`. (Det er ikke nødvendigt at angive denne URL som kilde i besvarelsen.)

- Angående opgave 7: Tegnet *carriage return* i type `char` skrives `#"\r"` i SML. Dette repræsenterer et *enkelt* tegn, selvom nedskrivningen i SML ser længere ud. Det samme gør sig gældende for tegnet *newline*, som skrives `#"\n"` i SML.

  Funktionen `wc` skal returnere antallet af tegn, der bliver læst ind ved hjælp af inputfunktioner i strukturen `TextIO`. Det kan give *forskellige resultater*, afhænigt af operativsystem. *Men det er sagen uvedkommende:* Funktionen `wc` skal tælle det antal tegn, der bliver læst ind på det operativsystem, det bliver udført på.

(Opgavesættet slut)