Introduktion til programmering, ugeseddel 2

Version 1.2

9. september 2014

Den anden undervisningsuge har til formål at gøre dig fortrolig med programmering i SML med brug af tupler (sæt) og lister.

Der er i uge 2 en obligatorisk opgave, som består af en gruppeopgave og en individuel opgave.

1 Pensum og plan for ugen

Pensum for uge 2: HR afsnit 3.3 samt kapitel 5 og 6. IP-2 afsnit 4.3–4.4, 4.6 (undtagen 4.6.1), 4.7, 5.1–5.4, 5.6, kapitel 6, afsnit 7.2–7.4.1. Noterne "Hvordan man angriber et programmeringsproblem" og "Vejledning om test" (findes under "Undervisningsmateriale" på Absalon).

Foreslået læserækkefølge: HR afsnit 3.3, IP-2 afsnit 4.3–4.4, 4.6, 5.1–5.4, 5.6, "Vejledning om test", HR kapitel 5, IP-2 kapitel 5, HR kapitel 6, "Hvordan man angriber et programmeringsproblem", (til fredag) IP-2 afsnit 7.1–7.4.1.

Forelæsningen mandag 8:15 – ca. 8:50 repeterer stoffet fra uge 1. Forelæsningen 09:00–10:00 vil fokusere på tupler (sæt), polymorfi og afprøvning. Øvelserne vil omhandle det samme og inddrage elementer fra uge 1.

Tirsdag omhandler både forelæsninger og øvelser lister samt systematisk problemløsning.

Forelæsningen fredag omhandler køretid. Øvelserne bruges til at arbejde med den individuelle opgave.

2 Mandagsopgaver

Mål: Fortrolighed med selekteringsfunktioner, erklæring af infix operatorer samt systematisk afprøvning.

2M1 HR opgave 3.3 (a) og (b). Lav test som beskrevet i "Vejledning om test". **Vink:** Se prædefinerede operator præcedens på side 316 i HR.

2M2 IP-2 opgave 4.1 og 4.4. Lav test som beskrevet i "Vejledning om test".

2M3 Omskriv gcd-funktionen fra side 26 i HR (den, der bruger if-then-else), så den tager ét argument `mn` af typen `int * int` og bruger selektionsfunktionerne `#1` og `#2` til at tilgå komponenterne i parret `mn`. Vurder læseligheden af programmet sammenlignet med de versioner af gcd, der er vist på side 20 og 26 i HR.

2M4 IP-2 opgave 4.6.

3 Tirsdagsopgaver

Mål: Programmering med lister og systematisk problemløsning.

Det forventes, at du inden øvelserne tirsdag har forberedt dig på opgaverne ved at løse så mange som muligt på egen hånd.

2T1 HR opgave 5.2, 5.3, og 5.7.

2T2 I ASCII alfabetet (HR side 346 og IP-2 side 17) kommer alle de store bogstaver før alle de små bogstaver, så sammenligningen `"XYZ" < "abc"` giver værdien `true`, hvilket ikke er konsistent med den måde, man ordner ord i en ordbog.

(a) Beskriv i tekst, hvordan ord er ordnet i en ordbog.

(b) Beskriv i tekst en plan for, hvordan man kan implementere denne ordning i SML og beskriv et antal tests, der med rimelighed kan overbevise en læser om, at denne funktion fungerer rigtigt. Tænk over specialtilfælde, og overvej, om der er funktioner i modulerne `Char` og `String` (Appendix D3 i HR), som kunne være brugbare til at løse problemet. Du behøver ikke at tage hensyn til danske bogstaver (æ, ø og å), accenter eller til rækkefølgen af ikke-alfabetiske tegn i ordbøger, og ejheller, at aa nogen gange alfabetiseres efter z. Med andre ord, er det kun ord, der er bygget af bogstaverne a til z og A til Z, du skal kunne teste for ordbogsorden.

(c) Skriv i SML en funktion `ordbogsOrden : string * string -> bool`, sådan at kaldet `ordbogsOrden (`p`,`q`)` giver `true`, hvis ordet p kommer før ordet q i ordbogsordenen.

(d) Udfør den planlagte test og reflekter over resultatet. Blev der fundet fejl? Var der flere ting, man med rimelighed havde kunnet afprøve?

(e) Reflekter over processen og resultatet: Havde det hjulpet at bruge mere tid i de første faser? Kan funktionen programmeres pænere?

2T3 HR opgave 5.17

4 Opgavetema: Geografiske koordinater

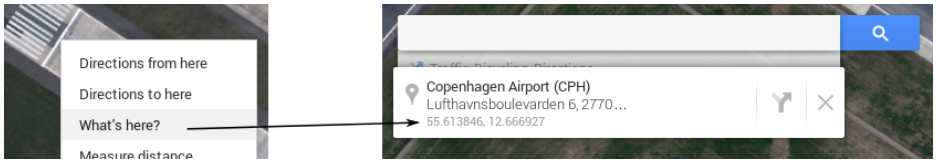
Den præcise angivelse af et punkt på jordens overflade kan defineres ud fra en *breddegrad*, der angiver hvor langt mod syd eller nord punktet er i forhold til ækvator, og en *længdegrad*, der angiver hvor langt øst/vestligt punktet er i forhold til *Royal Observatory Greenwich* i London.

De to tal opgives i et antal grader: En breddegrad er et *reelt* tal mellem $-90.0°$ (Sydpolen) og $90.0°$ (Nordpolen), og en længdegrad er et *reelt* tal mellem $-180.0°$ og $180.0°$. Breddegraden $0.0°$ er ved ækvator, og længdegraden $0.0°$ er ved føromtalte observatorium.

I SML kan vi altså definere et sådant geografisk koordinat som et par af to reelle tal (breddegrad og længdegrad). F.eks. kan vi definere koordinaterne til Universitetsparken 1 og koordinaterne til Peterskirken i Rom:

```
val diku_up1 = (55.702028, 12.561144)
val peterskirken = (41.902139, 12.453336)
```

For at verificere et koordinat kan i taste det ind på formen „55.702028, 12.561144“ i søgefeltet på Google Maps (`http://maps.google.dk`). Med Google Maps kan I også aflæse andre koordinater ved at højreklikke og vælge „Hvad er der her?“ eller „What's here?“ i den engelsksprogede udgave. At udvælge jeres egne koordinater på denne måde kan være anvendeligt når I skal skrive afprøvning af jeres funktioner.



Desværre er jorden ikke flad, og derfor er det ikke helt så simpelt at beregne afstande mellem to punkter på jordoverfladen, da man skal tage højde for jordens krumning og det noget anderledes koordinatsystem. Vi udleverer derfor en funktion, der kan beregne afstande (i kilometer), som I skal bruge til de efterfølgende opgaver. Funktionen hedder `distance` og findes i filen `uge2_distance.sml`, der ligger sammen med ugesedlen på Absalon. I kan enten kopiere indholdet ind i jeres aflevering eller tilføje en `use`-erklæring i starten af filen:

```
use "uge2_distance.sml";    (* Husk semikolonnet! *)
```

For at finde afstanden (kilometer i fugleflugt) mellem Universitetsparken 1 og Peterskirken skrives:

```
- distance (diku_up1, peterskirken);
> val it = 1534.49747481 : real
```

I må ikke blive bange for at bruge den slags funktioner i ikke helt forstår. Dataloger vil altid skulle arbejde tæt sammen med andre specialister. I skal vænne jer til at man ikke behøver at forstå, hvordan alle dele af et program helt konkret er sat sammen, når man bare kan forstå, hvad de gør (og stoler på at de gør det der er blevet lovet).

5 Gruppeaflevering

I uge 2 og fremover er gruppeafleveringen obligatorisk. Alle delspørgsmål skal besvares. Opgaven afleveres i Absalon. Der afleveres en fil pr. gruppe, men den skal angive alle deltageres fulde navne i kommentarlinjer øverst i filen. Filens navn skal være af formen `2G-`*`initialer`*`.sml`, hvor initialer er erstattet af gruppemedlemmernes initialer. Hvis f.eks. Bill Gates, Linus Torvalds, Steve Jobs og Gabe Logan Newell afleverer en opgave sammen, skal filen hedde `2G-BG-LT-SJ-GLN.sml`. Brug gruppeafleveringsfunktionen i Absalon.

Gruppeopgaven giver op til 2 point, som tæller til de 20 point, der kræves for eksamensdeltagelse. Genaflevering kan hæve pointtallet fra første aflevering med højest 1 point, så sørg for at gøre jeres bedste allerede i første aflevering.

NB! Der skal laves systematisk afprøvning af alle opgaverne i gruppeopgaven jvf. noten "Vejledning i test".

2G1 En løberute kan beskrives som en liste af koordinater, hvor det første er startkoordinatet, og det sidste er endepunktet på ruten. Vi har vedlagt en rute i dette format i filen `uge2_cphmarathon2014.sml`:

```
val cphMarathon2014 = [(55.66639000,12.57621000),
                       (55.66794000,12.57822000),
                       (55.66858000,12.57914000),
                       ...
                       (55.66659000,12.57647000)];
```

Skriv en funktion der måler den samlede afstand af ruten (i kilometer) vha. `distance`-funktionen beskrevet ovenfor. Funktionen skal have følgende typesignatur: `totalDistance : (real * real) list -> real`

Maratonruten er ikke helt nøjagtigt målt op af løberen, så i stedet for de 42,195 kilometer som maratondistancen er defineret til, skal I forvente følgende svar. I kan se bort fra afvigelser på de sidste decimaler.

```
- totalDistance cphMarathon2014;
> val it = 42.9686446838 : real
```

2G2 Når en løberute optages med et løbeur eller mobiltelefon med GPS-enhed gemmes ikke kun koordinaterne, men også tidspunkt. I vores repræsentation er et tidspunkt repræsenteret som et helt antal millisekunder siden starten af løbet. Vi har som eksempel vedlagt en rute med angivet tid i filen `uge2_dhl2014.sml`. Denne definerer en variabel `dhl2014 : ((real * real) * int) list`, hvor parret af kommatal er GPS-koordinaterne og heltallet er antallet af millisekunder siden start. Det første punkt på ruten er startpunktet, som følgelig har tiden 0.

Skriv en funktion der beregner hastigheden (i km/t) på hvert segment af en rute:

```
speeds : ((real * real) * int) list -> real list
```

Bemærk, at den resulterende liste er et element kortere end inddatalisten, da en sekvens af n punkter har $n-1$ mellemliggende segmenter. Hvis inddatalisten er tom, rejses undtagelsen `Empty`.

Eksempel: Hængebroen over Storebælt er ca. 6,8 km lang, hvis en løber skal løbe frem og tilbage over broen og han på vejen frem bruger 25 minutter (1500000 millisekunder) og på vejen tilbage bruger 35 minutter (2100000 millisekunder), vil datasættet se således ud (hvis løbeuret kun har registreret koordinater og tid i hver ende af broen):

```
val storebaelt = [((55.336145, 10.990714), 0),
                  ((55.349420, 11.095690), 1500000),
                  ((55.336145, 10.990714), 3600000)]
```

Gennemsnitshastigheden på hver af de to segmenter skal kunne beregnes til cirka:

```
- speeds storebaelt;
> val it = [16.3201496895, 11.6572497782] : real list
```

2G3 Skriv en funktion der returnerer gennemsnittet af tallene i en liste. Funktionen skal have følgende typesignatur: `averageAll : real list -> real`. Hvis listen er tom rejses undtagelsen `Empty`.

2G4 Brug funktionerne `speeds` og `averageAll` til at definere en funktion

```
averageSpeed : ((real * real) * int) list -> real
```

som finder gennemsnitshastigheden for en rute. Hvis ruten er tom, rejses undtagelsen `Empty`. Hvis ruten kun har et punkt, rejses undtagelsen `Domain`.

Forvent en gennemsnitshastighed på ca. 13.988 km/t for Storebælts-eksemplet.

2G5 Når jeres instruktor retter jeres opgaver, kommenterer de på blandt andet følgende punkter:

- Er funktionen korrekt: Har den den type, som opgaven krævede, og gør den det, der er krævet i opgaven?
- Er funktionen skrævet på en pæn og læselig måde: Er indrykningen passende, er der et passende omfang af informationsbærende kommenterarer? Er der ting, der kunne gøres enklere? Er der brugt fornuftig navngivning for funktioner og variabel navne.
- Tester afprøvningen alle relevante tilfælde uden at have overflødige testtilfælde?

Som svar på opgave 1I3 fra ugeseddel 1, har en elev indleveret følgende besvarelse:

```
(* hvis længden af s er w, returneres s, *)
(* ellers sættes et blanktegn foran, og der kaldes rekursivt *)
fun rightAlignHelper (s, w) = if String.size s = w then s else rightAlignHelper (" " ^ s, w)
(* Konverter tallet n til tekst og tilføj blanktegn foran, *)
(* indtil længden er w *)
fun rightAlign (n, 0) = raise Domain
```

```
  | rightAlign (n, w) = rightAlignHelper(Int.toString n, w)

val test_rightAlign1 = rightAlign(17,4) = "  17"
val test_rightAlign2 = rightAlign(~17,4) = " ~17"
```

Bedøm denne besvarelse ud fra de ovennævnte kriterier. Giv både ris og ros. Skriv jeres bedømmelse som en kommentar i den afleverede `.sml` fil.

6 Individuel aflevering

Også den individuelle opgave er obligatorisk i uge 2 og fremover. Alle delspørgsmål skal besvares. Opgaven afleveres i Absalon som en fil med navnet `2I-`*navn*`.sml`, hvor *navn* er erstatttet med dit navn. Hvis du fx hedder Anders A. And, skal filnavnet være `2I-Anders-A-And.sml`. Skriv også dit fulde navn som en kommentar i starten af filen.

Den individielle opgave giver op til 3 point, som tæller til de 20 point, der kræves for eksamensdeltagelse. Genaflevering kan hæve pointtallet fra første aflevering med højest 1 point, så sørg for at gøre dit bedste allerede i første aflevering.

Den individuelle opgave i denne uge fortsætter temaet om GPS-koordinater fra ugens gruppeopgave.

NB! Der skal også i den individuelle aflevering laves systematisk afprøvning af alle opgaverne jvf. noten "Vejledning i test".

2I1 I mange GPS-systemer er der mulighed for at finde det nærmeste interessepunkt. F.eks. „Nærmeste benzintank", „Nærmeste supermarked".

Skriv funktionen:

```
closestDistance : ((real * real) * (real * real) list) -> real
```

sådan at kaldet `closestDistance(p, qs)` beregner *afstanden* til det koordinat i listen `qs` som er tættest på koordinatet `p` i fugleflugtslinje. Hvis listen er tom, rejses undtagelsen `Empty`. Brug denne funktion til at finde den korteste afstand fra DIKU (Universitetsparken 1) til et punkt på ruten for Københavns maraton.

Vink: Brug `Real.min`.

2I2 Det er ikke altid man kun er interesseret i afstanden til det nærmeste punkt, men også noget information om selve punktet. I filen `uge2_kollegier.sml` har vi vedlagt listen:

```
val kollegier = [((55.662539, 12.609901), "Frankrigsgadekollegiet"),
                 ((55.654295, 12.592202), "Grønjordskollegiet"),
                 ((55.700325, 12.553688), "Kollegiegården"),
                 ...
                ]
```

Skriv funktionen:

```
closestPOI : ((real * real) * ((real * real) * 'a) list) -> 'a
```

der i stedet for at returnere *afstanden* til det nærmeste interessepunkt, skal I returnere *beskrivelsen* af interessepunktet.

Vink: Skriv en hjælpefunktion, der returnerer *både* koordinatet og beskrivelsen.

2I3 Bemærk, at typen af beskrivelsen i signaturen for `closestPOI` er angivet som `'a` og ikke som `string`. Forklar i en kommentar i din besvarelse, hvorfor dette er tilfældet.

7 Ugens nød

Ugens nød er en frivillig, særligt udfordrende opgave, der kan løses af særligt interesserede studerende. Opgaven kan løses kun ved brug af begreber, der allerede er undervist i, men kræver ekstra omtanke og nogen gange en god ide. Afleveringsfristen er den samme som for den individuelle opgave. Ved fredagsforelæsningen i den efterfølgende uge kåres den bedste løsning til ugens nød, og vinderen får en lille præmie, typisk et stykke chokolade eller andet guf.

Selvbeskrivende sætninger

En sætning af formen "Denne sætning ..." siges at være selvbeskrivende. En selvbeskrivende sætning kan være sand (som f.eks. "Denne sætning indeholder flere end ti tegn") eller falsk (som f.eks. "Denne sætning indeholder flere end hundrede tegn"). Den kan også have udefineret sandhedsværdi (som f.eks. "Denne sætning er falsk"), men hvis en sætning blot udtaler sig om antallet af tegn i sig selv, er den klart enten sand eller falsk.

2N1 Lav en funktion `talord : int -> string`, der tager et heltal $0 \leq n < 1000$ og returnerer en tegnfølge, der er talordet for n. F.eks. skal `talord 0` returnere tegnfølgen `"nul"` og `talord 345` skal returnere tegnfølgen `"trehundredefemogfyrre"`.

2N2 Lav et program `findSand : unit -> string`, der returnerer en sætning af formen `"Denne sætning indeholder præcis` n `bogstaver."`, hvor n er et talord, og sætningen er sand. Bemærk, at blanktegn og punktum ikke betragtes som bogstaver. Vis resultatet af kaldet `findSand ()`.

2N3 Lav et program `findSand2 : unit -> string`, der returnerer en sætning af formen `"Denne sætning indeholder præcis` v `vokaler og` k `konsonanter."`, hvor v og k er talord, og sætningen er sand. Vis resultatet af kaldet `findSand2 ()`.

2N4 Lav et program `findSand3 : unit -> string`, der returnerer en sætning af formen `"Denne sætning indeholder præcis` a `a'er, et b, et c,` d `d'er, ...,` y `y'er, et z, tre æ'er, et ø og et å."`, hvor $a, \ldots, y$ er talord, og sætningen er sand. Bemærk at, idet talord for tal under 1000 ikke bruger alle

bogstaver i alfabetet, er nogle af talordene givet på forhånd. F.eks. vil sætningen altid indeholde et b og tre æ'er. Bemærk også, at der ikke gøres forskel på store og små bogstaver, så initialen "D" tæller med til d'erne. Det er endvidere ikke korrekt at skrive f.eks. "et a'er", da det hedder "et a".

Vis resultatet af kaldet `findSand3 ()`.

Det er ikke sikkert, at nogen finder en helt korrekt løsning, så præmien gives til den, der finder den løsning, der er korrekt for flest mulige bogstaver. Man kan f.eks. lave en funktion, der tager et tal z som ekstra parameter og finder en løsning med mindst z rigtige talord. Som *tie breaker* bruges en vurdering af, hvor pænt programmet er skrevet.

NB! Mosml bruger tegnkodningen ISO-8859/1 (se side 20 i IP-2), hvor de fleste moderne programmer til tekstredigering og -visning bruger UTF-8. Du kan stille emacs til at bruge ISO-8859/1, men ellers er det helt i orden at erstatte æ, ø og å med andre tegn fra 7-bit ASCII alfabetet (side 17 i IP-2 og side 346 i HR), f.eks. &, % og @, så man skriver "`Denne s&tning indeholder ...et % og et @`".