# Programmering og Problemløsning
# Datalogisk Institut, Københavns Universitet
# Arbejdsseddel 7 - gruppeopgave

Jon Sporring

19. oktober - 9. november.
Afleveringsfrist: lørdag d. 9. november kl. 22:00.

Some introductory text ...

Emnerne for denne arbejdsseddel er:

- rekursion, pattern matching,
- sum-typer,
- endelige træer.

Opgaverne er opdelt i øve- og afleveringsopgaver. I denne periode skal I arbejde i grupper med jeres afleveringsopgaver. Regler for gruppe- og individuelle afleveringsopgaver er beskrevet i "'Noter, links, software m.m."'→"'Generel information om opgaver"'.

## Øveopgaver (in English)

7ø0 Betragt insertion sort funktionen `isort`. Omskriv funktionen `insert` således, at den benytter sig af pattern matching på lister.

7ø1 Betragt bubble sort funktionen `bsort`. Omskriv den, at den benytter sig af pattern matching på lister. Funktionen kan passende benytte sig af "nested pattern matching" i den forstand at den kan implementeres med et match case der udtrækker de to første elementer af listen samt halen efter disse to elementer.

7ø2 Opskriv black-box tests for de to sorteringsfunktioner og vær sikker på at grænsetilfældene dækkes (ingen elementer, et element, to elementer, samt flere elementer, sorteret, omvendt sorteret, etc.)

7ø3 Omskriv funktionen `merge`, som benyttes i forbindelse med funktionen `msort` (mergesort) fra forelæsningen, således at den benytter sig af pattern matching på lister.

7ø4 Opskriv black-box tests for sorteringsfunktionen `msort` og vær sikker på at grænsetilfældene dækkes (ingen elementer, et element, to elementer, samt flere elementer, sorteret, omvendt sorteret, etc.)

---

I det efterfølgende skal der arbejdes med sum-typen:

```
type weekday = Monday | Tuesday | Wednesday | Thursday
             | Friday | Saturday | Sunday
```

som repræsenterer ugens dage.

7ø5 Lav en funktion `dayToNumber : weekday -> int`, der givet en ugedag returnerer et tal, hvor mandag skal give tallet 1, tirsdag tallet 2 osv.

7ø6 Lav en funktion `nextDay : weekday -> weekday`, der givet en ugedag returnerer den næste dag, så mandag skal give tirsdag, tirsdag skal give onsdag, osv, og søndag skal give mandag.

7ø7 Lav en funktion `numberToDay : n : int -> weekday option`, sådan at `numberToDay n` returnerer `None`, hvis `n` ikke ligger i intervallet $1 \ldots 7$, og ellers returnerer ugedagen `Some d`. Det skal gælde, at `numberToDay (dayToNumber d)` $\rightsquigarrow$ `Some d` for alle ugedage `d`.

---

7ø8 Ved at benytte biblioteket `ImgUtil`, som beskrevet i forelæsningen, er det muligt at tegne simpel liniegrafik samt fraktaler, som f.eks. Sierpinski-fraktalen, der kan tegnes ved at tegne små firkanter bestemt af et rekursivt mønster. Koden for Sierpinski-trekanten er givet som følger:

```
open ImgUtil

let rec triangle bmp len (x,y) =
  if len < 25 then setBox blue (x,y) (x+len,y+len) bmp
  else let half = len / 2
       do triangle bmp half (x+half/2,y)
       do triangle bmp half (x,y+half)
       do triangle bmp half (x+half,y+half)

do runSimpleApp "Sierpinski" 600 600 (fun bmp -> triangle bmp
   512 (30,30) |> ignore)
```

Tilpas funktionen således at trekanten tegnes med røde streger samt således at den kun tegnes 2 rekursionsniveauer ned. Hint: dette kan gøres ved at ændre betingelsen `len < 25`.

7ø9 I stedet for at benytte `ImgUtil.runSimpleApp` funktionen skal du nu benytte `ImgUtil.runApp`, som giver mulighed for at din løsning kan styres ved brug af tastaturet. Funktionen `ImgUtil` har følgende type:

```
val runApp : string -> int -> int
          -> (int -> int -> 's -> System.Drawing.Bitmap)
          -> ('s -> System.Windows.Forms.KeyEventArgs
                 -> 's option)
          -> 's -> unit
```

De tre første argumenter til `runApp` er vinduets titel (en streng) samt vinduets initielle vidde og højde. Funktionen `runApp` er parametrisk over en brugerdefineret type af tilstande (`'s`). Antag at funktionen kaldes som følger:

```
runApp title width height draw react init
```

Dette kald vil starte en GUI applikation med titlen `title`, vidden `width` og højden `height`. Funktionen `draw`, som brugeren giver som 4. argument kaldes initielt når applikationen starter og hver gang vinduets størrelse justeres eller ved at funktionen `react` er blevet kaldt efter en tast er trykket ned på tastaturet. Funktionen `draw` modtager også (udover værdier for den aktuelle vidde og højde) en værdi for den brugerdefinerede tilstand, som initielt er sat til værdien `init`. Funktionen skal returnere et bitmap, som for eksempel kan konstrueres med funktionen `ImgUtil.mk` og ændres med andre funktioner i `ImgUtil` (f.eks. `setPixel`).

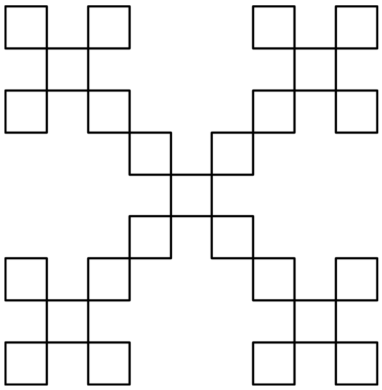
Funktionen `react`, som brugeren giver som 5. argument kaldes hver gang brugeren trykker på en tast. Funktionen tager som argument:

- en værdi svarende til den nuværende tilstand for applikationen, og
- et argument der kan benyttes til at afgøre hvilken tast der blev trykket på.[1]

Funktionen kan nu (eventuelt) ændre på dens tilstand ved at returnere en ændret værdi for denne.

Tilpas applikationen således at dybden af fraktalen kan styres ved brug af piletasterne, repræsenteret ved værdierne `System.Windows.Forms.Keys.Up` og `System.Windows.Forms.Keys.Down`.

7ø10 Med udgangspunkt i øvelsesopgave 7ø8 skal du i denne opgave implementere en GUI-applikation der kan tegne en version af X-fraktalen som illustreret nedenfor (eventuelt i en dybde større end 2).



Bemærk at det ikke er et krav, at dybden på fraktalen skal kunne styres med piletasterne, som det er tilfældet med Sierpinski-fraktalen i øvelsesopgave 7ø9.

# Afleveringsopgaver (in English)

Simple Jack er en forsimplet udgave af kortspillet Blackjack. I Simple Jack spiller man ikke om penge/jetoner men blot om sejr/tab mellem en spiller og dealer. Reglerne for Simple Jack er som følger:

[1] Hvis `e` har typen `System.Windows.Forms.KeyEventArgs` kan betingelsen `e.KeyCode = System.Windows.Forms.Keys.Up` benyttes til at afgøre om det var tasten "Up" der blev trykket på.

Spillet består af en dealer, 1-5 spillere samt et normalt kortspil (uden jokere). Ved spillets start får dealer og hver spiller tildelt 2 tilfældige kort fra bunken som placeres med billedsiden opad foran spilleren, så alle kan se dem. I Simple Jack spilles der med åbne kort dvs. alle trukne kort til hver en tid er synlige for alle spillere. Kortene har værdi som følger:

1. Billedkort (knægt, dame og konge) har værdien 10
2. Es kan antage enten værdien 1 eller 11
3. Resten af kortene har den påtrykte værdi

For hver spiller gælder spillet om at ende med en korthånd hvis sum af værdier er højere en dealers sum af værdier, uden at summen overstiger 21, i hvilket tilfælde spilleren er "bust". Spillerne får nu en tur hver, hvor de skal udføre en af følgende handlinger:

1. "Stand": Spilleren/dealeren vælger ikke at modtage kort og turen går videre.
2. "Hit": Spilleren/dealeren vælger at modtage kort fra bunken et ad gangen indtil han/hun vælger at stoppe og turen går videre.

Det er dealers tur til sidst efter alle andre spillere har haft deres tur. Når dealer har haft sin tur afsluttes spillet. Ved spillets afslutning afgøres udfaldet på følgende måde: En spiller vinder hvis ingen af følgende tilfælde gør sig gældende:

1. Spilleren er "bust"
2. Summen af spillerens kort-værdier er lavere end, eller lig med dealers sum af kort-værdier
3. Både spilleren og dealer har SimpleJack (SimpleJack er et Es og et billedkort)

Bemærk at flere spillere altså godt kan vinde på en gang. Et spil Simple Jack er mellem en spiller og dealer, så med 5 spillere ved bordet, er det altså 5 separate spil som spilles.

7g0 Design og implementér et program som kan simulere Simple Jack ved brug af klasser. Start med grundigt at overveje hvilke aspekter af spillet som giver mening at opdele i klasser. Spillet skal implementere således, at en spiller enten kan være en bruger af Simple Jack programmet, som foretager sine valg og ser kortene på bordet via terminalen, eller en spiller kan være en AI som skal følge en af følgende strategier:

   (a) Vælg altid "Hit", medmindre summen af egne kort kan være 17 eller over, ellers vælg "Stand"

   (b) Vælg tilfældigt mellem "Hit" og "Stand". Hvis "Hit" vælges trækkes et kort og der vælges igen tilfældigt mellem "Hit" og "Stand" osv.

   Dealer skal følge strategi nummer 1. Der skal også laves:

   - En rapport (maks 2 sider)
   - Unit-tests

- Implementation skal kommenteres jævnfør kommentarstandarden for F#

Hint: Man kan generere tilfældige tal indenfor et interval (f.eks. fra og med 1 til og med 100) ved brug af følgende kode:

```
let gen = System.Random()
let ran_int = gen.Next(1, 101)
```

# Krav til afleveringen

Afleveringen skal bestå af

- en zip-fil, der hedder `7g_<navn>.zip` (f.eks. `7g_jon.zip`)
- en pdf-fil , der hedder `7g_<navn>.pdf` (f.eks. `7g_jon.pdf`)

Zip-filen `7g_<navn>.zip` skal indeholde en og kun en mappe `7g_<navn>`. I den mappe skal der ligge en `src` mappe og filen `README.txt`. I `src` skal der ligge følgende og kun følgende filer: `textAnalysis.fs` og `testTextAnalysis.fs` svarende til hver af delopgaverne. De skal kunne oversættes med `fsharpc`, og de oversatte filer skal kunne køres med mono. Funktioner skal dokumenteres ifølge dokumentationsstandarden som minimum ved brug af `<summary>`, `<param>` og `<returns>` XML-tagsne. Filen `README.txt` skal ganske kort beskrive, hvordan koden oversættes og køres. Pdf-filen skal indeholde jeres rapport oversat fra LaTeX. Husk at pdf-filen skal uploades ved siden af zip-filen på Absalon.

God fornøjelse.