Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 7 — gruppeopgave

Jon Sporring

24. oktober – 21. november.
Afleveringsfrist: onsdag d. 21. november kl. 22:00

I denne periode skal I arbejde i grupper. Formålet er at arbejde med sumtyper og endelige træer. Opgaverne er delt i øve- og afleveringsopgaver. I denne periode skal I arbejde i grupper. Formålet er at arbejde med:

- rekursion
- pattern matching
- sumtyper
- endelige træer

Øveopgaver

7ø.0 Omskriv funktionen `insert`, som benyttes i forbindelse med funktionen `isort` (insertion sort) fra forelæsningen, således at den benytter sig af pattern matching på lister.

7ø.1 Omskriv funktionen `bsort` (bubble sort) fra forelæsningen således 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.

7ø.2 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.)

7ø.3 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 ned til dybde 2 (hint: du skal ændre betingelsen `len < 25`).

7ø.4 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 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 samt 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`.

[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.

I de næste to opgaver skal følgende sum-type benyttes til at repræsentere ugedage:

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

7ø.5 Lav en funktion `dayToNumber : weekday -> int`, der givet en ugedag returnerer et tal, hvor mandag skal give tallet 1, tirsdag tallet 2 osv.

7ø.6 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.

Afleveringsopgave

I denne opgave skal I programmere spillet Awari, som er en variant af Kalaha. Awari er et gammelt spil fra Afrika, som spilles af 2 spillere, med 7 pinde og 36 bønner. Pindene lægges så der dannes 14 felter ('pits' på engelsk), hvoraf 2 er hjemmefelter. Bønnerne fordeles ved spillet start med 3 i hvert felt pånær i hjemmefelterne. Startopstillingen er illustreret i Figur 1.

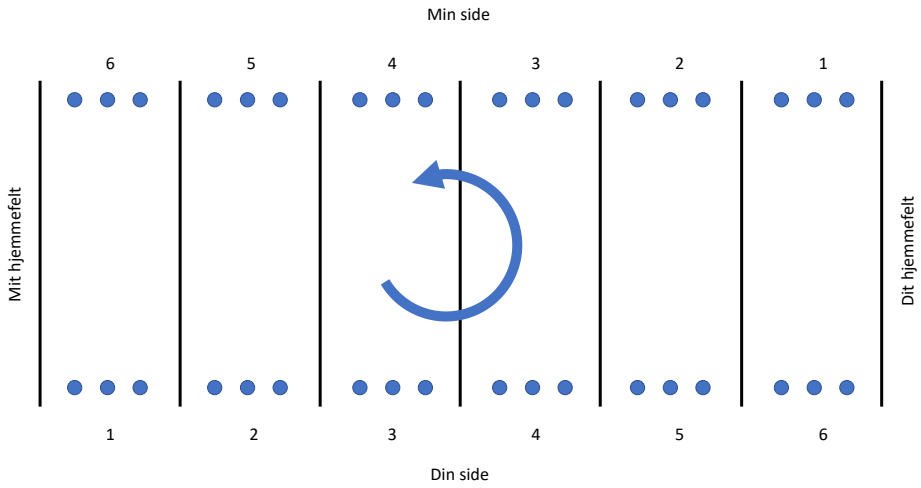

Figure 1: Udgangsopstillingen for spillet Awari.

Spillerne skiftes til at spille en tur efter følgende regler:

- En tur spilles ved at spilleren tager alle bønnerne i et af spillerens felter 1-6 og placerer dem i de efterfølgende felter inkl. hjemmefelterne en ad gangen og mod uret. F.eks., kan første spiller vælge at tage bønnerne fra felt 4, hvorefter spilleren skal placere en bønne i hver af felterne 5, 6 og hjemmefeltet.
- Hvis sidste bønne lægges i spillerens hjemmefelt, får spilleren en tur til.

- Hvis sidste bønne lander i et tom felt som ikke er et hjemmefelt, og feltet overfor indeholder bønner, så flyttes sidste bønne til spillerens hjemmefelt, og alle bønnerne overfor fanges og flyttes ligså til hjemmefeltet.
- Spillet er slut når en af spillerne ingen bønner har i sine felter 1-6, og vinderen er den spiller, som har flest bønner i sit hjemmefelt.

Afleveringsopgaven er:

7g.0 I skal implementere spillet Awari, som kan spilles af 2 spillere, og skrive en kort rapport. Kravene til jeres aflevering er:

- Koden skal organiseres som bibliotek, en applikation og en test-applikation.
- Biblioteket skal tage udgangspunkt i følgende signatur- og implementationsfiler:

Listing 1 awariLibIncompleteLowComments.fsi: En ikke færdigskrevet signaturfil.

```
module Awari
type pit = // intentionally left empty
type board = // intentionally left empty
type player = Player1 | Player2

/// Print the board
val printBoard : b:board -> unit

/// Check whether a pit is the player's home
val isHome : b:board -> p:player -> i:pit -> bool

/// Check whether the game is over
val isGameOver : b:board -> bool

/// Get the pit of next move from the user
val getMove : b:board -> p:player -> q:string -> pit

/// Distributing beans counter clockwise,
/// capturing when relevant
val distribute :
    b:board -> p:player -> i:pit -> board * player * pit

/// Interact with the user through getMove to perform
/// a possibly repeated turn of a player
val turn : b:board -> p:player -> board

/// Play game until one side is empty
val play : b:board -> p:player -> board
```

Listing 2 awariLibIncomplete.fs:
En ikke færdigskrevet implementationsfil.

```
module Awari
type pit = // intentionally left empty
type board = // intentionally left empty
type player = Player1 | Player2

// intentionally many missing implementations and additions

let turn (b : board) (p : player) : board =
  let rec repeat (b: board) (p: player) (n: int) : board =
    printBoard b
    let str =
      if n = 0 then
        sprintf "Player %A's move? " p
      else
        "Again? "
    let i = getMove b p str
    let (newB, finalPitsPlayer, finalPit)= distribute b p i
    if not (isHome b finalPitsPlayer finalPit)
       || (isGameOver b) then
      newB
    else
      repeat newB p (n + 1)
  repeat b p 0

let rec play (b : board) (p : player) : board =
  if isGameOver b then
    b
  else
    let newB = turn b p
    let nextP =
      if p = Player1 then
        Player2
      else
        Player1
    play newB nextP
```

En version af signaturfilen med yderligere dokumentation og implementationsfilen findes i Absalon i opgaveområdet for denne opgave.

- Jeres løsning skal benytte funktionsparadigmet såvidt muligt.
- Koden skal dokumenteres vha. kommentarstandarden for F#
- Jeres aflevering skal indeholde en afprøvning efter white-box metoden.
- I skal skrive en kort rapport i LaTeX på maks. 10 sider og som indeholder:
  - en beskrivelse af jeres design og implementation
  - en gennemgang af jeres white-box afprøvning
  - kildekoden som appendiks.

Afleveringen skal bestå af en zip-fil og en pdf-fil. Zip-filen skal indeholde en mappe med fsharp koden. Koden skal kunne oversættes med fsharpc og køres med mono. I skal tilføje en README.txt fil, hvor I beskriver, hvordan man oversætter og kører programmet. Pdf-filen skal indeholde jeres LaTeX rapport oversat til pdf.