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

Martin Elsman

23. oktober – 1. november.
Afleveringsfrist: onsdag d. 1. november kl. 22:00

I denne periode skal I arbejde individuelt. Regler for individuelle afleveringsopgaver er beskrevet i "Noter, links, software m.m." → "Generel information om opgaver". Formålet er at arbejde med:

- rekursion
- pattern matching

Opgaverne er delt i øve- og afleveringsopgaver.

## Øveopgaver

I de følgende opgaver skal vi arbejde med en træstruktur til at beskrive geometriske figurer med farver. For at gøre det muligt at afprøve jeres opgaver skal I gøre brug af det udleverede bibliotek `img_util.dll`, der blandt andet kan omdanne såkaldte bitmap-arrays til png-filer. Biblioteket er beskrevet i forelæsningerne (i uge 6) og koden for biblioteket ligger sammen med forelæsningsplancherne for uge 6. Her bruger vi funktionerne til at konstruere et bitmap-array samt til at gemme arrayet som en png-fil:

```
// colors
type color = System.Drawing.Color
val fromRgb : int * int * int -> color
// bitmaps
type bitmap = System.Drawing.Bitmap
val mk       : int -> int -> bitmap
val setPixel : color -> int * int -> bitmap -> unit

// save a bitmap as a png file
val toPngFile : string -> bitmap -> unit
```

Funktionen `toPngFile` tager som det første argument navnet på den ønskede png-fil (husk extension). Det andet argument er bitmap-arrayet som ønskes konverteret og gemt. Et bitmap-array kan konstrueres med funktionen `ImgUtil.mk`, der tager som argumenter vidden og højden af billedet i antal pixels, samt funktionen `ImgUtil.setPixel`, der kan bruges til at opdatere bitmap-arrayet før det eksporteres til en png-fil. Funktionen `ImgUtil.setPixel` tager tre argumenter. Det første argument repræsenterer en farve og det andet argument repræsenterer et punkt i bitmap-arrayet (dvs. i billedet). Det tredie argument repræsenterer det bitmap-array, der skal opdateres. En farve kan nu konstrueres med funktionen `ImgUtil.fromRgb` der tager en triple af tre tal mellem 0 og 255 (begge inklusive), der beskriver hhv. den røde, grønne og blå del af farven.

Koordinaterne starter med $(0,0)$ i øverste venstre hjørne og $(w-1,h-1)$ i nederste højre hjørne, hvis bredde og højde er hhv. $w$ og $h$. Antag for eksempel at programfilen `testPNG.fsx` indeholder følgende F# kode:

```
let bmp = ImgUtil.mk 256 256
do ImgUtil.setPixel (ImgUtil.fromRgb (255,255,0)) (10,10) bmp
do ImgUtil.toPngFile "test.png" bmp
```

Det er nu muligt at generere en png-fil med navn `test.png` ved at køre følgende kommando:

```
fsharpi -r img_util.dll testPNG.fsx
```

Den genererede billedfil `test.png` vil indeholde et sort billede med et pixel af gul farve i punktet (10,10).

Bemærk, at alle programmer, der bruger `ImgUtil` skal køres eller oversættes med `-r img_util.dll` som en del af kommandoen. Bemærk endvidere, at LaTeX kan inkludere png-filer med kommandoen `includegraphics`.

I det følgende vil vi repræsentere geometriske figurer med følgende datastruktur:

```
type point = int * int // a point (x, y) in the plane
type colour = int * int * int  // (red, green, blue), 0..255

type figure =
  | Circle of point * int * colour
     // defined by center, radius, and colour
  | Rectangle of point * point * colour
     // defined by corners bottom-left, top-right, and colour
  | Mix of figure * figure
     // combine figures with mixed colour at overlap
```

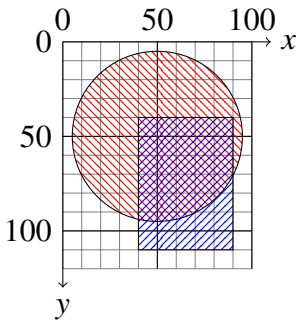
For eksempel kan man lave følgende funktion til at finde farven af en figur i et punkt. Hvis punktet ikke ligger i figuren, returneres `None`, og hvis punktet ligger i figuren, returneres `Some` $c$, hvor $c$ er farven.

```
// finds colour of figure at point
let rec colourAt (x,y) figure =
  match figure with
  | Circle ((cx,cy), r, col) ->
      if (x-cx)*(x-cx)+(y-cy)*(y-cy) <= r*r
        // uses Pythagoras' formular to determine
```

```
        // distance to center
      then Some col else None
| Rectangle ((x0,y0), (x1,y1), col) ->
    if x0<=x && x <= x1 && y0 <= y && y <= y1
       // within corners
    then Some col else None
| Mix (f1, f2) ->
     match (colourAt (x,y) f1, colourAt (x,y) f2) with
     | (None, c) -> c  // no overlap
     | (c, None) -> c  // no overlap
     | (Some (r1,g1,b1), Some (r2,g2,b2)) ->
        // average color
        Some ((r1+r2)/2, (g1+g2)/2, (b1+b2)/2)
```

Bemærk, at punkter på cirklens omkreds og rektanglens kanter er med i figuren. Farver blandes ved at lægge dem sammen og dele med to, altså finde gennemsnitsfarven.

8ø.1 Lav en figur `figTest : figure`, der består af en rød cirkel med centrum i (50,50) og radius 45, samt en blå rektangel med hjørnerne (40,40) og (90,110), som illustreret i tegningen nedenfor (hvor vi dog har brugt skravering i stedet for udfyldende farver.)




8ø.2 Brug `ImgUtil`-funktionerne og `colourAt` til at lave en funktion

```
makePicture : string -> figure -> int -> int
           -> unit
```

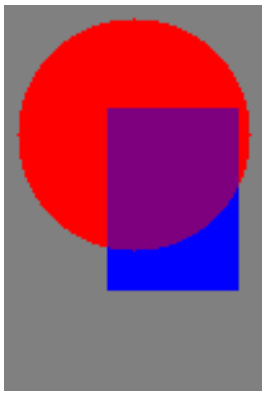
sådan at kaldet `makePicture` *`filnavn figur b h`* laver en billedfil ved navn *`filnavn`*`.png` med et billede af *`figur`* med bredde *`b`* og højde *`h`*.

På punkter, der ingen farve har (jvf. `colourAt`), skal farven være grå (som defineres med RGB-værdien (128,128,128)).

Du kan bruge denne funktion til at afprøve dine opgaver.

8ø.3 Lav med `makePicture` en billedfil med navnet `figTest.png` og størrelse $100 \times 150$ (bredde 100, højde 150), der viser figuren `figTest` fra opgave 8ø.2.

Resultatet skulle gerne ligne figuren nedenfor.



8ø.4 Lav en funktion `checkFigure : figure -> bool`, der undersøger, om en figur er korrekt: At radiusen i cirkler er ikke-negativ, at øverste venstre hjørne i en rektangel faktisk er ovenover og til venstre for det nederste højre hjørne (bredde og højde kan dog godt være 0), og at farvekompenterne ligger mellem 0 og 255.

Vink: Lav en hjælpefunktion `checkColour : colour -> bool`.

8ø.5 Lav en funktion `move : figure -> int * int -> figure`, der givet en figur og en vektor flytter figuren langs vektoren.

Ved at foretage kaldet

```
makePicture "moveTest" (move figTest (-20,20)) 100 150
```

shulle der gerne laves en billedfil `moveTest.png` med indholdet vist nedenfor.

8ø.6 Lav en funktion `boundingBox : figure -> point * point`, der givet en figur finder hjørnerne (top-venstre og bund-højre) for den mindste akserette rektangel, der indeholder hele figuren.

`boundingBox figTest` skulle gerne give `((5, 5), (95, 110))`.

# Afleveringsopgaver

8i.3 Du skal i de følgende to opgaver arbejde med en funktion til at bestemme den såkaldte *Levensthein-distance* mellem to strenge $a$ og $b$. Distancen er defineret som det mindste antal editeringer, på karakter-niveau, det er nødvendigt at foretage på strengen $a$ før den resulterende streng er identisk med strengen $b$. Som editeringer forstås (1) sletninger af karakterer, (2) indsættelser af karakterer, og (3) substitution af karakterer.

Varianter af Levensthein-distancen mellem to strenge kan således benyttes til at identificere om studerende selv har løst deres indleverede opgaver eller om der potentielt set er tale om plagiatkode ;)

Matematisk set kan Levensthein-distancen $leven(a,b)$, mellem to karakterstrenge $a$ og $b$, defineres som $lev_{a,b}(|a|,|b|)$, hvor $|a|$ og $|b|$ henviser til længderne af henholdsvis $a$ og $b$, og hvor funktionen $lev$ er defineret som følger:[1]

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

hvor $1_{(a_i \neq b_j)}$ henviser til *indikatorfunktionen*, som er 1 når $a_i \neq b_j$ og 0 ellers.

Implementér funktionen *leven* direkte efter den matematiske definition (ved brug af rekursion) og test korrektheden af funktionen på nogle små strenge, såsom "house" og "horse" (distance 1) samt "hi" og "hej" (distance 2).

8i.4 Den direkte implementerede rekursive funktion er temmelig ineffektiv når strengene $a$ og $b$ er store. F.eks. tager det en del millisekunder at udregne distancen mellem strengene "dangerous house" and "danger horse". Årsagen til denne ineffektivitet er at en løsning der bygger direkte på den rekursive definition resulterer i en stor mængde genberegninger af resultater der allerede er beregnet.

For at imødekomme dette problem skal du implementere en såkaldt "caching mekanisme" der har til formål at sørge for at en beregning højst foretages en gang. Løsningen kan passende gøre brug af gensidig rekursion og tage udgangspunkt i løsningen for den direkte rekursive definition (således skal løsningen nu implementeres med to gensidigt rekursive funktioner `leven` og `leven_cache` forbundet med `and`). Som cache skal der benyttes et 2-dimensionelt array af størrelse $|a| \times |b|$ indeholdende heltal (initielt sat til $-1$).

Funktionen `leven_cache`, der skal tage tilsvarende argumenter som `leven`, skal nu undersøge om der allerede findes en beregnet værdi i cachen, i hvilket tilfælde denne værdi returneres. Ellers skal funktionen `leven` kaldes og cachen opdateres med det beregnede resultat. Endelig er det nødvendigt at funktionen `leven` opdateres til nu at kalde funktionen `leven_cache` i hver af de rekursive kald.

Test funktionen på de små strenge og vis at funktionen nu virker korrekt også for store input.

Det skal til slut bemærkes at den implementerede løsning benytter sig af $O(|a| \times |b|)$ plads og at der findes effektive løsninger der benytter sig af mindre plads ($O(\max(|a|,|b|))$). Det er ikke et krav at din løsning implementerer en af disse mere pladsbesparende strategier.

Afleveringsopgaven skal afleveres som et antal `fsx` tekstfiler navngivet efter opgaven, som f.eks. `8i0.fsx`. Tekstfilerne skal kunne oversættes med `fsharpc`,[2] og resultatet skal kunne køres med `mono` eller eventuelt `mono32`. Funktioner skal dokumenteres ifølge dokumentationsstandarden, og udover selve programteksten skal besvarelserne indtastes som kommentarer i de `fsx`-filer, de hører til. Det hele skal samles i en zip fil og uploades på Absalon.

[1] See `https://en.wikipedia.org/wiki/Levenshtein_distance`.

[2] Løsningerne kan antage at biblioteket `ImgUtil.dll` fra forelæsningerne er tilgængeligt.