Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 8 — gruppeopgave

Martin Elsman*

30. oktober – 29. november.
Afleveringsfrist: onsdag d. 29. 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.

Øveopgaver

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

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

8ø.0 Lav en funktion `dayToNumber : weekday -> int`, der givet en ugedag returnerer et tal, hvor mandag skal give tallet 1, tirsdag tallet 2 osv.

8ø.1 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.

I de følgende opgaver (både øveopgaver og afleveringsopgaver) 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
```

*Denne opgaver er baseret på en tidligere variant af opgaven konstrueret af Torben Mogensen.

```
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:

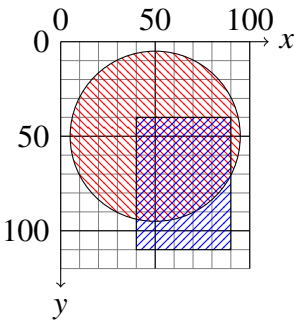
```
type point = int * int // a point (x, y) in the plane
type colour = int * int * int  // (red, green, blue), 0..255

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.2 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 til højre (hvor vi dog har brugt skravering i stedet for udfyldende farver.)



8ø.3 Brug `ImgUtil`-funktionerne og `colourAt` til at lave en funktion

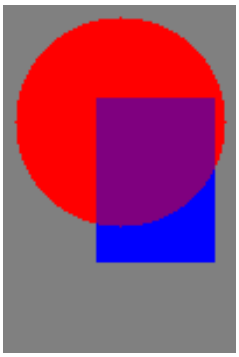
```
makePicture : string -> figure -> int -> int
           -> unit
```

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



Resultatet skulle gerne ligne figuren til højre.

8ø.5 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.6 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 til højre.

8ø.7 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

8g.0 Givet typen for ugedage øverst på denne ugeseddel, lav en funktion `numberToDay : int -> weekday option`, sådan at `numberToDay` n returnerer `None`, hvis n ikke ligger i intervallet $1 \ldots 7$, og returnerer `Some` d, hvor d er den til n hørende ugedag, hvis n ligger i intervallet $1 \ldots 7$.

Det skulle gerne gælde, at `numberToDay (dayToNumber` d`)` $\rightsquigarrow$ `Some` d for alle ugedage d.

Til de følgende opgaver udvider vi typen `figure` med en mulighed for at repræsentere trekanter:

```
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
  | Triangle of point * point * point * colour
     // defined by the three points and colour
```

Konstruktøren `Triangle` tager tre punkter og en farve som argument. Der er intet krav til hvordan de tre punkter er placeret i forhold til hinanden.

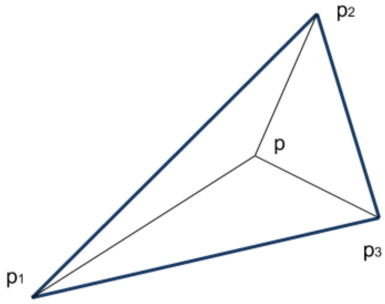
8g.1 Lav en figur `figHouse : figure`, som består af en rød firkant udspændt af punkterne (20,70) og (80,120), en blå trekant udspændt af punkterne (15,80), (45,30) og (85,70), samt en gul cirkel med centrum (70,20) og radius 10.

For at bestemme hvorvidt et punkt er placeret inde i en trekant benytter vi os af et trick der forudsætter at vi kan beregne arealet af en trekant ved at kende dens hjørnepunkter. Det viser sig at hvis en trekant er bestemt af punkterne $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ og $p_3 = (x_3, y_3)$ vil følgende relativt simple formel kunne benyttes til at udregne arealet:

$$area = \left| \frac{x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)}{2} \right|$$

8g.2 Skriv en F# funktion `triarea2` der kan beregne den dobbelte værdi af arealet af en trekant udfra dens tre hjørnepunkter ved at benytte formlen ovenfor. Funktionen skal tage hjørnepunkterne som argumenter og have typen `point -> point -> point -> int`. Test funktionen på et par simple trekanter.[1]

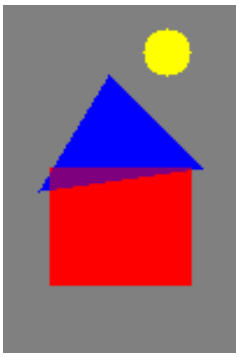
Tricket som vi nu skal benytte til at afgøre om et punkt p ligger inden i en trekant udspændt af hjørnerne p_1, p_2 og p_3 forklares lettest ved at iagttage figuren til højre. Såfremt arealet af de tre trekanter (p_1, p_2, p), (p_2, p_3, p), og (p_1, p_3, p) tilsammen er større end arealet af trekanten (p_1, p_2, p_3), da ligger punktet p udenfor trekanten (p_1, p_2, p_3); ellers ligger punktet indenfor trekanten.



8g.3 Udvid funktionen `colourAt` til at håndtere trekantsudvidelsen ved at implementere tricket nævnt ovenfor samt ved at benytte den implementerede funktion `triarea2`.

8g.4 Lav en fil `figHouse.png`, der viser figuren `figHouse` i et 100×150 bitmap.

Resultatet skulle gerne ligne figuren til højre.



8g.5 Udvid funktionerne `checkFigure` og `boundingBox` fra øvelsesopgaverne til at håndtere udvidelsen.

`boundingBox houseFig` skulle gerne give `((15, 10), (85, 120))`.

Der skal laves black-box testing og in-code dokumentation af funktionerne.

[1] Det viser sig at være hensigtsmæssigt at undgå divisionen med 2, som kan forårsage uheldige afrundingsfejl.

Afleveringsopgaven skal afleveres som både LaTeX, genererede billedfiler, den genererede PDF, samt en fsx fil med løsningen for hver delopgave, navngivet efter opgaven (f.eks. `8g-1.fsx`), som kan oversættes med fsharpc, og hvis resultat kan køres med mono. Det hele samles i en zip-fil med sædvanlig navnekonvention (se tidligere ugesedler).

LaTeX-rapporten skal vise de billedfiler, der bliver lavet i opgaverne, og forklare ikke-oplagte designvalg i løsningerne af opgaverne.

God fornøjelse