

# Programmering og Problemløsning

## Datalogisk Institut, Københavns Universitet

### Arbejdsseddel 8 - individuel opgave

Martin Elsman, Ken Friis Larsen og Jon Sparring

16. november - 21. november.  
Afleveringsfrist: lørdag d. 21. november kl. 22:00.

Some introductory text ...

Emnerne for denne arbejdsseddel er:

- højereordens funktioner.

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

## Øveopgaver (in English)

I det følgende skal I arbejde med polynomier. Et polynomium af grad  $n$  skrives som

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n = \sum_{i=0}^n a_ix^i.$$

- 8ø0 Skriv en funktion `poly: float list -> float -> float`, der tager som argumenter (1) en liste `a` af coefficienter med `a.[i] = ai` og (2) en  $x$ -værdi for derefter at returnere polynomiets værdi. Afprøv funktionen ved at lave tabeller for et lille antal polynomier af forskellig grad med forskellige koefficienter og forskellige værdier for  $x$ , og validér den beregnede værdi.
- 8ø1 Afled en ny funktion `line : float -> float -> float -> float` fra `poly` således at `line a0 a1 x` beregner værdien for et 1. grads polynomium hvor  $a_0 = a_0$ ,  $a_1 = a_1$  og  $x = x$ . Afprøv funktionen ved at tabellere værdier for `line` med det samme sæt af coefficienter  $a_0 \neq 0$  og  $a_1 \neq 0$  og et passende antal værdier for  $x$ .
- 8ø2 Benyt Currying af `line` til at lave en funktion `theLine : float -> float`, hvor parametrene  $a_0$  og  $a_1$  er sat til det samme som brugt i Opgave 8ø1. Afprøv `theLine` tilsvarende som `line` afprøves i Opgave 8ø1.

8ø3 Lav en funktion `lineA0 : float -> float` ved brug af `line`, men hvor `a1` og `x` holdes fast (funktionen `lineA0` tager således kun en  $a_0$  værdi som argument). Diskutér om funktionen kan implementeres ved Currying uden brug af hjælpefunktioner? Hvis ikke, foreslå en hjælpefunktion, som vil gøre en definition vha. Currying mulig.

---

De følgende opgaver omhandler integration. Integralet af næsten alle integrable funktioner kan approximeres som

$$\int_a^b f(x) dx \simeq \sum_{i=0}^{n-1} f(x_i) \Delta x,$$

hvor  $x_i = a + i\Delta x$  og  $\Delta x = \frac{b-a}{n}$ .

8ø4 Skriv en funktion `integrate : n:int -> a:float -> b:float -> (f : float -> float) -> float`, hvis argumenter `n`, `a`, `b`, er som i ligningerne, og `f` er en integrabel 1 dimensionel funktion. Afprøv `integrate` på `theLine` fra Opgave 8ø2 og på `cos` med  $a = 0$  og  $b = \pi$ . Udregn integralerne analytisk og sammenlign med resultatet af `integrate`.

8ø5 Funktionen `integrate` er en approximation, og præcisionen afhænger af  $n$ . Undersøg afhængigheden ved at udregne fejlen, dvs. forskellen mellem det analytiske resultat og approximationen for værdier af  $n$ . Dertil skal du lave to funktioner `IntegrateLine : n:int -> float` og `integrateCos : n:int -> float` vha. `integrate`, `theLine` og `cos`, hvor værdierne for  $a$  og  $b$  og  $f$  er fastlåste. Afprøv disse funktioner for  $n = 1, 10, 100, 1000$ . Overvej om der er en tendens i fejlen, og hvad den kan skyldes.

## Afleveringsopgaver (in English)

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 canvas-objekter til png-filer. Biblioteket er beskrevet i forelæsningerne (i uge 7) og koden for biblioteket ligger sammen med forelæsningsplancherne for uge 6 og er også tilgængeligt via github på <https://github.com/diku-dk/img-util-fs>.

Her bruger vi funktionerne til at tegne på et canvas samt til at gemme canvas-objektet som en png-fil:<sup>1</sup>

```
// colors
type color
val fromRgb : int * int * int -> color

// canvas
type canvas
val mk      : int -> int -> canvas
val setPixel : color -> int * int -> canvas -> unit
```

---

<sup>1</sup>Bemærk at interfacet ikke definerer de konkrete repræsentationstyper for typerne `color` og `canvas`. Disse typer er holdt *abstrakte*, hvilket vil sige at deres repræsentationer ikke kan ses af brugeren af modulet.

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

Funktionen `toPngFile` tager som det første argument navnet på den ønskede png-fil (husk extension). Det andet argument er canvas-objektet som ønskes konverteret og gemt. Et canvas-objekt 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 canvas-objektet 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 canvas-objektet (dvs. i billedet). Det tredje argument repræsenterer det canvas-objekt, 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.

Koordinatsystemet har nulpunkt  $(0,0)$  i øverste venstre hjørne og, såfremt vidden og højden af koordinatsystemet er henholdsvis  $w$  og  $h$ , optræder punktet  $(w-1, h-1)$  i nederste højre hjørne. Antag for eksempel at programfilen `testPNG.fsx` indeholder følgende F# kode:

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

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

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

Den genererede billedfil `test.png` vil indeholde et hvidt billede med et pixel af rød 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 L<sup>A</sup>T<sub>E</sub>X 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 color = ImgUtil.color

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

Man kan, for eksempel, 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 color of figure at point
let rec colorAt (x,y) figure =
  match figure with
```

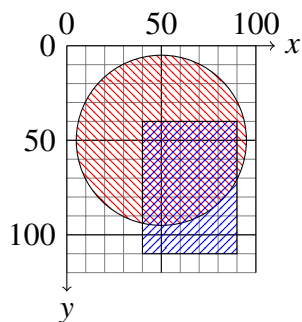
```

| Circle ((cx,cy), r, col) ->
  if (x-cx)*(x-cx)+(y-cy)*(y-cy) <= r*r
    // uses Pythagoras' equation 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 (colorAt (x,y) f1, colorAt (x,y) f2) with
  | (None, c) -> c // no overlap
  | (c, None) -> c // no overlap
  | (Some c1, Some c2) ->
    let (a1,r1,g1,b1) = ImgUtil.fromColor c1
    let (a2,r2,g2,b2) = ImgUtil.fromColor c2
    in Some(ImgUtil.fromArgb((a1+a2)/2, (r1+r2)/2, // calculate
      (g1+g2)/2, (b1+b2)/2)) // average
color

```

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

8i0 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.)



8i1 Brug `ImgUtil`-funktionerne og `colorAt` til at lave en funktion

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

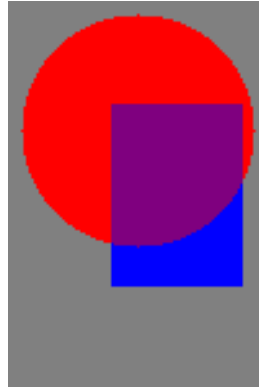
sådan at kaldet `makePicture filnavn figur b h` laver en billedfil ved navn `filnavn.png` med et billede af `figur` med bredde `b` og højde `h`.

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

Du kan bruge denne funktion til at afprøve dine opgaver.

8i2 Brug funktionen `makePicture` til at konstruere en billedfil med navnet `figTest.png` og størrelse  $100 \times 150$  (bredde 100, højde 150), der viser figuren `figTest` fra Opgave 8i0.

Resultatet skulle gerne ligne figuren nedenfor.



8i3 Lav en funktion `checkFigure : figure -> bool`, der undersøger, om en figur er korrekt: At radiusen i cirkler er ikke-negativ og 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).

8i4 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
```

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



8i5 Lav en funktion `boundingBox : figure -> point * point`, der givet en figur finder hjørnerne (top-venstre og bund-højre) for den mindste akserrette rektangel, der indeholder hele figuren.

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

## Krav til afleveringen

Afleveringen skal bestå af

- en zip-fil

Zip-filen skal indeholde en `src` mappe og filen `README.txt`. Mappen skal indeholde fsharp koden, der skal være en fsharp tekstfil per fsharp-opgave, og de skal navngives `8i0.fsx` osv. De skal kunne oversættes med fsharpc, og de oversatte filer skal kunne køres med mono. Funktioner skal

dokumenteres ifølge dokumentationsstandarden, og udover selve programteksten skal besvarelsenerne indtastes som kommentarer i de fsx-filer, de hører til. Filen README.txt skal ganske kort beskrive, hvordan koden køres.

God fornøjelse.