

Programmering og Problemløsning

Datalogisk Institut, Københavns Universitet

Arbejdsseddel 8 - individuel opgave

Jon Sparring

18. november - 24. november.
Afleveringsfrist: søndag d. 24. november kl. 22:00.

Emnerne for denne arbejdsseddel er:

- højereordens funktioner,
- fejl og undtagelser.

Opgaverne er delt 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

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

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

8ø.0 Skriv en funktion `poly`: `a: float list -> x: float -> float`, som tager en liste af koefficienter med `a.[i] = ai` og en x -værdi og returnerer 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.

8ø.1 Afled en ny funktion `line` fra `poly` således at `line : a0: float -> a1: float -> x: float -> float` beregner værdien for et 1. grads polynomium hvor $a_0 = a_0$ og $a_1 = a_1$. Afprøv funktionen ved at tabellere værdier for `line` med det samme sæt af koefficienter $a_0 \neq 0$ og $a_1 \neq 0$ og et passende antal værdier for x .

8ø.2 Benyt Currying af `line` til at lave en funktion `theLine` : `x: float -> float`, hvor parametrene a_0 og a_1 er sat til det samme som brugt i Opgave 8ø.1. Afprøv `theLine` som Opgave 8ø.1.

- 8ø.3 Lav en funktion `lineA0 : a0:float -> float` ved brug af `line`, men hvor `a1` og `x` holdes fast. Diskutér om dette kan laves ved Currying uden brug af hjælpefunktioner? Hvis ikke, foreslå en hjælpefunktion, som vil gøre en definition vha. Currying muligt.
- 8ø.4 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 8ø.2 og på `cos` med $a = 0$ og $b = \pi$. Udregn integralerne analytisk og sammenlign med resultatet af `integrate`.
- 8ø.5 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.
- 8ø.6 Implementer fakultetsfunktionen $n! = \prod_{i=1}^n i$, $n > 0$ som `fac : n:int -> int` og kast en `System.ArgumentException` undtagelse, hvis funktionen bliver kaldt med $n < 1$. Kald `fac` med værdierne $n = -4, 0, 1, 4$, og fang evt. undtagelser.
- 8ø.7 Tilføj en ny og selvdefineret undtagelse `ArgumentTooBig` af string til `fac`, og kast den med argumentet `"calculation would result in an overflow"`, når n er for stor til `int` typen. Fang undtagelsen og udskriv beskeden sendt med undtagelsen på skærmen.
- 8ø.8 Lav en ny fakultetsfunktion `facFailwith : n:int -> int`, som `fac`, men hvor de 2 undtagelser bliver erstattet med `failwith` med hhv. argument `"argument must be greater than 0"` og `"calculation would result in an overflow"`. Kald `facFailWith` med $n = -4, 0, 1, 4$, fang evt. undtagelser vha. `Failure` mønsteret, og udskriv beskeden sendt med `failwith` undtagelsen.
- 8ø.9 Omskriv fakultetsfunktionen i Opgave 8ø.7, som `facOption : n:int -> int option`, således at den returnerer `Some m`, hvis resultatet kan beregnes og `None` ellers. Kald `fac` med værdierne $n = -4, 0, 1, 4$, og skriv resultatet ud vha. en af `printf` funktionerne.
- 8ø.10 Skriv en funktion `logIntOption : n:int -> float option`, som udregner logaritmen af n , hvis $n > 0$ og `None` ellers. Afprøv `logIntOption` for værdierne $-10, 0, 1, 10$.
- 8ø.11 Skriv en ny funktion `logFac : int -> float option` vha. `Option.bind` 1 eller flere gange til at sammensætte `logIntOption` og `facOption`, og sammenlign `logFac` med Stirlings approximation $n * (\log n) - n$ for værdierne $n = 1, 2, 4, 8$.
- 8ø.12 Funktionen `logFac : int -> float option` kan defineres som en enkelt sammensætning af funktionerne `Some` og `Option.bind` en eller flere gange og med `logIntOption` og `facOption` som argument til `Option.bind`. Opskriv 3 udtryk, der bruger hhv. `|>` eller `>>` operatorene eller ingen af dem.

Afleveringsopgaver

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 7) 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 tredje 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.

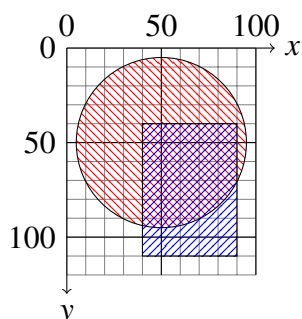
```

// 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' 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 (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 cirkelns omkreds og rektanglens kanter er med i figuren. Farver blandes ved at lægge dem sammen og dele med to, altså finde gennemsnitsfarven.

8i.0 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.)



8i.1 Brug ImgUtil-funktionerne og colourAt til at lave en funktion

```

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

```

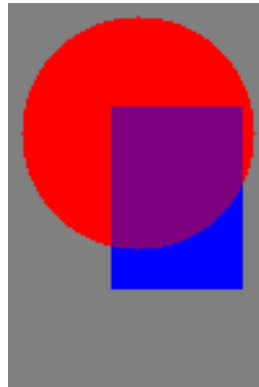
sådan at kaldet `makePicture` filnavn figur b h laver en billedfil ved navn `filnavn.png` med et billede af figur med bredde b og højde h.

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.

- 8i.2 Lav med `makePicture` en billedfil med navnet `figTest.png` og størrelse 100×150 (bredde 100, højde 150), der viser figuren `figTest` fra Opgave 8i.0.

Resultatet skulle gerne ligne figuren nedenfor.



- 8i.3 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 farvekomponenterne ligger mellem 0 og 255.

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

- 8i.4 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.



- 8i.5 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))`.

Afleveringen skal bestå af

- en zip-fil og
- en rapport i pdf-format.

Zip-filen skal indeholde mapperne `tex` og `src` og filen `README.txt`. Mapperne skal indeholde henholdsvis LaTeX koden m.m. og fsharp koden. Der skal være en fsharp tekstfil per fsharp-opgave, og de skal navngives `8i0.fsx` osv. Rapporten skal kunne oversættes vha. `pdflatex` kommandoen i `tex`-mappen, og fsharp koden skal kunne oversættes med `fsharpc` og køres med `mono` i `src` mappen. Filen `README.txt` skal ganske kort beskrive, hvordan rapporten oversættes til pdf og koden køres.

God fornøjelse.