PoP aflevering 3

Bjørn Møller

27. september 2017

i3.0

a
I denne opgave skulle nedenstående kode laves:

```
let sum n = (
 let mutable i = 1
 let mutable total = 0
 while i <= n do
    total <- total + i
    i <- i + 1
 total)
printfn "sum med n = %i  giver  %A" 3   (sum 3)
```

Resultatet af koden er: "sum med n = 3 giver 6
Her kan desduden ses, at der er brugt den eftersøgte "while- løkke, som blev efterspurgt i opgaven. Løkken vil derfor køre indtil, at den Den har kørt antal omgange svarende til indputtet n.
b

I denne opgave skulle nedenstående kode laves:

```
let smartsum n = (n*(n+1)/2 )
printfn "smartsum med n = %i giver  %A" 3   (smartsum 3)
```

Resultatet af koden er: "smartsum med n = 3 giver 6

c

I denne opgave skulle nedenstående kode laves:

```
let sum n = (
 let mutable i = 1
 let mutable total = 0
 while i <= n do
    total <- total + i
    i <- i + 1
 total)

let smartsum n = (n*(n+1)/2 )
printfn "smartsum med n = %i giver  %A" 3  (smartsum 3)

for n = 0 to 10 do
    printfn "%5i %5i %5i" n (sum n) (smartsum n)
```

Denne koder giver en tabel i tone med nedenstående:

0	0	0
1	1	1
2	3	3
3	6	6
4	10	10
5	15	15
6	21	21
7	28	28
8	36	36
9	45	45
10	55	55

Vi kan altså konkludere, at de to koder fungerer ens.

d

For at løse denne opgave har jeg kørt nedenstående kode, og derefter undersøgt, ved hvilket n, at deres output bliver negativt, hvilket er en konsekvens af "overflow":

```
let sum n = (
 let mutable i = 1
 let mutable total = 0
 while i <= n do
    total <- total + i
    i <- i + 1
 total)

let smartsum n = (n*(n+1)/2 )
printfn "smartsum med n = %i giver  %A" 3  (smartsum 3)

for n = 0 to 66500 do
    printfn "%5i %5i %5i" n (sum n) (smartsum n)
```

Ved at køre denne kode kan jeg konkluderere, at det maksimale n = 46340 for smartsum før den laver overflow, hvor overflow først sker ved n = 65536 for funktionen sum. Man kan ligeledes bemærke, at ved n = 65536 går smartsum igen fra negativ til positiv. Disse fakta giver god mening eftersom smartsum udregner $n * n$. Variablen bliver altså dobbelt så stor inden den divideres med 2. Dette gør reelt, at variablen holder en dobbelt så stor værdi, som den normale sum.

Man kan delvist løse ovenstående overflow problem, ved at omdanne disse integers til float variable som kan indeholde større variable.

Et eksempel på dette, er vist i nedenstående kode:

```
let sum (n: float) = (
 let mutable i = 1.
 let mutable total = 0.
 while i <= n do
    total <- total + i
    i <- i + 1.
 total)

let smartsum (n: float) = (n*(n+1.)/2. )
printfn "smartsum med n = %f giver  %A" 3.  (smartsum 3.)

for n = 0 to 10   do
    let y = float(n)
    printfn "%5f %5f %5f" y (sum y) (smartsum y)
```

i3.1

a
Denne Opgave kan løses med nedenstående kode:

```
let multable j  = (
 let mutable str = sprintf "\n%4s " (" ")
 for n = 1 to 10 do
  str <- str + sprintf "%4i" n
 str <- sprintf  "%4s" str

 for n = 1 to 10 do
  str <- sprintf "%s\n %4i" str (n)
 for y = 1 to 10 do
   let k = sprintf "%4i" (n * y)
   str <- str  + k
 let b = j*46+44
 sprintf "%s" str.[0..b]
)
printf "%s"  (multable 1)
```

Man i koden se, at jeg har sat $b = j * 46 + 44$. Dette tal virker en smule tilfældigt, men er reelt et tal, som medfører at for hvert n bliver den tilhørende række printet ud. Den første række bliver altid printet ud derfor lægges 44 til. For hvert n bliver der tilført 44 +2 , da der er 11 pladser med 4 længder mellem hver ergo 44, og de to ekstra sørger for linje skiftet.

b

Denne Opgave kan løses med nedenstående kode:

```
let Loopmultable a = (
 let mutable str = sprintf "\n%4s " (" ")
 for n = 1 to 10 do
  str <- str + sprintf "%4i" n
 str <- sprintf  "%4s" str

 for n = 1 to a do
  str <- sprintf "%s\n %4i" str (n)
  for y = 1 to 10 do
   let k = sprintf "%4i" (n * y)
   str <- str  + k
 str
)
```

I denne kode, afhænger antallet af printede rækker dynamisk af indputtet, da indputtet n bruges til at bestemme hvor mange gange Den yderste løkke kører, hvilket opgbygger den færdige .

c

Denne Opgave kan løses med nedenstående kode:

```
let test n = (
    (Loopmultable n) = (multable n)
)
printfn "n test(n) "
for n = 1 to 10 do
 printfn "%A %A"n (test n)
```

En gennemkørsel fra n = 1 til 10, viser, returnere TRUE for alle n, at de er identiske ved alle n. Dette giver reelt mening, da begge koder i mine forskellige test af n har returneret identiske strenge.

d

Ved en simpel test, kan man se, at den kosmetiske forskel er, at hvis du forsøger at printe en string med $\%A$ kommer der citations tegn rundt om værdierne printet ud, hvilket der ikke gør hvis man bruger $\%s$. Dette er fordi, at $\%A$ bruger disse tegn til at fortælle, at værdien den viser er en string, hvilket $\%s$ ikke behæver, da den kun modtager string elementer.