Programmering og Problemløsning

4.3: Håndkøring

Repetition af Nøglekoncepter

- Stakken og bunken
- Referenceceller
- Højere-ordens funktioner
- Anonyme funktioner

- Moduler og biblioteker
- Black- og white-box testing
- Højere-ordens funktioner
- Anonyme funktioner

Unit	Case	Expected output	Comment
dec2bin n	n = -1	"Illegal value"	negative tal
	n = 0	"0b0"	grænsetilfælde
	n = 1	"0b1"	1 bit
	n = 2	"0b10"	2 bit
	n = 10	"0b1010"	stort lige tal (venstre bit sat min ikke højre)
	n = 11	"0b1011"	stort ulige tal (venstre og højre bit sat)

Unit	Branch	Condition	Input	Expected output	Comment
dec2bin	1	n < 0			
	1a	true	-1	"Illegal value"	
	1b	branch 2			Fall through
	2 (n>=0)	n = 0			
	2a	true	0	"0b0"	
	2b	branch 3			Fall through
	3 (n>0)				
	3a	true	1	"0b1"	
	3b	false	1	"0b1"	

Closures = funktioner som værdier

En højere-ordens function:

```
let mul x y = x * y
let factor = 2.0
let applyFactor fct x =
  let a = fct factor x
  string a

printfn "%g" (mul 5.0 3.0)
printfn "%s" (applyFactor mul 3.0)
```

Closure notation:

Navn -> (input, krop, virkefeltets værdier)

Værdier:

```
mul -> ((x, y), (x * y), ())
factor -> 2.0
applyFactor -> ((fct, x), let a = fct factor x; string a, (mul -> ((x, y), (x * y), ()), factor -> 2.0))
```

Håndkøring: simulér computeren

```
1   let doit n =
2     for i = 1 to n do
3       let p = i * i
4       printfn "%d: %d" i p
5
6   let N = 3
7   doit N
```

```
$ fsharpi simpleForLoop.fsx
1: 1
2: 4
3: 9
```

E0:
- doit -> ((n), doit-body, ())
- N -> 3
- linje 6: doit N -> ? ()

E1: ((n -> 3), doit-body, ())
- i -> 1
- p -> 1
- stdout -> “1: 1”
- i -> 2
- p -> 4
- stdout -> “2: 4”
- i -> 3
- p -> 9
- stdout -> “3: 9”
- return -> ()

Leksikografisk versus Dynamisk Virkefelt

Leksikografisk

```
let testScope x =
  let a = 3.0
  let f z = a * z
  let a = 4.0
  f x
printfn "%A" (testScope 2.0)
```

Dynamisk

```
let testScope x =
  let mutable a = 3.0
  let f z = a * z
  a <- 4.0
  f x
printfn "%A" (testScope 2.0)
```

Håndkøring: Leksikografisk virkefelt

```
1   let testScope x =
2     let a = 3.0
3     let f z = a * z
4     let a = 4.0
5     f x
6   printfn "%A" (testScope 2.0)
```

```
$ fsharpi lexicalScopeTracing.fsx
6.0
```

E0:
  testScope -> ((x), testScope-body, ())
  linje 6: testScope 2.0 -> ? 6.0
  stdout -> "6.0"
  return -> ()

E1: ((x -> 2.0), testScope-body, ())
  a -> 3.0
  f -> ((z), a * z, (a -> 3.0))
  a -> 4.0
  f 2.0 -> ? 6.0
  return -> 6.0

E2: ((z -> 2.0), a * z, (a -> 3.0))
  return -> 6.0

Håndkøring: Dynamisk virkefelt

```
1   let testScope x =
2     let mutable a = 3.0
3     let f z = a * z
4     a <- 4.0
5     f x
6   printfn "%A" (testScope 2.0)
```

```
$ fsharpi
dynamicScopeTracing.fsx
8.0
```

E0:
  testScope -> ((x), testScope-body, ())
  linje 6: testScope 2.0 -> ? 8.0
  stdout -> "8.0"
  return -> ()

E1: ((x -> 2.0), testScope-body, ())
  a -> alpha
  f -> ((z), a * z, (a –> alpha))
  f 2.0 -> ? 8.0
  return -> 8.0

E2: ((z -> 2.0), a * z, (a -> alpha))
  return -> 8.0

Linje	Navn	Værdi
2	alpha	3.0
4	alpha	4.0

Til tavlen

Closure i closure

```
let mul x y = x * y
let factor = 2.0
let applyFactor fct x =
  let a = fct factor x
  string a

printfn "%g" (mul 5.0 3.0)
printfn "%s" (applyFactor mul 3.0)
```

```
$ fsharpi functionFirstClass.fsx
15
6
```

Den virkeligt sværre

```
let incr =
  let mutable counter = 0
  fun () ->
    counter <- counter + 1
    counter

printfn "%d" (incr ())
printfn "%d" (incr ())
printfn "%d" (incr ())
```

```
$ fsharpi inc.fsx
1
2
3
```