Programkonstruktion och Datastrukturer

Repetition, vårterminen 2012

Att läsa och förstå kod

Elias Castegren
elias.castegren.7381@student.uu.se

”**Programs should be written for people to read, and only incidentally for machines to execute**”

- Harold Abelson

Att översätta kod

- `String.sub(s, 3)` =>

  ”Det tredje tecknet i strängen `s`”

- `if String.sub(s, 3) = #”a” then` =>

  ”Om det tredje tecknet i `s` är ett 'a' så...”

- `if x > y then ... else ...`

  ”Om x är större än y så …

  Annars (om x inte är större än y) så …”

Att översätta kod

- `fun dist(x,y) = abs(x-y);`

  `dist(a,b)` => ”Avståndet mellan a och b”
- Funktioner med returtypen bool blir påståenden:
  - `fun isClose(x,y) = dist(x,y) < 3;`

    `isClose(a,b)` =>

    ”Avståndet mellan a och b är mindre än 3”
  - `if isClose(a,b) then ...` =>

    ”Om avståndet mellan a och b är mindre än 3 så ...”

Rekursion

- Hitta en översättning av vad funktionen gör redan innan funktionen är skriven:

```
fun findMax ... = ...
```

`findMax(l)` => ”det största talet i listan `l`”

- ”Det största talet i en lista är det största av det första talet och det största i resten av listan”

```
fun findMax [x] = x
  | findMax (f::r) =
        Int.max(f, findMax(r));
```

Rekursion

- Hitta en översättning av vad funktionen gör redan innan funktionen är skriven:

```
fun findMax ... = ...
```

`findMax(l)` => ”det största talet i listan `l`”

- ”Det största talet i en lista är det största av det första talet och det största i resten av listan”

```
fun findMax [x] = x
  | findMax (f::r) =
        Int.max(f, findMax(r));
```

Rekursion

- Man kan testa en rekursiv funktion manuellt genom att mekaniskt följa instruktionerna på ett litet exempel:

```
fun findMax [x] = x
  | findMax (f::r) = Int.max(f, findMax(r));
```

```
findMax([3,9,6,8])
Int.max(3, findMax([9,6,8]))
Int.max(3, Int.max(9, findMax([6,8])))
Int.max(3, Int.max(9, Int.max(6, findMax([8]))))
Int.max(3, Int.max(9, Int.max(6, 8)))
```

Rekursion

- Man kan testa en rekursiv funktion manuellt genom att mekaniskt följa instruktionerna på ett litet exempel:

```
fun findMax [x] = x
  | findMax (f::r) = Int.max(f, findMax(r));
```

```
Int.max(3, Int.max(9, Int.max(6, 8)))
Int.max(3, Int.max(9, 8))
Int.max(3, 9)
9
```

Ett exempel

- Ett antal studenter svarar med ”Ja”, ”Nej” eller ”Kanske” på en enkät om de tror de kan lösa ett antal uppgifter.
- Om man ger ”Ja”, ”Nej” och ”Kanske” värdena 0, 1 och 0,5 kan man mäta hur relevant det är att repetera en viss frågas ämne genom att räkna ut svarets medelvärde bland alla svarande.

```
datatype answer = Ja | Nej | Kanske;

(* ansToReal a
   TYPE: answer -> real
   POST: 1.0, 0.0 eller 0.5 beroende på om a
är Nej, Ja respektive Kanske
   EXAMPLES: ansToReal Kanske = 0.5
*)
fun ansToReal a =
    if a = Ja then
        0.0
    else if a = Nej then
        1.0
    else
        0.5;
```

```
datatype answer = Ja | Nej | Kanske;

(* ansToReal a
   TYPE: answer -> real
   POST: 1.0, 0.0 eller 0.5 beroende på om a
är Nej, Ja respektive Kanske
   EXAMPLES: ansToReal Kanske = 0.5
*)
fun ansToReal a =
    case a of
        Ja => 0.0
      | Nej => 1.0
      | _ => 0.5;
```

```
datatype answer = Ja | Nej | Kanske;

(* ansToReal a
   TYPE: answer -> real
   POST: 1.0, 0.0 eller 0.5 beroende på om a
är Nej, Ja respektive Kanske
   EXAMPLES: ansToReal Kanske = 0.5
*)
fun ansToReal Ja  = 0.0
  | ansToReal Nej = 1.0
  | ansToReal _   = 0.5;
```

```
(* getStatistics ansList
   TYPE: answer list list -> real list
   POST: En lista där det n:te elementet är medelvärdet
av det n:te svaret i varje lista i ansList
   EXAMPLES: getStatistics [[Nej,Ja,Ja], [Nej,Nej,Ja]] =
[1.0, 0.5, 0.0]
*)

fun getStatistics l =
```

Basfall

else

Allmänt fall

```
(* getStatistics ansList
   TYPE: answer list list -> real list
   POST: En lista där det n:te elementet är medelvärdet
av det n:te svaret i varje lista i ansList
   EXAMPLES: getStatistics [[Nej,Ja,Ja], [Nej,Nej,Ja]] =
[1.0, 0.5, 0.0]
*)

fun getStatistics l =
```

Basfall

```
    else
```

Medelvärdet av första frågan följt av

Medelvärdena av de resterande frågorna

```
(* getStatistics ansList
   TYPE: answer list list -> real list
   POST: En lista där det n:te elementet är medelvärdet
av det n:te svaret i varje lista i ansList
   EXAMPLES: getStatistics [[Nej,Ja,Ja], [Nej,Nej,Ja]] =
[1.0, 0.5, 0.0]
*)

fun getStatistics l =
```

Basfall

```
    else
        sumHeads(l) / real(length(l))
```

följt av

Medelvärdena av de resterande frågorna

```
(* getStatistics ansList
   TYPE: answer list list -> real list
   POST: En lista där det n:te elementet är medelvärdet
av det n:te svaret i varje lista i ansList
   EXAMPLES: getStatistics [[Nej,Ja,Ja], [Nej,Nej,Ja]] =
[1.0, 0.5, 0.0]
*)

fun getStatistics l =
```

Basfall

```
    else
        sumHeads(l) / real(length(l)) ::
```

Medelvärdena av de resterande frågorna

```
(* getStatistics ansList
   TYPE: answer list list -> real list
   POST: En lista där det n:te elementet är medelvärdet
av det n:te svaret i varje lista i ansList
   EXAMPLES: getStatistics [[Nej,Ja,Ja], [Nej,Nej,Ja]] =
[1.0, 0.5, 0.0]
*)

fun getStatistics l =
```

Basfall

```
    else
        sumHeads(l) / real(length(l)) ::
            getStatistics
```

de resterande frågorna

```
(* getStatistics ansList
   TYPE: answer list list -> real list
   POST: En lista där det n:te elementet är medelvärdet
av det n:te svaret i varje lista i ansList
   EXAMPLES: getStatistics [[Nej,Ja,Ja], [Nej,Nej,Ja]] =
[1.0, 0.5, 0.0]
*)

fun getStatistics l =
```

Basfall

```
    else
        sumHeads(l) / real(length(l)) ::
            getStatistics(getTails l);
```

```
(* getStatistics ansList
   TYPE: answer list list -> real list
   PRE: Ingen lista i ansList är kortare än den första
   POST: En lista där det n:te elementet är medelvärdet
av det n:te svaret i varje lista i ansList
   EXAMPLES: getStatistics [[Nej,Ja,Ja], [Nej,Nej,Ja]] =
[1.0, 0.5, 0.0]
*)
(* VARIANT:                *)
fun getStatistics l =
    if null(hd(l)) then
        []
    else
        sumHeads(l) / real(length(l)) ::
            getStatistics(getTails l);
```

```
(* getStatistics ansList
   TYPE: answer list list -> real list
   PRE: Ingen lista i ansList är kortare än den första
   POST: En lista där det n:te elementet är medelvärdet
av det n:te svaret i varje lista i ansList
   EXAMPLES: getStatistics [[Nej,Ja,Ja], [Nej,Nej,Ja]] =
[1.0, 0.5, 0.0]
*)
(* VARIANT: length(hd l) *)
fun getStatistics l =
    if null(hd(l)) then
        []
    else
        sumHeads(l) / real(length(l)) ::
            getStatistics(getTails l);
```

```
(* getStatistics ansList
   TYPE: answer list list -> real list
   PRE: Ingen lista i ansList är kortare än den första
   POST: En lista där det n:te elementet är medelvärdet
av det n:te svaret i varje lista i ansList
   EXAMPLES: getStatistics [[Nej,Ja,Ja], [Nej,Nej,Ja]] =
[1.0, 0.5, 0.0]
*)
(* VARIANT: length(hd l) *)
fun getStatistics ([]::_) = []
  | getStatistics l =
        sumHeads(l) / real(length(l)) ::
            getStatistics(getTails l);
```

Vi implementerar `sumHeads` och `getTails`!

```
(* sumHeads l
   TYPE: answer list list -> real
   PRE: Alla listor i l har minst ett element
   POST: summan av det första elementet i varje lista i
l, där Ja = 0.0, Kanske = 0.5 och Nej = 1.0
   EXAMPLES: sumHeads [[Ja, Kanske], [Kanske, Nej]] =
                  0.5
*)
(* VARIANT: length l *)
fun sumHeads [] = 0.0
  | sumHeads l = ansToReal(hd(hd(l)))
                    + sumHeads(tl(l));
```

”Summan av det första elementet i varje lista i l är det första element i den första listan av l adderat med summan av det första elementet i varje lista i resten av l”

```
(* sumHeads l
   TYPE: answer list list -> real
   PRE: Alla listor i l har minst ett element
   POST: summan av det första elementet i varje lista i
l, där Ja = 0.0, Kanske = 0.5 och Nej = 1.0
   EXAMPLES: sumHeads [[Ja, Kanske], [Kanske, Nej]] =
                  0.5
*)
(* VARIANT: length l *)
fun sumHeads [] = 0.0
  | sumHeads l = ansToReal(hd(hd(l)))
                    + sumHeads(tl(l));
```

”**Summan av det första elementet i varje lista i l** är det första element i den första listan av l adderat med summan av det första elementet i varje lista i resten av l”

```
(* sumHeads l
   TYPE: answer list list -> real
   PRE: Alla listor i l har minst ett element
   POST: summan av det första elementet i varje lista i
l, där Ja = 0.0, Kanske = 0.5 och Nej = 1.0
   EXAMPLES: sumHeads [[Ja, Kanske], [Kanske, Nej]] =
                  0.5
*)
(* VARIANT: length l *)
fun sumHeads [] = 0.0
  | sumHeads l = ansToReal(hd(hd(l)))
                    + sumHeads(tl(l));
```

”Summan av det första elementet i varje lista i l är det **första element** i den **första listan av l** adderat med summa av det första elementet i varje lista i resten av l”

```
(* sumHeads l
   TYPE: answer list list -> real
   PRE: Alla listor i l har minst ett element
   POST: summan av det första elementet i varje lista i
l, där Ja = 0.0, Kanske = 0.5 och Nej = 1.0
   EXAMPLES: sumHeads [[Ja, Kanske], [Kanske, Nej]] =
                  0.5
*)
(* VARIANT: length l *)
fun sumHeads [] = 0.0
  | sumHeads l = ansToReal(hd(hd(l)))
                    + sumHeads(tl(l));
```

”Summan av det första elementet i varje lista i l är det första element i den första listan av l **adderat med** summan av det första elementet i varje lista i resten av l”

```
(* sumHeads l
   TYPE: answer list list -> real
   PRE: Alla listor i l har minst ett element
   POST: summan av det första elementet i varje lista i
l, där Ja = 0.0, Kanske = 0.5 och Nej = 1.0
   EXAMPLES: sumHeads [[Ja, Kanske], [Kanske, Nej]] =
                  0.5
*)
(* VARIANT: length l *)
fun sumHeads [] = 0.0
  | sumHeads l = ansToReal(hd(hd(l)))
                    + sumHeads(tl(l));
```

”Summan av det första elementet i varje lista i l är det första element i den första listan av l adderat med **summan av det första elementet i varje lista** i **resten av l**”

```
(* sumHeads l
   TYPE: answer list list -> real
   PRE: Alla listor i l har minst ett element
   POST: summan av det första elementet i varje lista i
l, där Ja = 0.0, Kanske = 0.5 och Nej = 1.0
   EXAMPLES: sumHeads [[Ja, Kanske], [Kanske, Nej]] =
                  0.5
*)
(* VARIANT: length l *)
fun sumHeads [] = 0.0
  | sumHeads (l::ls) = ansToReal(hd(l))
                         + sumHeads(ls);
```

```
(* sumHeads l
   TYPE: answer list list -> real
   PRE: Alla listor i l har minst ett element
   POST: summan av det första elementet i varje lista i
l, där Ja = 0.0, Kanske = 0.5 och Nej = 1.0
   EXAMPLES: sumHeads [[Ja, Kanske], [Kanske, Nej]] =
                  0.5
*)
(* VARIANT: length l *)
fun sumHeads [] = 0.0
  | sumHeads ((f::_)::ls) = ansToReal(f)
                            + sumHeads(ls);
```

```
(* sumHeads l
   TYPE: answer list list -> real
   PRE: Alla listor i l har minst ett element
   POST: summan av det första elementet i varje lista i
l, där Ja = 0.0, Kanske = 0.5 och Nej = 1.0
   EXAMPLES: sumHeads [[Ja, Kanske], [Kanske, Nej]] =
                  0.5
*)
(* VARIANT: length l *)
fun sumHeads [] = 0.0
  | sumHeads l =
    let
        val firstHead = hd(hd(l))
        val rest = tl(l)
    in
        ansToReal(firstHead) + sumHeads rest
    end;
```

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)
(* VARIANT: length l *)
fun getTails [] = []
  | getTails l = tl(hd(l)) :: getTails(tl(l));
```

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)
(* VARIANT: length l *)
fun getTails [] = []
  | getTails l = tl(hd(l)) :: getTails(tl(l));
```

”**Svansarna av alla listor i l**...”

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)
(* VARIANT: length l *)
fun getTails [] = []
  | getTails l = tl(hd(l)) :: getTails(tl(l));
```

”Svansarna av alla listor i l är **svansen av**...”

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)
(* VARIANT: length l *)
fun getTails [] = []
  | getTails l = tl(hd(l)) :: getTails(tl(l));
```

”Svansarna av alla listor i l är svansen av **den första listan i l**...”

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)
(* VARIANT: length l *)
fun getTails [] = []
  | getTails l = tl(hd(l)) :: getTails(tl(l));
```

”Svansarna av alla listor i l är svansen av den första listan i l **följt av**...”

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)
(* VARIANT: length l *)
fun getTails [] = []
  | getTails l = tl(hd(l)) :: getTails(tl(l));
```

”Svansarna av alla listor i l är svansen av den första listan i l följt av **svansarna av**...”

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)
(* VARIANT: length l *)
fun getTails [] = []
  | getTails l = tl(hd(l)) :: getTails(tl(l));
```

”Svansarna av alla listor i l är svansen av den första listan i l följt av svansarna av **resten av listorna i l**”

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)
(* VARIANT: length l *)
fun getTails [] = []
  | getTails l = tl(hd(l)) :: getTails(tl(l));
```

”**Svansarna av alla listor i l** är **svansen av den första listan i l följt av svansarna av resten av listorna i l**”

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)
(* VARIANT: length l *)
fun getTails [] = []
  | getTails l = tl(hd(l)) :: getTails(tl(l));



getTails [[Ja, Kanske], [Kanske, Nej]]
tl([Ja, Kanske]) :: getTails([[Kanske, Nej]])
tl([Ja, Kanske]) :: tl([Kanske, Nej]) :: getTails([])
[Kanske] :: [Nej] :: []
[[Kanske], [Nej]]
```

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)
(* VARIANT: length l *)
fun getTails [] = []
  | getTails (l::ls) = tl(l) :: getTails ls;
```

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)
(* VARIANT: length l *)
fun getTails [] = []
  | getTails ((_::r)::ls) = r :: getTails ls;
```

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)

fun getTails l = map tl l;
```

”Svansen av alla listor i l får man genom att applicera tail-funktionen på varje element i l”

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)

fun getTails l = map tl l;
```

”**Svansen av alla listor** i **l** får man genom att **applicera tail-funktionen på varje element** i **l**”

```
(* getTails l
   TYPE: 'a list list -> 'a list list
   PRE: Alla listor i l har minst ett element
   POST: l med alla förstaelement borttagna från
listorna
   EXAMPLES: getTails [[Ja, Kanske], [Kanske, Nej]] =
             [[Kanske], [Nej]]
*)

val getTails = map tl;
```

”**Svansen av alla listor** får man genom att **applicera tail-funktionen på varje element**”

```
(* getStatistics ansList
   TYPE: answer list list -> real list
   PRE: Ingen lista i ansList är kortare än den första
   POST: En lista där det n:te elementet är medelvärdet
av det n:te svaret i varje lista i ansList
   EXAMPLES: getStatistics [[Nej,Ja,Ja], [Nej,Nej,Ja]] =
[1.0, 0.5, 0.0]
*)
(* VARIANT: length(hd l) *)
fun getStatistics ([]::_) = []
  | getStatistics l =
        sumHeads(l) / real(length(l)) ::
            getStatistics(getTails l);
```

`getStatistics` har de funktioner den behöver!

```
(* getStatistics ansList
   TYPE: answer list list -> real list
   PRE: Ingen lista i ansList är kortare än den första
   POST: En lista där det n:te elementet är medelvärdet
av det n:te svaret i varje lista i ansList
   EXAMPLES: getStatistics [[Nej,Ja,Ja], [Nej,Nej,Ja]] =
[1.0, 0.5, 0.0]
*)
(* VARIANT: length(hd l) *)
fun getStatistics ([]::_) = []
  | getStatistics l =
    let
        val listCount = real(length l)
    in
        sumHeads(l) / listCount ::
            getStatistics(getTails l)
    end;
```

```
fun getStatistics ([]::_) = []
  | getStatistics l =
    let
       val listCount = real(length l)
       val firstMean = sumHeads(l) / listCount
       val restOfMeans = getStatistics(getTails l)
    in
       firstMean :: restOfMeans
    end;
```

”Medelvärdet av alla frågor är medelvärdet av den första frågan, följt av medelvärdena av resten av frågorna”

```
fun getStatistics ([]::_) = []
  | getStatistics l =
    let
        val listCount = real(length l)
        val firstMean = sumHeads(l) / listCount
        val restOfMeans = getStatistics(getTails l)
    in
        firstMean :: restOfMeans
    end;
```

”**Medelvärdet av alla frågor** är **medelvärdet av den första frågan**, **följt av** **medelvärdena av resten av frågorna**”

Övningar

- Ett tal är delbart med tre om dess siffersumma är delbar med tre: 252414 => 2 + 5 + 2 + 4 + 1 + 4 = 18 = 6*3

  Skriv ett program som använder den här metoden för att bestämma om ett ett tal är delbart med tre, utan att använda `n mod 3 = 0`

- I Rövarspråket dubblerar man alla konsonanter och lägger ett ”o” mellan. Vokaler lämnas som de är. Skriv ett program som översätter en sträng till Rövarspråket:

  rovar(”hej på dig”) => ”hohejoj popå dodigog”

- Skriv en funktion som hittar det längsta ordet i en sträng och returnerar det i en tupel tillsammans med ordets längd.