Some Practical Information
+
Programming with Lists

Koen Lindström Claessen

Exercises

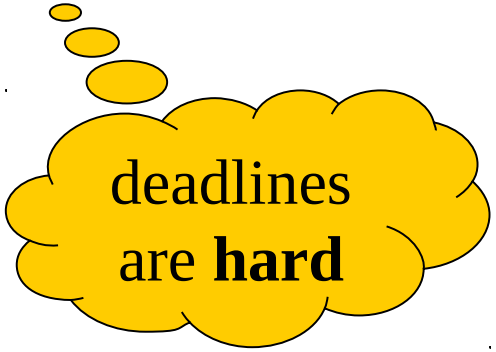
Did you go to the exercises yesterday?

Lab Assignments

- Total of 4 assignments
  - Power function
  - BlackJack (2 parts)
  - Sudoku (2 parts)
  - Graphical calculator (2 parts)

Each Lab has Three Deadlines

- **First** deadline:
  - initial part of the lab
  - serious try
- **Second** deadline: 1 week later
  - complete lab
  - serious try
  - not perfect -- feedback
- **Final** deadline: 1.5 weeks later
  - Can submit several times over this period
  - Each time you get new feedback
  - Final, correct solution has to be submitted before final deadline



symbols

Lab Feedback

- **-- Your function f does not work**
  - Denote something that has to be corrected and submitted again
- **== Your function f is a bit too complicated**
  - Denote something that has to be corrected only if the lab has to be submitted anyway
- **** I see you have solved the problem**
  - Just a regular comment, nothing to correct
- **++ Your implementation of f is better than mine!**
  - Something extra good, should of course not be corrected

Missing a Deadline

- Submitting after the deadline
  - In principle: **Unacceptable**
  - Submit what you have done
    - even if it is not finished
    - You might get one more chance
  - Good reason: Contact us BEFORE the deadline
- New opportunity: Next year!

Cheating (fusk)

- UNACCEPTABLE
  - Using someone else’s code
  - Showing your code to someone else
    - Copying
    - E-mailing
    - Printing
    - Pen-and-paper writing
  - Copying code from the web

Instead…

- If you have problems
  - Talk to us (course assistants)
  - We are nice, reasonable people
    - More time (if needed)
    - More help
  - Wait until next year
  - DO NOT CHEAT!

If Cheating Happens… ☹

- We report this to
  - Disciplinary board (Chalmers)
  - Disciplinary board (GU)
- You might be suspended (”avstängd”)
  - 1 – 3 months (no studiemedel)
  - This has actually happened...
- You might be expelled

Cheating Detection

- Lab graders
  - Discovery of similar solutions
  - Similar:
    - Changing comments
    - Changing layout
    - Changing names of functions and variables
- At the end of the course
  - Automatic software system
    - Pairwise similarity of solutions

Allowed

- Orally discuss exercises
- Orally discuss lab assignments
- Orally discuss solutions

- Web-based discussion board
  - General questions
  - Specific questions
  - Finding a lab partner
  - …

Lab Assignments

- Booking lists
  - Book one block at a time
- Extra assignments
  - For your own pleasure
  - **No bonus points**

Att Lämna In

- Skapa en grupp i Fire
  - 2 personer (inte 1, inte 3)
  - Båda två ska gå med i gruppen
- ”Submit” i Fire
  - klicka på ”submit” efter uppladdningen av filerna

”Clean Code”

- Before you submit your code, clean it up!
  - Polite thing to do
  - Easier for us to understand your code
  - Easier for you to understand your code!
  - We will reject your solution if it is not clean
  - Important!
- To clean your code:
  - No long lines (<78 characters)
  - Consistent layout
  - Good comments
  - No ”junk” – unused code, unneccessary comments
  - No overly complicated function definitions

Kursevaluering

- Kursen utvärderas av er
  - 3 studentrepresentanter
  - kursenkät
- Prata med representanterna
  - kommentar
  - förslag

Recursive Datatypes and Lists

Koen Lindström Claessen

Types vs. Constructors

a *type*

a *function*

a *constructor function*

```
data Card = Card Rank Suit
```

the *type*

```
colourCard :: Card -> Colour
colourCard (Card r s) = colour s
```

the *constructor function*

Types vs. Constructors

a *constructor function*

a *type*

```
data Card = MkCard Rank Suit
```

the *type*

```
colourCard :: Card -> Colour
colourCard (MkCard r s) = colour s
```

the *constructor function*

Reminder: Modelling a Hand

- A Hand is either:
  - An empty hand
  - Formed by *adding a card* to a smaller hand

```
data Hand = Empty | Add Card Hand
    deriving Show
```

- Discarding the first card:

```
discard :: Hand -> Hand
discard (Add c h) = h
```

Lists

-- how they work

Lists

```
data List = Empty | Add ?? List
```

- A list is either:
  - An empty list
  - Formed by *adding an element* to a smaller list
- What to put on the place of the ??

Lists

```
data List a = Empty | Add a (List a)
```

- A *type parameter*
- Add 12 (Add 3 Empty) :: List Integer
- Add ”apa” (Add ”bepa” Empty) :: List String

Lists

```
data List a = Empty | Add a (List a)
```

- Empty :: List Integer
- Empty :: List Bool
- Empty :: List String
- ...

Lists

- Can represent 0, 1, 2, … things
  - [], [3], [”apa”,”katt”,”val”,”hund”]
- They all have the same type
  - [1,3,True,”apa”] is not allowed
- The order matters
  - [1,2,3] /= [3,1,2]
- Syntax
  - 5 : (6 : (3 : [])) == 5 : 6 : 3 : [] == [5,6,3]
  - ”apa” == [’a’,’p’,’a’]

Different Notation

```
data List a = Empty
            | Some a (List a)
```

```
data [a] =  []
         |  a : [a]
```

uttal: ”cons”

More Notation

list-type

```
length :: [a] -> Int
```

list with one element

```
[12]
```

```
12 : []
```

```
[12, 0, 3, 17, 123]
```

```
12 : (0 : (3 : (17 : (123 : []))))
```

Quiz

- Vad är typen på funktionen [] ?

```
[] :: [a]
```

- Vad är typen på funktionen (:) ?

```
(:) :: a -> [a] -> [a]
```

Programming Examples

- empty
- first / last
- maximum
- append (+++)
- reverse (rev)
- value :: String -> Integer

- (see files Lists0.hs and Lists1.hs)

Lists

- Can represent 0, 1, 2, … things
  - [], [3], [”apa”,”katt”,”val”,”hund”]
- They all have the same type
  - [1,3,True,”apa”] is not allowed
- The order matters
  - [1,2,3] /= [3,1,2]
- Syntax
  - 5 : (6 : (3 : [])) == 5 : 6 : 3 : [] == [5,6,3]
  - ”apa” == [’a’,’p’,’a’]

More on Types

- Functions can have ”general” types:
  - *polymorphism*
  - reverse :: [a] -> [a]
  - (++) :: [a] -> [a] -> [a]
- Sometimes, these types can be restricted
  - Ord a => … for comparisons (<, <=, >, >=, …)
  - Eq a => … for equality (==, /=)
  - Num a => … for numeric operations (+, -, *, …)

Do’s and Don’ts

```
isBig :: Integer -> Bool
isBig n | n > 9999  = True
        | otherwise = False
```

guards and boolean results

```
isBig :: Integer -> Bool
isBig n = n > 9999
```

Do’s and Don’ts

```
resultIsSmall :: Integer -> Bool
resultIsSmall n = isSmall (f n) == True
```

comparison with a boolean constant

```
resultIsSmall :: Integer -> Bool
resultIsSmall n = isSmall (f n)
```

Do’s and Don’ts

```
resultIsBig :: Integer -> Bool
resultIsBig n = isSmall (f n) == False
```

comparison with a boolean constant

```
resultIsBig :: Integer -> Bool
resultIsBig n = not (isSmall (f n))
```

nd Don’ts

Do not make unnecessary case distinctions

```
fun1 :: [Integer] -> Bool
fun1 []       = False
fun1 (x:xs) = length (x:xs) == 10
```

necessary case distinction?

repeated code

```
fun1 :: [Integer] -> Bool
fun1 xs = length xs == 10
```

s and Don’ts

Make the base case as simple as possible

```
fun2 :: [Integer] -> Integer
fun2 [x]      = calc x
fun2 (x:xs) = calc x + fun2 xs
```

right base case ?

repeated code

```
fun2 :: [Integer] -> Integer
fun2 []       = 0
fun2 (x:xs) = calc x + fun2 xs
```