COMP3141

Curtis Millar
CSE, UNSW (and Data61)
Term 2 2020

Recap: What is this course?

Software must be high quality: **correct, safe and secure.**

Software must developed **cheaply and quickly**

Recall: Safety-critical Applications

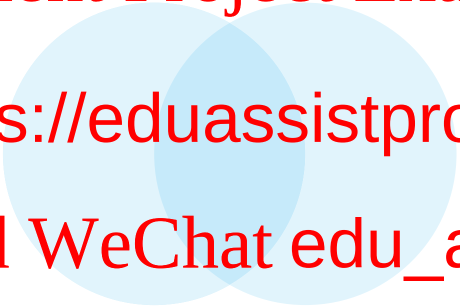
For safety-critical applications, failure is not an option:

- planes, self-
- rockets, M
- drones, nuclear missiles
- banks, hedge funds, cryptocurrency exchanges
- radiation therapy machines, artificial cardiac pacem

Safety-critical Applications

rac-25
as directly respon-
sible he 1980s
whe ties of
beta r

COMP3141: Functional Programming



Functional Programming: How does it Help?

1. Close to Mat
2. Types: act as
3. Property-
4. Verification: equational reasoning eases proofs (in W

COMP3141: Learning Outcomes

1. Identify basic Haskell **type errors** involving concrete types.
2. Work comfortably with **GHCi** on your working machine.
3. Use Haskell **if** etc.
4. Understan e (.) and ($) operators.
5. Write Haskell programs to manipulate **lists** with recursion.
6. Makes use of **higher order functions** like
7. Use λ-**abstraction** to define anonymous functio
8. Write Haskell programs to compute **basic arithmetic**, **character, and string manipulation**.
9. Decompose problems using **bottom-up design.**

Functional Programming: History in Academia

1930s Alonzo Church developed lambda calculus (equiv. to Turing Machines)

1950s John McCarthy developed Lisp (LISt Processor, first FP language)

1960s Peter Landi

1970s John Back ns, reasoning

1970s Robin Milner and others developed ML (Meta-Langu language, polymorphic types, type inference)

1980s David Turner developed Miranda (lazy, predecesso

1987- An international PL committee developed Haskell (named after the logician Curry Haskell)

... received Turing Awards (similar to Nobel prize in CS).

Functional programming is now taught at most CS departments.

Functional Programming: Influence In Industry

- Facebook's motto was:
  - "Mov
  - as they e
  - now F

- JaneStreet, Facebook, Google, Microsoft, Intel, Apple (... and the list goes on)
- Facebook building React and Reason, Apple pivoting t MapReduce.

Closer to Maths: Quicksort Example

Let's solve a problem to get some practice:

Example (Quicksort, recall from Algorithms)

Quicksort is a divide and conquer algorithm.

1. Picks a pivot f
2. Divides the a
   and the large
3. Recursively sorts the sub-components.

- What is the average complexity of Quicksort?
- What is the worst case complexity of Quicksort?
- Imperative programs describe **how** the program works.
- Functional programs describe **what** the program does.

Quicksort Example (Imperative)

```
algorithm quicksort(A, lo, hi) is
    if lo < hi then
        p := partition(A, lo, hi)
        qui
        qui

algorithm par
    pivot := A[hi]
    i := lo
    for j := lo to hi - 1 do
        if A[j] < pivot then
            swap A[i] with A[j]
            i := i + 1
    swap A[i] with A[hi]
    return i
```

Quick Sort Example (Functional)


```haskell
qsort :: Ord a => [a] -
qsort [] = []
qsort (x:xs) = qs

        smaller = filter (\ a-> a <= x) xs
        larger = filter (\ b-> b > x) xs
```


Is that it? Does this work?

Practice Types

In the previous lecture, you learned about the importance of types in functional programming. Let's practice figuring out the types of terms.

1. `True :: Bool`
2. `'a' :: Cha`
3. `['a', 'b', '`
4. `"abc" :: [`
5. `["abc"] :: [[Char]]`
6. `[('f',True), ('e', False)] :: [(Char`

- In Haskell and GHCi using `:t`.
- Using Haskell documentation and GHCi, answer the questions in this week's quiz (assessed!).

COMP3141: Learning Outcomes

1. Identify basic Haskell **type errors** involving concrete types.
2. Work comfortably with **GHCi** on your working machine.
3. Use Haskell **if** etc.
4. Understan e (.) and ($) operators.

5. Write Haskell programs to manipulate **lists** with recursion.
6. Makes use of **higher order functions** like
7. Use λ-**abstraction** to define anonymous functio

8. Write Haskell programs to compute **basic arithmetic**, **character, and string manipulation**.
9. Decompose problems using **bottom-up design.**

Recall: Higher Order List Functions

The rest of last lecture was spent introducing various list functions that are built into Haskell's standard library by way of live coding.

Functions cove

1. `map`
2. `filter`
3. `concat`
4. `sum`
5. `foldr`
6. `foldl`

In the process, you saw **guards** and **if**, and the `.` operator.

Higher Order List Functions

The rest of last lecture was spent introducing various list functions that are built into Haskell's standard library by way of live coding.

Functions covered:

1. `map`
2. `filter`
3. `concat`
4. `sum`
5. `foldr`
6. `foldl`

In the process, you saw **guards** and **if**, and the `.` operator.

Let's do that again in Haskell.

COMP3141: Learning Outcomes

1. Identify basic Haskell **type errors** involving concrete types.
2. Work comfortably with **GHCi** on your working machine.
3. Use Haskell **if** etc.
4. Understan e (.) and ($) operators.

5. Write Haskell programs to manipulate **lists** with recursion.
6. Makes use of **higher order functions** like
7. Use λ-**abstraction** to define anonymous functio

8. Write Haskell programs to compute **basic arithmetic**, **character, and string manipulation**.
9. Decompose problems using **bottom-up design.**

Numbers into Words

Let's solve a problem to get some practice

Example (Demo Task)

Given a number … tring form) that describes the num…

We must:

1. Convert single-digit numbers into words (0 …
2. Convert double-digit numbers into words (0 …
3. Convert triple-digit numbers into words ($0 \leq n < 1000$).
4. Convert hexa-digit numbers into words ($0 \leq n < 1000000$).

Single Digit Numbers into Words

$$0 \leq n < 10$$


```haskell
units :: [Strin
units =
    ["zero"
     "six", "seven", "eight", "nine", "ten"]

convert1 :: Int -> String
convert1 n = units !! n
```

Double Digit Numbers into Words

$$0 \leq n < 100$$


```
teens :: [String]
teens =
    ["ten", "
     "fifte
     "nineteen"]
```



```
tens :: [String]
tens =
    ["twenty", "thirty", "fourty", "fifty", "sixty",
     "seventy", "eighty", "ninety"]
```

Double Digit Numbers into Words Continued $(0 \leq n < 100)$


```
digits2 :: Int -> (Int, Int)
digits2 n = (div n 10, mod n 10)
combine2 :: (In
combine2 (t, u)
    | t == 0             = convert1 u
    | t == 1             = teens !! u
    | t > 1 && u == 0    = tens !! (t-2)
    | t > 1 && u /= 0    = tens !! (t-2)
                           ++ "-" ++ convert1 u
convert2 :: Int -> String
convert2 = combine2 . digits2
```

Infix Notation

Instead of

```
digits2 n = (div n 10
```


for **infix** notati

```
digits2 n = (n `div` 10, n `mod` 10)
```


Note: this is not the same as single quote used for

Simpler Guards but Order Matters

You could also simplify the guards as follows:

```
combine2 :: (In
combine2 (t,u
  | t == 0      = convert1 u
  | t == 1      = teens !! u
  | u == 0      = tens !! (t-2)
  | otherwise   = tens !! (t-2) ++ "-" ++ convert1 u
```

but now the order in which we write the equations is crucial.   is a synonym for True.

Where instead of Function Composition

Instead of implementing convert2 as digit2 . combine2, we can implement it directly using the where keyword:


```
convert2 :: Int -> String
convert2 n
  | t == 0       = convert1 u
  | t == 1       = teens !! u
  | u == 0       = tens !! (t-2)
  | otherwise    = tens !! (t-2) ++ "-" ++ convert1 u
  where (t, u) = (n `div` 10, n `mod` 10)
```

Triple Digit Numbers into Words

$(0 \leq n < 1000)$


```
convert3 :: Int -

convert3 n
    | h == 0        = convert2 n
    | t == 0        = convert1 h ++ "hundred"
    | otherwise     = convert1 h ++ " hundred and "
                      ++ convert2 t
    where (h, t) = (n `div` 100, n `mod` 100)
```

Hexa Digit Numbers into Words

$(0 \leq n < 1000000)$


```
convert6 :: Int -> String

convert6 n
    |  m == 0       = convert3 n
    |  h == 0       = convert3 m ++ "th
    |  otherwise    = convert3 m ++ link h ++ convert3 h
   where (m, h) = (n `div` 1000, n `mod` 1000)

link :: Int -> String
link h = if (h<100) then " and " else " "

convert :: Int -> String
convert = convert6
```

COMP3141: Learning Outcomes

1. Identify basic Haskell **type errors** involving concrete types.
2. Work comfortably with **GHCi** on your working machine.
3. Use Haskell **if** etc.
4. Understan e (.) and ($) operators.
5. Write Haskell programs to manipulate **lists** with recursion.
6. Makes use of **higher order functions** like
7. Use λ-**abstraction** to define anonymous functio
8. Write Haskell programs to compute **basic arithmetic**, **character, and string manipulation**.
9. Decompose problems using **bottom-up design.**

Homework

1. Get Haskell
course web

2. Using Hask
(assessed!).