# EECS 368
# Programming Language Paradigms

David O. Johnson

Fall 2022

# Reminders

- Assignment 5 due: 11:59 PM, Monday, October 31
  - One of our SIs, Soujanya, has come up with a great way for you to create, test, and submit Assignment 5.
  - Please review the StudentVideoDemo at the following link for more information:
  - https://drive.google.com/drive/folders/1n1R5b3YihQcbCVyQwUjVBD1IMGMdEvBE
  - If you have any question, please contact Soujanya Ambati at: saisoujanyaambati@ku.edu
  - Soujanya will also be grading Assignment 5.
- Guest Lecture (Nick Smith): Wednesday, November 2
  - Attendance Required
  - Submit a written report on the guest lecture in place of the In-Class Problem (see instructions and rubric in Canvas Lectures module)
- Assignment 6 due: 11:59 PM, Monday, November 14

# Any Questions?

# In-Class Problem Solution

- 25-(10-24) In-Class Problem Solution.pptx

# Any Questions?

# Help on Haskell

- Here is another free book that might help you learn Haskell:

  http://learnyouahaskell.com/

# Glasgow Haskell Compiler

- GHC is the leading implementation of Haskell, and comprises a compiler and interpreter.
- The interactive nature of the interpreter makes it well suited for teaching and prototyping.
- GHC is freely available from:

  https://www.haskell.org/downloads

# Access to GHC at KU

- GHC is available on the KU cycle servers:

  cycle1.eecs.ku.edu through cycle4.eecs.ku.edu
- Any modern ssh client should work, but the two most popular are Putty and the one inside Visual Studio.
- Some users have also been using the Powershell ssh client.
- You'll use your standard online id and password.

- GHC is installed on all Linux machines in Eaton Hall.
- Eaton 1005B and 1005C, and 1005D are all Linux, as well as the first two rows outside of 1005D.
- Putty is installed on all the EECS Windows stations, so you can also access the cycles from there.

# Starting GHCi

- The interpreter can be started from the terminal command prompt $ by simply typing ghci:

```
$ ghci

GHCi, version X: http://www.haskell.org/ghc/  :? for help

Prelude>
```

- The GHCi prompt > means that the interpreter is now ready to evaluate an expression.

# Using GHCi

- For example, it can be used as a desktop calculator to evaluate simple numeric expressions:

```
> 2+3*4
14

> (2+3)*4
20

> sqrt (3^2 + 4^2)
5.0
```

# Useful GHCi Commands

| Command | Meaning |
|---|---|
| :load *name* | load script *name* |
| :reload | reload current script |
| :set editor *name* | set editor to *name* |
| :edit *name* | edit script *name* |
| :edit | edit current script |
| :type *expr* | show type of *expr* |
| :? | show all commands |
| :quit | quit GHCi |

# Any Questions?

# The Standard Prelude

- Haskell comes with a standard library of functions called: Prelude.
- Prelude includes the 5 main arithmetic operations of:

  - Addition: +

    ```
    > 2 + 3
    5
    ```

  - Subtraction: -

    ```
    > 2 - 3
    -1
    ```

  - Multiplication: *

    ```
    > 2 * 3
    6
    ```

  - Division: `div`
    - div is enclosed in back quotes (`), not forward quotes
    - Integer division; rounds down to nearest integer

    ```
    > 7 `div` 2
    3
    ```

  - Exponentiation: ^

    ```
    > 2 ^ 3
    8
    ```

# The Standard Prelude

- In addition to the arithmetic operations, Prelude also provides many useful functions on lists.

- Select the first element of a list:

```
> head [1,2,3,4,5]
1
```

- Remove the first element from a list:

```
> tail [1,2,3,4,5]
[2,3,4,5]
```

- Select the nth element of a list (Note: 0 based indexing):

```
> [1,2,3,4,5] !! 2
3
```

# Any Questions?

# Function Application

Recall from last time:

- In mathematics, function application is denoted using parentheses, and multiplication is often denoted using juxtaposition or space.

f(a,b) + c d

- Apply the function f to a and b, and add the result to the product of c and d.

- In Haskell, function application is denoted using space, and multiplication is denoted using *.

f a b + c * d

- Apply the function f to a and b, and add the result to the product of c and d.

- Moreover, function application is assumed to have higher priority than all other operators.

f a + b

- Means (f a) + b, rather than f (a + b)

# Function Application Examples

| Mathematics | Haskell |
| --- | --- |
| `f(x)` | `f x` |
| `f(x,y)` | `f x y` |
| `f(g(x))` | `f (g x)` |
| `f(x,g(y))` | `f x (g y)` |
| `f(x)g(y)` | `f x * g y` |

# Any Questions?

# Haskell Scripts

- As well as the functions in the standard library, you can also define your own functions.
- New functions are defined within a script
  - a text file comprising a sequence of definitions
- By convention, Haskell scripts usually have a .hs suffix on their filename.
- This is not mandatory, but is useful for identification purposes.
- When developing a Haskell script, it is useful to keep two windows open:
  - one running an editor for the script
  - and the other running GHCi

# My First Script

- Start an editor, type in the following two function definitions, and save the script as test.hs:

```
double x = x + x
quadruple x = double (double x)
```

- Leaving the editor open, in another window start up GHCi with the new script:

```
$ ghci test.hs
```

- Now both the standard library and the file test.hs are loaded, and functions from both can be used:

```
> quadruple 10
40

> take (double 2) [1,2,3,4,5,6]
[1,2,3,4]
```

# My First Script Revision

- Leaving GHCi open, return to the editor, add the following two definitions, and resave:

```
factorial n = product [1..n]
average ns = sum ns `div` length ns
```

- Note: div is enclosed in back quotes (`), not forward quotes.
- GHCi does not automatically detect that the script has been changed.
- So a reload command must be executed before the new definitions can be used.

```
> :reload
Reading file "test.hs"

> factorial 10
3628800

> average [1,2,3,4,5]
3
```

# Any Questions?

# Integers vs. Floating Point

- Start an editor, type in the following two function definitions, and save the script as Example2.hs:

```
average xs = sum xs / length xs
$ ghci Example.hs

Could not deduce (Fractional Int) arising from a use of ‘/’
    from the context: Foldable t
      bound by the inferred type of average :: Foldable t => t Int -> Int
      at <interactive>:3:1-31
  • In the expression: sum xs / length xs
    In an equation for ‘average’: average xs = sum xs / length xs
```

# Integers vs. Floating Point (fixed)

- The library function fromIntegral converts an integer into a floating-point number.

```
average xs = fromIntegral (sum xs) / fromIntegral (length xs)
```

```
$ ghci Example.hs
> average [1,2]
1.5
```

# Any Questions?

# Naming Requirements

- Function and argument names must begin with a lower-case letter and may be followed by one or more letters (both lower- and upper-case), digits, underscores, or single quotes (').
- For example:

```
myFun
```

```
fun1
```

```
arg_2
```

```
x'
```

- By convention, list arguments usually have an s suffix on their name.
- For example:

```
xs
```

```
ns
```

```
nss
```

# The Layout Rule

In a sequence of definitions, each definition must begin in precisely the same column:

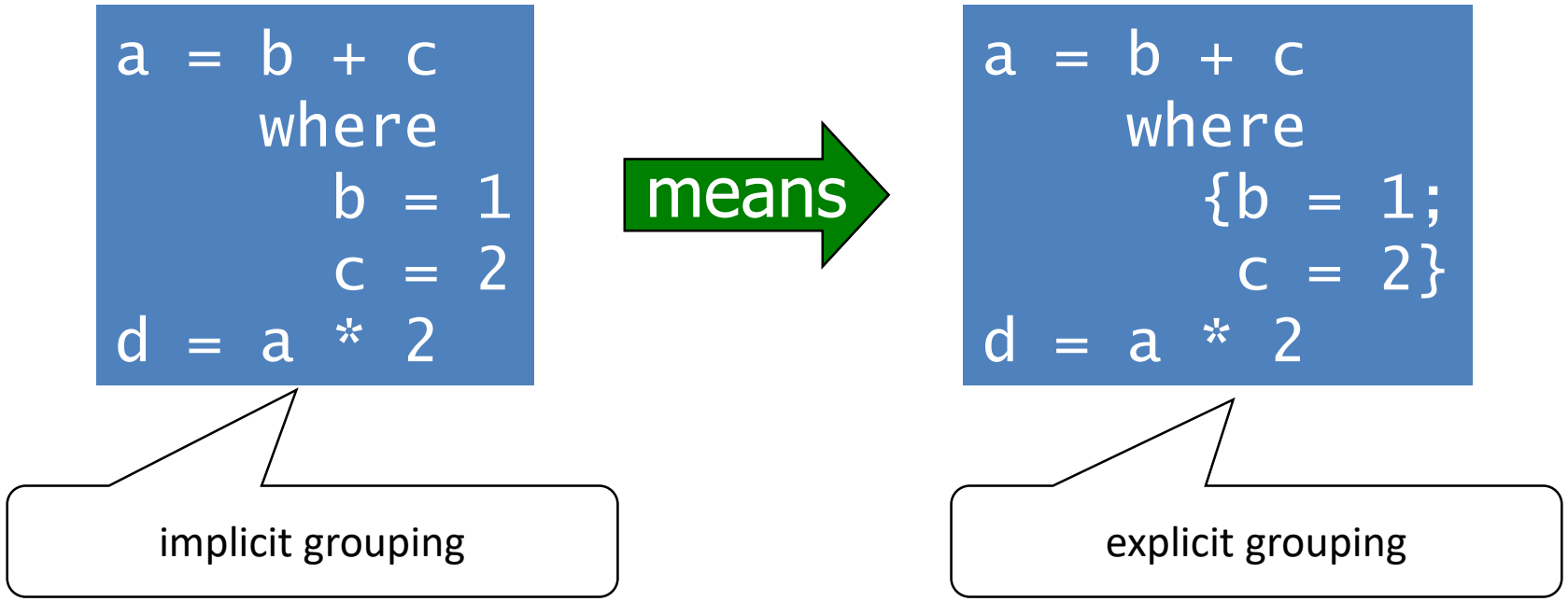
```
a = 10
b = 20
c = 30
```

```
a = 10
 b = 20
c = 30
```

```
  a = 10
b = 20
  c = 30
```

# The Layout Rule

- The layout rule avoids the need for explicit syntax to indicate the grouping of definitions.



# Comments

**Haskell supports two kinds of comments in scripts:**

- Ordinary:
  - Begin with -- and extend to the end of the line

```
-- Factorial of a positive integer
factorial n = product [1..n]
```

- Nested:
  - Begin and end with the {- and -} and may span more than one line:

```
{-
Factorial of a positive number
-}
factorial n = product [1..n]
```

  - Useful for temporarily removing sections of a script while debugging:

```
{-
double x = x + x
quadruple x = double (double x)
-}
```

# Any Questions?

# In-Class Problem

The script below contains three syntactic errors:

```
N = a 'div' length xs
  where
    a = 10
  xs = [1,2,3,4,5]
```

1. What are the three errors?
2. What is the correct script syntax?