Getting Started with Haskell

COMP2209 - Programming III

Dr Julian Rathke
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:

  www.haskell.org/platform

• If you haven’t done so already then I recommend installing this on your machine.
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.
Simple numeric expressions

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
```
Haskell comes with a large number of built-in functions which are known as the Standard Prelude. This library is loaded by default.

The Prelude contains familiar arithmetic and boolean functions as well as many useful functions on lists and strings.

**Arithmetic Functions**: + * - div mod abs

**Comparison Functions**: > >= == /= <= <

**Example expressions**:

```
3 * 7    (*) 3 7    mod 10 2    10 `mod` 2
1 - (2 * 3)    (1 - 2) * 3    5 >= (1 + 2)    5 >= (-5)
```

Familiar rules for bracketing and operator precedence apply.
Standard List functions

Just as in LISP, Haskell uses lists as a built-in structured data type. For this reason, there are many built-in list functions:

Select the first element of a list:
> head [ 1, 2, 3, 4, 5 ]
1

Remove the first element of a list:
> tail [ 1, 2, 3, 4, 5 ]
[ 2, 3, 4, 5 ]

Calculate the length of a list:
> length [ 1, 2, 3, 4, 5 ]
5
More List functions

Select the nth element of a list:

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

(lists are 0 indexed)

Select the first n elements of a list:

> take 3 [ 1, 2, 3, 4, 5 ]
[ 1, 2, 3 ]

Remove the first n elements of a list:

> drop 3 [ 1, 2, 3, 4, 5 ]
[ 4, 5 ]

Append two lists:

> [ 1, 2, 3 ] ++ [ 4, 5 ]
[ 1, 2, 3, 4, 5 ]
Even More List functions

Calculate the sum of the elements of a list:
> sum [ 1, 2, 3, 4, 5 ]
15

Calculate the product of the elements of a list:
> product [ 1, 2, 3, 4, 5 ]
120

Reverse a list:
> reverse [ 1, 2, 3, 4, 5 ]
[ 5, 4, 3, 2, 1 ]

Create an infinite list of identical elements:
> repeat 1
[ 1, 1, 1, 1, ... ???? ]
In mathematics, function application is denoted using parentheses, and multiplication is often denoted using juxtaposition or space.

\[ f(a,b) + c \cdot d \]

We understand this to mean: 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 juxtaposition and multiplication is denoted using \( \ast \)

\[ f \; a \; b \; + \; c \; \ast \; d \]

This just means the same thing as the mathematics but it is written in Haskell syntax.
Function Application

Function application is an operator of the language and it has higher precedence than all other operators, that is

\[ f(a + b) \] means \( (f(a) + b) \) ✔ not \( f(a + b) \) ✘

It is also left associative which means that

\[ f(a b c) \] is to be read as

\[ ((f(a)b)c) \]
# Quick Quiz

<table>
<thead>
<tr>
<th>Mathematics</th>
<th>Haskell</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(x)$</td>
<td>???</td>
</tr>
<tr>
<td>$f(x, y)$</td>
<td>???</td>
</tr>
<tr>
<td>$f(g(x))$</td>
<td>???</td>
</tr>
<tr>
<td>$f(x, g(y))$</td>
<td>???</td>
</tr>
<tr>
<td>$f(x)g(y)$</td>
<td>???</td>
</tr>
</tbody>
</table>
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.
My First Script

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.

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

```haskell
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]

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

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

These are back quotes
My First Script

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:

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

> factorial 10
3628800

> average [1,2,3,4,5]
3
```
### Useful GHCi Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>:load name</td>
<td>load script name</td>
</tr>
<tr>
<td>:reload</td>
<td>reload current script</td>
</tr>
<tr>
<td>:set editor name</td>
<td>set editor to name</td>
</tr>
<tr>
<td>:edit name</td>
<td>edit script name</td>
</tr>
<tr>
<td>:edit</td>
<td>edit current script</td>
</tr>
<tr>
<td>:type expr</td>
<td>show type of expr</td>
</tr>
<tr>
<td>:?</td>
<td>show all commands</td>
</tr>
<tr>
<td>:quit</td>
<td>quit GHCi</td>
</tr>
</tbody>
</table>
Naming requirements

Function and argument names must begin with a lower case letter

myFun  noFunAtAll  arg1  arghh’

You can define new operators and use symbols to name them

\[ (>£) \ x \ y = x + y \]
\[ > 5 \ £ \ 6 \]
\[ 11 \]

It’s not a rule but by convention any arguments representing lists usually have an s suffix on their name

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

- Correct: a = 10
  b = 20
  c = 30

- Incorrect: a = 10
  b = 20
  c = 30

- Incorrect: a = 10
  b = 20
  c = 30
The layout rule avoids the need for explicit syntax to indicate the grouping of definitions.

- **Implicit Grouping**
  
  \[
  a = b + c \\
  \text{where} \\
  b = 1 \\
  c = 2 \\
  d = a \times 2
  \]

- **Explicit Grouping**
  
  \[
  a = b + c \\
  \text{where} \\
  \{b = 1; \\
  c = 2\} \\
  d = a \times 2
  \]
YOUR QUESTIONS

Next Lecture:
Types and Classes.