PROGRAMMING IN HASKELL

Chapter 2 - First Steps

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
```

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.

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

```
> 2+3*4
14

> (2+3)*4
20

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

The Standard Prelude

Haskell comes with a large number of standard library functions. In addition to the familiar numeric functions such as + and *, the library 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:

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

- Select the first n elements of a list:

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

- Remove the first n elements from a list:

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

- Calculate the length of a list:

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

- Calculate the sum of a list of numbers:

```
> sum [1,2,3,4,5]
15
```

- Calculate the product of a list of numbers:

```
> product [1,2,3,4,5]
120
```

- Append two lists:

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

- Reverse a list:

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

Function Application

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
```

As previously, but in Haskell syntax.

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).

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`

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:

```
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:

```
factorial n = product [1..n]

average ns = sum ns `div` length ns
```

Note:

- div is enclosed in back quotes, not forward;
- x `f` y is just syntactic sugar for f x y.

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
```

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
:! ‘cmd line cmd’	runs the cmd

Naming Requirements

- Function and argument names must begin with a lower-case letter. For example:

`myFun`   `fun1`   `arg_2`   `x’`

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

`xs`   `ns`   `nss`

Comments in Haskell (1)

- In Haskell, we strive to write programs clear enough to be understandable without comments. (see https://wiki.haskell.org/Commenting)

- Is it a comment that documents unexpected behaviour? Then avoid unexpected behaviour!

- Is it an obvious comment? Then leave it out!

```haskell
-- swap the elements of a pair
swap :: (a,b) -> (b,a)
```

Comments in Haskell (2)

- Does the comment duplicate the Haskell report?

```
let b = a+1 -- add one to ‘a’
```

At least describe the intention of your code.

```
let b = a+1 -- increase the loop counter‘a’
```

- Writing programs that do not require comments is the better choice

```
let newCounter = oldCounter + 1
```

Comments in Haskell (3)

- But when you write comments …

```
-- A one-line comment looks like this
```

```
{- A multiline
     comment can continue for many lines
-}
```

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 avoids the need for explicit syntax to indicate the grouping of definitions.

```
a = b + c
    where
      b = 1
      c = 2
d = a * 2
```

means

```
a = b + c
    where
      {b = 1;
       c = 2}
 d = a * 2
```

implicit grouping

explicit grouping