Concepts of Higher Programming Languages
First Steps in Haskell

Jonathan Thaler

Department of Computer Science

FH Vorarlberg
University of Applied Sciences

Glasgow Haskell Compiler (GHC)

- 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**, check the lecturenotes [1] on how to install the Haskell Platform with GHC and the interpreter GHCi and how use it;

[1] `https://homepages.fhv.at/thjo/lecturenotes/concepts/running-haskell.html`

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]
```

The Standard Prelude

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]
```

The Standard Prelude

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
```

The Standard Prelude

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`.

Function Application

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

```
f a b + c*d
```

As previously, but in Haskell syntax.

Function Application

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

Math	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 using 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`:

Test.hs

```
double x = x + x

quadruple x = double (double x)
```

My First Script

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

Console

```
ghci Test.hs
```

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

GHCi Session

```
> quadruple 10
40

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

My First Script

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

Test.hs

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

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

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

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:

GHCi Session

```
> :reload
-- Ok one module loaded.

> :r
-- Ok one module loaded.

> factorial 10
3628800

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

Naming Requirements

Function and Argument Names

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

```
myFun    fun1    arg_2    x'
```

List Arguments

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

```
xs    ns    nss
```

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
```

Layout Rule

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
```

implicit grouping


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

explicit grouping