NAMAL COLLEGE

UNIVERSITY of BRADFORD

**Symbolic & Declarative Computing / Artificial Intelligence (COS5012-B)**

October 04, 2016 Week 2 Lab1

# LAB MANUAL: HASKELL BASICS

In this tutorial you will learn

- Expressions
- Functions
- Precedence of operators and functions
- Equations

## Integer Expressions

- We start with some simple integer arithmetic. At the prompt on the left there you can type in Haskell expressions. Type an integer number, e.g. 42, and observe that it evaluates to itself.

```
Hugs> 42
42
```

- Now type a simple integer arithmetic operation, e.g. 6*7, and observe that it evaluates to the expected result.

```
Hugs> 6*7
42
```

## Syntax of Expressions

- You can use parenthesis to group sub-expressions, e.g. `(3+4)*6`, but they are optional. The arithmetic operations have the same precedence as in maths. For example `3+4*6` means `3+(4*6)`.
- You can let Haskell prove this for you: try `3+(4*6) == 3+4*6`.

```
Hugs> 3+(4*6) == 3+4*6
True
```

So this expression returned `True` and it illustrates the use of the equality test operator.

You can nest as many parentheses as you like (even if it looks silly): `((6))*(((7)))`

## Special Cases

- There are some special cases, in particular regarding the '-' sign. For example, try `4+-3`.

  As you can see, this fails: Haskell thinks you wanted to use a special operation '+-'.

- Now, try 4+ -3 (that's right, just an extra space).

  Again, that did not work as expected: Haskell does not allow you to combine 'infix' operations (like 3+4) with 'prefix' operations (like '-4').

- So what should we do? Enclose the infix operation in parentheses: 4+(-3)

  And yes, that one worked! So in general it is best to enclose negative numbers with parentheses in expressions.

## Boolean Algebra

Boolean algebra is also pretty straightforward. As you probably know, `&&` means a boolean and, `||` means a boolean or. `not` negates a True or a False.

```
Hugs> True && False
False
Hugs> True && True
True
Hugs> False || True
True
Hugs> not False
True
Hugs> not (True && True)
False
```

## Functions

A function has a textual name (e.g. 'abs' or 'sin') rather than an operator. It takes argument(s), performs some computation, and produces result(s).

- To use a function in Haskell, you apply it to an argument: write the function followed by the argument, separated by a space. For example

```
Hugs> abs 7
7
```

  As expected, applying abs to a positive number just returns its value.

- Now let's try a negative number:

```
Hugs> abs (-3)
3
```

And indeed, applying abs to a negative number returns the absolute value.

- Functions can take several arguments, e.g. min and max take two arguments. The arguments are given after the function, separated by whitespace. For example,

```
Hugs> min 3 8
3
Hugs> max 3 8
8
```

See? No need for parentheses!

- To combine functions you need to know their precedence. In Haskell this is simple: Function application binds tighter than anything else. For example

```
Hugs> sqrt 9+7
10.0
```

Surprised? Haskell interprets this as `(sqrt 9)+7`, not `sqrt (9+7)`. So now try `sqrt (9+7)`.

So if an argument to a function is an expression, you need to put it in parentheses.

**Exercise**

So what about combining two functions? Try for example to apply '`min`' to '`max 3 4`' and '`5`'.

## Equations

Equations are used to give names to values, e.g. `answer = 42`.

Open notepad++ or your favourite editor and type `answer = 42`. Now save this file with a .hs extension. Open WinHugs and open the .hs file you just created. You will see a :load command will be executed by the WinHugs. Now type

```
Hugs> answer
42
```

- Equations give names to values. So now '`answer`' is just another name for '`42`'
- An equation in Haskell is a mathematical equation: it says that the left hand side and the right hand side denote the same value.
- The left hand side should be a name that you're giving a value to.
- So now you can say `double = answer * 2` by appending it in the .hs file.
- Type :reload in WinHugs after saving the .hs file. Now try

```
Hugs> answer
42
Hugs> double
84
```

**Exercise**

Replace double with answer. Reload it again after saving. What happened?

- In a pure functional language like Haskell, you can't define a name in terms of itself. You can only assign a name once!

## My first function

Open up your favourite text editor and punch in this function that takes a number and multiplies it by two.

```
doubleMe x = x + x
```

Functions are defined in a similar way that they are called. The function name is followed by parameters separated by spaces. But when defining functions, there's a = and after that we define what the function does. Save this as first.hs.

**Exercise**

- Load `first.hs` and execute the function `doubleMe` by supplying different parameters. You can try both integer and floating point values.
- What will be the result of `doubleMe (doubleMe 3)`?
- Make a function that takes two numbers and multiplies each by two and then adds them together.

## References


- http://learnyouahaskell.com/starting-out#ready-set-go
- https://www.futurelearn.com/courses/functional-programming-haskell