# INTRODUCTION TO FUNCTIONAL PROGRAMMING

Pieter Van Gorp & Hans Schippers

Original slides created by:

Dr. Graham Hutton

University of Nottingham

Universiteit Antwerpen

# *What is Functional Programming?*

Opinions differ, and it is difficult to give a precise definition, but generally speaking:

- Functional programming is style of programming in which the basic method of computation is the application of functions to arguments;
- A functional language is one that supports and encourages the functional style.

# *Example*

Summing the integers 1 to 10 in Java:

```
total = 0;
for (i = 1; i ≤ 10; ++i)
    total = total+i;
```

The computation method is variable assignment.

# *Example*

Summing the integers 1 to 10 in Haskell:

```
sum [1..10]
```

The computation method is function application.

Universiteit Antwerpen

# *Why is it Useful?*

Again, there are many possible answers to this question, but generally speaking:

- The abstract nature of functional programming leads to considerably simpler programs;
- It also supports a number of powerful new ways to structure and reason about programs.

Universiteit Antwerpen

# *Course Overview*

- Lab 1:
  - *The Hugs system,*
  - *Function applications,*
  - *Types (tuples, lists, function types)*
- Lab 2:
  - *Defining functions*
  - *Currying*
- Lab 3:
  - *Lists comprehensions*
  - *Recursion*
- Lab 4:
  - *Higher order functions*
- Lab 5:
  - *Defining custom types*
- Lab 6:
  - *Polymorphism, overloading,*
  - *Classes*
- Lab 7: *Monads*

Universiteit Antwerpen

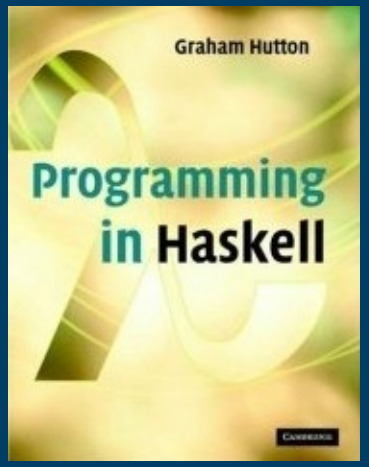
End of course:

Simple GAME Application!

Note:

- Course based upon Graham Hutton's book:

*Programming in Haskell*




Universiteit Antwerpen

# Session 1

Hugs,
Function Applications,

Types

Universiteit Antwerpen

# *What is Hugs?*

- An interpreter for Haskell, and the most widely used implementation of the language;
- An interactive system, which is well-suited for teaching and prototyping purposes;
- Hugs is freely available from:

  `www.haskell.org/hugs`

Universiteit Antwerpen

# *The Standard Prelude*

When Hugs is started it first loads the library file Prelude.hs, and then repeatedly prompts the user for an expression to be evaluated.

For example:

```
> 2+3*4
14

> (2+3)*4
20
```

Universiteit Antwerpen

The standard prelude also provides many useful functions that operate on lists. For example:

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

> product [1,2,3,4]
24

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

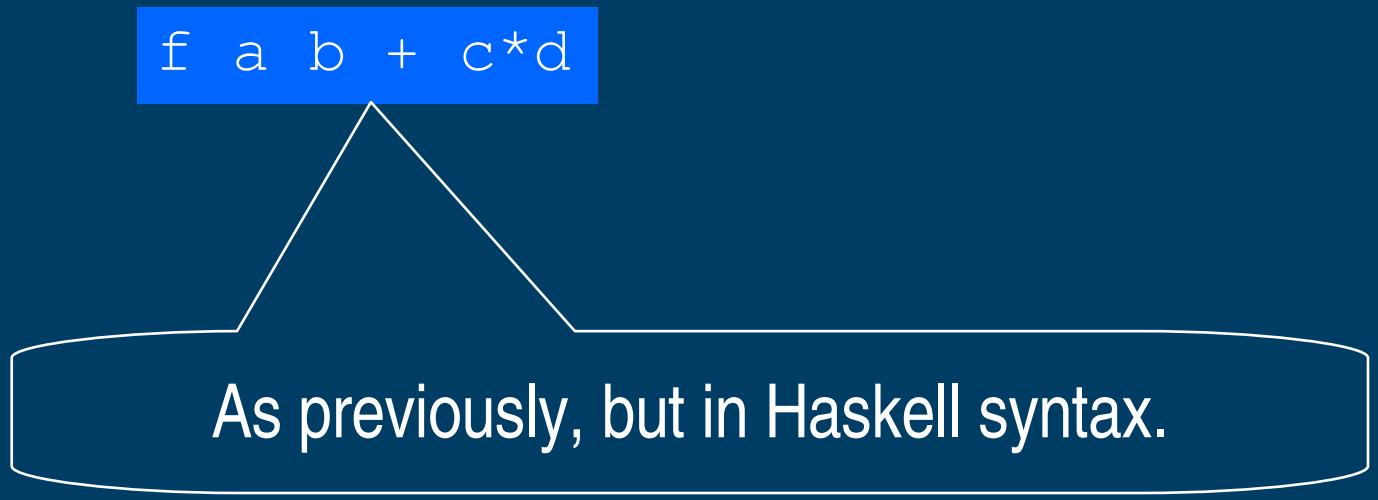
# *Function Application*

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 *.



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)

```
a + f b
```

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

Universiteit Antwerpen

# 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` |

Universiteit Antwerpen

# *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 Hugs.

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)
```

Universiteit Antwerpen

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

```
% hugs test.hs
```

Now both Prelude.hs and test.hs are loaded, and functions from both scripts can be used:

```
> quadruple 10
40

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

Universiteit Antwerpen

Leaving Hugs 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.

Universiteit Antwerpen

Hugs does not automatically reload scripts when they are 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..5]
3
```

Universiteit Antwerpen

# *Exercises*

(1) Try out some of the other functions from the standard prelude using Hugs.

(2) Work through "My First Script" using Hugs.

(3) Show how the functions last and init from the standard prelude could be re-defined using other functions from the prelude.

Note: there are many possible answers!

# *Haskell functions "interact"*

- Try this in Java, with two independent methods:

```
 -- repeat x is an infinite list,
 -- with x the value of every element.
repeat           :: a -> [a]
repeat x         =  xs where xs = x:xs

 -- replicate n x is a list of length n with x
 -- the value of every element
replicate        :: Int -> a -> [a]
replicate n x    =  take n (repeat x)
```

- Call to *repeat* would result in memory problems

*Source from http://www.haskell.org/onlinereport/standard-prelude.html*

Universiteit Antwerpen