

Introducing GHCi

Reading: Thompson Ch.2

COMP1100 — Introduction to Programming and Algorithms

Clem Baker-Finch

Australian National University

Semester 1, 2006

The expressions can be simple:

```
Prelude> 1+3-2
```

```
2
```

```
Prelude>
```

or not so simple:

```
Prelude> scanl (*) 1 [1..]
```

```
[1,1,2,6,24,120,720,5040,40320,362880,3628800, ...]
```

GHCi knows about *lots* of functions and operators that can be used in expressions. They are defined in the *Standard Prelude* which is just a Haskell module (`Prelude.hs`) containing a collection of definitions.

GHCi

GHC has an *interactive mode* that we will use a lot in this course.

Start it up by typing `ghci` in a terminal window.

Your computer responds with some messages as it starts up, ending with:

```
Prelude>
```

This is a **prompt**. You can type in an *expression* at the prompt.

GHCi will print its value and then prompt for another expression.

The 'language' or notation in which you write the expression is Haskell.

Example: Resting metabolic rates

Use GHCi to calculate your resting metabolic rate.

(How much energy you use being totally idle.)

From the *CSIRO Total Wellbeing Diet* book:

women: $(655.1 + 9.56 \times \text{weight} + 1.85 \times \text{height} - 4.68 \times \text{age}) \times 4.2$

men: $(66.47 + 13.75 \times \text{weight} + 5 \times \text{height} - 6.76 \times \text{age}) \times 4.2$

Weight in kg, height in cm, age in years. Result in kJ per day.

Haskell scripts

A GHCi session is like a calculator, but using it is not really *programming*.

Programming in Haskell centres around defining functions in a *script*.

[A *program* is a particular kind of script. We'll get to that later.]

A script is a file that contains *definitions*, *declarations* and *comments*.

By *loading* a script into GHCi, you can use the session to evaluate expressions containing functions and operators that are defined in that script, as well as those in the Prelude.

If we call the function `maleRestRate` we can write its *definition* in Haskell as an equation:

```
maleRestRate weight height age =  
    (66.47+13.75*weight+5*height-6.76*age)*4.2
```

Create a Haskell script containing this definition.

If we load the script into GHCi, we can evaluate expressions that **apply** the function `maleRestRate` to arguments:

```
*Main> maleRestRate 85 190 21  
8581.691
```

Resting metabolic rate, revisited...

Write a Haskell script with *functions* to calculate resting metabolic rates.

An algebraic expression with *unknowns* directly correspond to a function.

In the formula for males:

$$(66.47 + 13.75 \times \text{weight} + 5 \times \text{height} - 6.76 \times \text{age}) \times 4.2$$

`weight`, `height` and `age` represent **values** that we substitute into the formula to do the calculation.

That is, they are supplied as **arguments** to a *function* which calculates resting metabolic rate.

A note on the form of a Haskell function definition:

The right hand side of the definition is called the **body** and is just the formula with unknowns:

$$(66.47+13.75*\text{weight}+5*\text{height}-6.76*\text{age})*4.2$$

The left hand side is called the **head** and consists of the names of the **function** and its **arguments**:

```
maleRestRate weight height age
```

[You might have preferred `maleRestRate(weight,height,age)` but you'll get used to it ...]

Note:

The textbook refers to another Haskell system called Hugs.

It is almost identical to GHCi but does not include some features of Haskell that we will be using later in the course.