CA320 Computability and Complexity

Lab 1: Getting Started with Haskell

Aim

The aim of this practical is to show you how to use GHCi. This will involve becoming familiar with the GHCi environment, entering expressions into GHCi, loading scripts and creating your own scripts.

1. Starting GHCi

To start GHCi, simply type `ghci` at the Linux prompt.

2. A GHCi Session

You can now enter expressions at the GHCi prompt and GHCi will evaluate them.

You can also check the type of the expression by putting `:type` in front of it.

Try the following examples. Many of them use functions provided in the standard prelude. Are the answers what you expected?

```
3
3.0
'3'
"3"
3+4
20 `div` 5
20 `mod` 5
20 / 5
20.0 / 5.0
2 ^ 5
sqrt 2.0
sqrt (-1)
sqrt 5*5
sqrt (5*5)
sqrt 15.0 >= 3.5
sin (pi/2.0)
6 < 4
3>4 && 4>3
3>4 || 4>3
not (3>4 && 4>3)
"Hello World!\n"
putStr "Hello World!\n"
"abc" ++ "def"
length "This sentence no verb."
words "The quick brown fox jumped over the lazy
dog"
```

```
"The length of \"this sentence\" is :" ++
     show (length "this sentence")
```

Try to calculate the following:

(a) Multiply 12345 by 67890

(b) Add 2468 and 13579

(c) Multiply the first 5 numbers (i.e. 1 times 2 times 3 times 4 times 5)

(d) Multiply the first 30 numbers

(e) Add the numbers from 1 to 100

(f) Add the odd numbers from 1 to 99

(g) What is the remainder when 456 is divided by 10?

(h) What is the remainder when 365 is divided by 7?

(i) Evaluate:

where a=2, b=8 and c=6.

3. Loading a Haskell Script

Download the file collatz.hs. Don't worry – you are not expected to be able to fully understand this program at this stage.

To try out this script, it must be loaded into GHCi, so start a GHCi session if you don't already have one running. Now you can load the script into GHCi by typing `:load collatz` at the GHCi prompt.

The function `collatzSeq` in the script takes a positive integer argument and returns as result the Collatz sequence for that number. Try the following and any other input values you like:

```
collatzSeq 1
collatzSeq 3
collatzSeq 7
```

The function `maxSeq` in the script takes a positive integer argument and returns as result the length of the longest Collatz sequence up to that number. Try the following and any other input values you like:

```
maxSeq 10
maxSeq 100
maxSeq 1000
```

4. Creating a Haskell Script

To create your own script, open the editor and type in the following:

```
-- This Haskell program defines functions for a
sphere
-- function square returns the square of the
input value
square :: Float -> Float
square x = x * x
```

```
-- function cube returns the cube of the input value

cube :: Float -> Float

cube x = x * x * x

-- function volume returns the volume of the sphere with the given radius

volume :: Float -> Float

volume r = (4.0 * pi * cube r)/3.0

-- function surfaceArea returns the surface area of the sphere with the given radius

surfaceArea :: Float -> Float

surfaceArea r = 4.0 * pi * square r
```

Save the file with an appropriate name e.g. exercise1.hs. Don't forget that the suffix must be `.hs` or GHCi won't know that it is a Haskell script. To try out the script that you have just created, it must be loaded into GHCi, so start a GHCi session if you don't already have one running. Now you can load your script as described in the previous exercise. If you haven't made any typing errors, GHCi should successfully load your script. If you have made some typing mistakes, GHCi will give you an error message while attempting to load the script. The error message will include the line number at which GHCi could not continue. An easy way to go to that line in an editor is to type `:edit` at the GHCi prompt. When you save your corrections, your script will automatically be re-loaded into GHCi. Note that your error will often *precede* the line reported in the error message. Alternatively, if you already have an editor running, you can make corrections there, and then tell GHCi to re-load by typing the command `:reload`, or just `:r`. When you have successfully loaded your script you should experiment by getting GHCi to evaluate a few expressions using the definitions in your script.

Note: The above lab was originally created as part of Geoff Hamilton's CA320 course.