Getting started on the Haskell HUGS interpreter

The HUGS Haskell interpreter works similarly to the GAP interpreter. When you start it up, you will see a prompt that looks like:

Hugs>

As with GAP, you can sit and type commands at the prompt, but for most purposes, it is best to edit your program file on an editor and read it into the interpreter. If your file is named program.hs, enter

Hugs> `:load program.hs`

to instruct the interpreter to read the file. The `:load` can be abbreviated as `:l`, so you only need to enter `:l program.hs` . If you make changes and want to reload an edited version of program.hs, all you need to enter is `:r` (which is short for `:reload`) and it will load whatever version of program.hs is currently written on the hard drive.

To quit, enter `:q`, and for a complete list of the interpreter commands enter `:?` .

Now, at the prompt, try entering the following:

Hugs> `5*4 + 3`

Hugs> `5*(4+3)`

Hugs> `2^5000`

Hugs> `sqrt 2`

Haskell understands ordered pairs. When you enter

Hugs> `(5, 3)`

Haskell's response is just to print out $(5, 3)$. Why? Because like GAP, Haskell is an *evaluator*. When you give it $(5, 3)$, it evaluates it and finds that the value is $(5, 3)$.

If `f` is a function and `x` is a variable, then `f x` is Haskell for `f(x)`. Haskell will correctly parse `f(x)`, but the convention is not to use parentheses unless they are necessary. Try these two useful functions:

Hugs> `fst (5,3)`

Hugs> `snd (5,3)`

Now enter

Hugs> `fst (5,3,2)`

You should get the following error message:

```
ERROR - Type error in application
*** Expression :  fst (5,3,2)
*** Term :  (5,3,2)
*** Type :  (c,d,e)
*** Does not match :  (a,b)
```

Haskell is very strongly typed, and has detected a type error. Let's go through the parts of the message. First it tells the entire expression that caused the error, then the particular term in it, $(5, 3, 2)$. It then gives the type of $(5, 3, 2)$ as being `(c,d,e)`, i. e. a triple of any kind of entries, and finally says it does not match the expected type `(a,b)` of an ordered pair. The `fst` function only accepts ordered pairs, although their *entries* may be of any type. Try entering the following to see how "polymorphic" (accepts different types in inputs) the `fst` function is:

Hugs> `fst (5,(5,3,2))`

Hugs> `fst ((5,3,2),(2,4,8))`

Hugs> `fst ("A character string", (2,4,8))`

Now, let's ask Haskell the type of `fst`. Enter

Hugs> `:t fst`

Haskell evaluates this to

```
fst ::  (a,b) -> a
```

The symbol `::` is read as "is of type". The "arrow" `->` has the usual function meaning that it does is mathematics. This line tells us that `fst` is of type a function that takes an element of the form `(a,b)`, that is, an ordered pair where the first and second elements have arbitrary types, and not necessarily of the same type, and returns an element of the same type as the first entry of the ordered pair.

Now, let's look at compositions of functions. Try

Hugs> `snd fst ((1,2),(3,4))`

We might expect this to evaluate to 2, but instead we get

```
ERROR - Type error in application
*** Expression :  snd fst ((1,2),(3,4))
*** Term :  fst
*** Type :  (g,h) -> g
*** Does not match :  (a,((b,c),(d,e)) -> f)
```

This show that Haskell looked at `snd` and was expecting it to be followed by an ordered pair, but instead got `fst`. The type that was encountered, that is the type of `fst`, was `(g,h) -> g`. The expected type, `(a,((b,c),(d,e)) -> f)`, is an ordered pair. Rather than describing the expected type as `(a,b)`, it gives the second entry as `((b,c),(d,e)) -> f`, that is, a function that takes ordered pairs of ordered pairs and returns an object of a possibly different type. I don't know why it is given this way.

Let's fix the error. We already know that Haskell understands grouping with parentheses, so let's try

Hugs> `snd (fst ((1,2),(3,4)))`

This works. But a more "functional" solution would be to use a composition of `snd` and `fst`. Haskell's composition symbol is the period, so if we enter

```
Hugs> ( snd . fst) ((1,2),(3,4))
```

we also get the correct evaluation.

Check the type of `snd . fst` by entering

```
Hugs> snd . fst
```

The evaluation is

```
snd . fst ::  ((a,b),c) -> b
```

which is exactly correct.