Learning Haskell

Personal notes to learning Haskell

by

Johnson Ng

Table of Contents

1 Getting Started

1.1 Installation

For Arch-based distributions, and who prefers to manage their packages[1], see the Arch Wiki.

For most systems, see the official download site for an installation guide.

[1] Not necessarily for Haskell packages, but just for `cabal` or `stack`, while managing Haskell packages using either of these two.

1.2 Introduction via `ghci`

Launch `ghci`[2] via the terminal. You should see something along the lines of the following:

[2] `ghci` stands for "GHC Interactive"

```
GHCi, version 6.8.2: http://www.haskell.org/ghc/ :? for help
Loading package base ... linking ... done.
Prelude>
```

The prompt is `Prelude>`, but since it gets longer as stuff are loaded, we shall simplify that and use `ghci>`. You can achieve the same in your prompt by running `:set prompt "ghci>"`.

`ghci` comes with some of the familiar cli editing, including the **up** arrow key to get the last line of input, and **tab completion**.

1.2.1 Basic Arithmetic

Basic arithmetic is similar to most other languages. We write them in **infix form**, where the operator appears between its operands.

```
ghci> 3 + 7
```

```
10
ghci> 49 * 23
1127
ghci> 1923 - 333
1590
ghci> 5 / 2
2.5
```

Since we explicitly stated that, it means that there is also another way of writing these expressions. In particular, we can write them in **prefix form**.

```
ghci> (+) 3 7
10
ghci> (*) 49 23
1127
ghci> (-) 1923 333
1590
ghci> (/) 5 2
2.5
```

Warning (Negative Numbers)

It is often necessary to enclose a negative number in parenthesis, just like we do in mathematics. In Haskell, - is a unary operator, and is in fact its only unary operator, and it cannot be mixed with infix operators.

```
ghci> 2 + -3

<interactive>:1:0: error:
    Precedence parsing error
        cannot mix ‘+’ [infixl 6] and prefix ‘-’ [infixl 6]
            in the same infix expression
```

A similar problematic example is the following, despite having a different error message:

```
ghci> 2*-3

<interactive>:1:1: error:
    Variable not in scope: (*-) :: Integer -> Integer -> t
```

1.2.2 *Boolean logic, operators, and value comparisons*

The values of **Boolean logic** in Haskell are `True` and `False`. Capitalization of the names matter.

There is also

- `(&&)` as logical "and"; and
- `(||)` as logical "or".

```
ghci> True && False
False
ghci> False || True
True
```

Warning

While some programming languages treat the number zero as False, this is not the case in Haskell, nor does Haskell consider non-zero values to be `True`

```
ghci> False && 0

<interactive>:18:10: error:
• No instance for (Num Bool) arising from the literal ‘0’
• In the second argument of ‘(&&)’, namely ‘0’
  In the expression: False && 0
  In an equation for ‘it’: it = False && 0
```

Here's a breakdown of the error message:

- "`No instance for (Num Bool) arising from the literal ‘0’`" *tells us that* `ghci` *tried to treat the numeric value* `0` *as a* `Bool` *type, but it failed to do so.*
- "`In an equation for ‘it’: it = False && 0`" *refers to a shortcut in* `ghci` *that we shall visit later.*

The **comparison operators** in Haskell are similar to those in C and many other languages.

```
ghci> 1 == 1
True
ghci> 2 < 3
True
ghci 4 >= 3.99
True
```

The "is not equal to" operator in Haskell is `(/=)`.

```
ghci> 2 /= 3
True
```

The **logical negation** for Haskell is `not`.

```
ghci> not True
False
```

1.2.3 *Operator precedence and associativity*

Like in written algebra and other languages that use infix operators, Haskell has operator precedence. We can use parenthesis to explicitly group parts of an expression, and precedence allows us to omit a few parenthesis. For example, we know that multiplication precedes addition, and in Haskell:

```
ghci> 1 + ( 4 * 4 )
17
ghci> 1 + 4 * 4
17
```

In Haskell, operators are assigned numeric precedence values, with 1 being the lowest and 9 the highest. An operator with a higher precedence is applied before one that has a lower precedence. We can use `ghci` to inspect the precedence levels of individual operators using the `:info` command.

```
ghci> :info (+)
class Num a where
  (+) :: a -> a -> a
  ...
        -- Defined in ‘GHC.Num’
infixl 6 +
ghci> :info (*)
class Num a where
  ...
  (*) :: a -> a -> a
  ...
        -- Defined in ‘GHC.Num’
infixl 7 *
```

The information we want here is in the line "`infixl 6 +`", which says that the `(+)` operator has precedence 6. Similarly, the `(*)` operator has precedence 7.

Haskell also defines **associativity** of operators[3]. The `(+)` and `(*)` are left associative, which is shown as `infixl` in the `ghci` output above. The right associative operator is displayed with `infixr`. An example of a right associative operator is

[3] See `https://en.wikipedia.org/wiki/Operator_associativity`

```
ghci> :info (^)
 (^) :: (Num a, Integral b) => a -> b -> a   -- Defined in
     ‘GHC.Real’
 infixr 8 ^
```

Left associativity means that `1 + 2 + 3` is interpreted as `(1 + 2) + 3`, while right associativity means that `1^2^3` is interpreted as `1^(2^3)`.

The combination of precedence and associativity rules are usually referred to as **fixity** rules.

Note

It is sometimes better to leave at least some parentheses in place, even when you are sure that Haskell will parse your expressions correctly by fixity rules. This helps future readers to read and understand the code better.

Bibliography

Lipovača, M. Learn you a haskell. `http://learnyouahaskell.com/chapters`.

O'Sullivan, B., Stewart, D., and Goerzen, J. (2007). Real world haskell. `http://book.realworldhaskell.org/read/`.

Index