Programming Languages: Functional Programming
Worksheet for 2. Introduction to Haskell

Shin-Cheng Mu

Spring 2022

If you have your notebook computer with you (and have Haskell Platform installed), start `ghci` and try the following tasks.

List Deconstruction

1. (a) What is the type of the function $head$? Use the command `:t` to find out the type of a value.

   (b) Since the input type of $head$ is a list ($[a]$), let us try it on some input.

      i. $head\ [1,2,3] =$
      ii. $head$ "abcde" $=$
      iii. $head\ [\,] =$

   (c) In words, what does the function $head$ do?

2. (a) What is the type of the function $tail$?

   (b) Try $tail$ on some input.

      i. $tail\ [1,2,3] =$
      ii. $tail$ "abcde" $=$
      iii. $tail\ [\,] =$

   (c) In words, what does the function $tail$ do?

   (d) For what xs is it always true that $head\ xs : tail\ xs = xs$?

3. (a) What is the type of the function $last$?

   (b) Try $last$ on some input. Think about some input yourself.

      i. $last \qquad =$
      ii. $last \qquad =$
      iii. $last \qquad =$

   (c) In words, what does the function $last$ do?

4. (a) What is the type of the function $init$?

   (b) Try $init$ on some input. Think about some input yourself.

      i. $init \qquad =$
      ii. $init \qquad =$
      iii. $init \qquad =$

   (c) In words, what does the function $init$ do?

   (d) What property does $init$ and $last$ jointly satisfy?

5. (a) What is the type of the function $null$?

   (b) Try $init$ on some input. Think about some input yourself.

      i. $null \qquad =$
      ii. $null \qquad =$
      iii. $null \qquad =$

   (c) Can you write down a definition of $null$, by pattern matching?

List Generation

1. What are the results of the following expressions?

(a) $[0..25]\ =$

(b) $[0, 2..25]\ =$

(c) $[25..0]\ =$

(d) $['a'..'z']\ =$

(e) $[1..]\ =$

2. What are the results of the following expressions?

   (a) $[x \mid x \leftarrow [1..10]]\ =$

   (b) $[x \times x \mid x \leftarrow [1..10]]\ =$

   (c) $[(x, y) \mid x \leftarrow [0..2], y \leftarrow \texttt{"abc"}]\ =$

   (d) What is the type of the expression above?

   (e) $[x \times x \mid x \leftarrow [1..10], odd\ x]\ =$

3. What are the results of the following expressions?

   (a) $[(a, b) \mid a \leftarrow [1..3], b \leftarrow [1..2]] =$

   (b) $[(a, b) \mid b \leftarrow [1..2], a \leftarrow [1..3]] =$

   (c) $[(i, j) \mid i \leftarrow [1..4], j \leftarrow [(i + 1)..4]] =$

(d) $[(i, j) \mid i \leftarrow [1..4], \mathit{even}\ i,\ j \leftarrow [(i+1)..4], \mathit{odd}\ j] =$

(e) $['a'|i \leftarrow [0..10]]\ =$

Combinators on Lists

1. (a) What is the type of the function !! (two exclamation marks)?

   (b) Try !! on some input. Think about some input yourself. Note that !! is an infix operator.

      i. $[1, 2, 3]\ !!\ 1\ =$

      ii. $\qquad !! \quad =$

      iii. $\qquad !! \quad =$

   (c) In words, what does the function !! do?

2. (a) What is the type of the function length?

   (b) Try length on some input.

      i. $\mathit{length} \qquad =$

      ii. $\mathit{length} \qquad =$

   (c) In words, what does the function length do?

3. (a) What is the type of the function $(++)$? (In ASCII one types ++.)

   (b) Try $(++)$ on some input. Think about some input yourself. Note that $(++)$ is an infix operator.

      i.

      ii.

   (c) In words, what does the function $(++)$ do?

(d) Wait a minute… Both $(:)$ and $(+\!\!+)$ appear to add elements to a list. How are they different?

4. (a) What is the type of the function $concat$?

   (b) Try $concat$ on some input.

      i. $concat \qquad\qquad =$

      ii. $concat \qquad\qquad =$

   (c) In words, what does the function $concat$ do?

5. (a) What is the type of the function $take$?

   (b) Try $take$ on some input. Since $take$ expects an integer and list, try it on some extreme cases. For example, when the integer is zero, negative, or larger than the length of the list.

      i. $take \qquad\qquad =$

      ii. $take \qquad\qquad =$

      iii. $take \qquad\qquad =$

   (c) In words, what does the function $take$ do?

6. (a) What is the type of the function $drop$?

   (b) Try $drop$ on some input. Like $take$, try it on some extreme cases.

      i. $drop \qquad\qquad =$

      ii. $drop \qquad\qquad =$

      iii. $drop \qquad\qquad =$

   (c) In words, what does the function $drop$ do?

   (d) Does $take$, $drop$, and $(+\!\!+)$ together satisfy some properties?

7. (a) What is the type of the function map?

   (b) Try map on some input. It is a little bit harder, since map expects a functional argument.

      i. $map\ square\ [1,2,3,4]\ =$
      ii. $map\ (1+)\ [1,2,3,4]\ =$
      iii. $map\ (const\ 'a')\ [1..10]\ =$

   (c) In words, what does the function map do?

   (d) Is $(1+)$ a function? Try it.

      i. $(1+)\ 2\ =$
      ii. $((1+) \cdot (1+) \cdot (1+))\ 0\ =$
         where $(\cdot)$ is function composition.

Sectioning

- Infix operators are *curried* too. The operator $(+)$ may have type $Int \to Int \to Int$.
- Infix operator can be partially applied too.

$$\begin{aligned}(x \oplus)\ y &= x \oplus y \\ (\oplus\ y)\ x &= x \oplus y\end{aligned}$$

  - $(1\ +) :: Int \to Int$ increments its argument by one.
  - $(1.0\ /) :: Float \to Float$ is the "reciprocal" function.
  - $(/\ 2.0) :: Float \to Float$ is the "halving" function.

1. Define a function $doubleAll :: List\ Int \to List\ Int$ that doubles each number of the input list. E.g.

   - $doubleAll\ [1,2,3] = [2,4,6]$.

   - How do you define a new function? I'd suggest you to

     (a) create a new text file (using your favourite editor) in your current working directory (the directory you executed `ghci`). The file should have extension `.hs`.
     (b) Type your definitions in the file.
     (c) Load the file into `ghci` by the command `:l <filename>`.

2. Define a function $quadAll :: List\ Int \to List\ Int$ that multiplies each number of the input list by 4. Of course, it's cool only if you define $quadAll$ using $doubleAll$.

λ Abstraction

- Every once in a while you may need a small function which you do not want to give a name to. At such moments you can use the λ notation:
  - $map\ (\lambda x \to x \times x)\ [1, 2, 3, 4] = [1, 4, 9, 16]$
  - In ASCII λ is written $\backslash$.

1. What is the type of $(\lambda x \to x + 1)$?
2. $(\lambda x \to x + 1)\ 2\ =$
3. What is the type of $(\lambda x \to \lambda y \to x + 2 \times y)$?
4. What is the type of $(\lambda x \to \lambda y \to x + 2 \times y)\ 1$?
5. $(\lambda x \to \lambda y \to x + 2 \times y)\ 1\ 2\ =$
6. What is the type of $(\lambda x\ y \to x + 2 \times y)$?
7. What is the type of $(\lambda x\ y \to x + 2 \times y)\ 1$?
8. $(\lambda x\ y \to x + 2 \times y)\ 1\ 2\ =$
9. Define $doubleAll :: List\ Int \to List\ Int$ again. This time using a λ expression.

10. **Pattern matching in** λ. To extract, for example, the two components of a pair
    (a) What is the type of $(\lambda(x, y) \to (y, x))$?

    (b) $(\lambda(x, y) \to (y, x))\ (1, \text{'}a\text{'})\ =$
    (c) Alternatively, try
    $(\lambda p \to (snd\ p, fst\ p))\ (1, \text{'}a\text{'})\ =$

Back to Lists

1. (a) What is the type of the function filter?

   (b) Try filter on some input.

      i. $\mathit{filter}\ \mathit{even}\ [1..10]\ =$

      ii. $\mathit{filter}\ (>10)\ [1..20]\ =$

      iii. $\mathit{filter}\ (\lambda x \to x\ \text{`}\mathit{mod}\text{`}\ 3 == 1)\ [1..20]\ =$

   (c) In words, what does the function filter do?

2. (a) What is the type of the function $\mathit{takeWhile}$?

   (b) Try $\mathit{takeWhile}$ on some input.

      i. $\mathit{takeWhile}\ \mathit{even}\ [1..10]\ =$

      ii. $\mathit{takeWhile}\ (<10)\ [1..20]\ =$

      iii. $\mathit{takeWhile}\ (\lambda x \to x\ \text{`}\mathit{mod}\text{`}\ 3 == 1)\ [1..20]\ =$

   (c) In words, what does the function $\mathit{takeWhile}$ do? How does it differ from filter?

   (d) Define a function $\mathit{squaresUpto} :: \mathit{Int} \to \mathit{List}\ \mathit{Int}$ such that $\mathit{squaresUpto}\ n$ is the list of all positive square numbers that are at most n. For some examples,

      - $\mathit{squaresUpto}\ 10 = [1, 4, 9]$.
      - $\mathit{squaresUpto}\ (-1) = [\,]$

3. (a) What is the type of the function $\mathit{dropWhile}$?

   (b) Try $\mathit{dropWhile}$ on some input.

      i. $\mathit{dropWhile}\ \mathit{even}\ [1..10]\ =$

      ii. $\mathit{dropWhile}\ (<10)\ [1..20]\ =$

      iii. $\mathit{dropWhile}\ (\lambda x \to x\ \text{`}\mathit{mod}\text{`}\ 3 == 1)\ [1..20]\ =$

(c) In words, what does the function $dropWhile$ do?

4. (a) What is the type of the function zip?

   (b) Try zip on some input.

      i. zip $[1..10]$ "abcde" $=$
      ii. zip "abcde" $[0..]$ $=$
      iii. zip $=$

   (c) In words, what does the function zip do?

   (d) Define $positions :: Char \to String \to List\ Int$, such that $positions\ x\ xs$ returns the positions of occurrences of x in xs. E.g.

      - $positions$ 'o' "roodo" $= [1, 2, 4]$.

      Check the handouts if you just cannot figure out how.

   (e) What if you want only the position of the *first* occurrence of x? Define $pos :: Char \to String \to Int$, by reusing $positions$.

Morals of the Story

- Lazy evaluation helps to improve modularity.
  - List combinators can be conveniently re-used. Only the relevant parts are computed.
- The combinator style encourages "wholemeal programming".
  - Think of aggregate data as a whole, and process them as a whole!