Programming Languages: Functional Programming Practicals 1: Functions and Definitions

Shin-Cheng Mu

Sep. 07, 2023

You should have installed GHC, with its commandline interface GHCi. Open your favourite text editor, create a new plain text file. The filename extension must end in `.hs`. This will be your working file for this practical. Type `ghci <filename>.hs` in the command line to load the working file into GHCi.

1. Define a function $\mathit{myeven} :: \mathsf{Int} \to \mathsf{Bool}$ that determines whether the input is an even number. You may use the following functions:

   $$\begin{aligned} &\mathit{mod} :: \mathsf{Int} \to \mathsf{Int} \to \mathsf{Int} \enspace , \\ &(==) \;\; :: \mathsf{Int} \to \mathsf{Int} \to \mathsf{Bool} \enspace . \end{aligned}$$

   (Types of the functions written above are not in their most general form.)

2. Define a function that computes the area of a circle with given radius r (using $22 / 7$ as an approximation to π). The return type of the function might be Float.

3. Part-time students in Institute of Information Science are paid NTD 130 per hour. Define a function $\mathit{payment} :: \mathsf{Int} \to \mathsf{Int}$ that, when applied to the numbers of weeks a student work, compute the amount of money the Institute has to pay the student.

   (a) Assume that there are five working days in a week, eight working hours per day. Define *payment*. For clarity, use **let** to define local variables recording number of days worked, number of hours worked, etc.

   (b) Define *payment* again, but declare the local variables using **where**. Which style do you prefer?

   (c) The regulation states that students are considered workers, and if a worker works for more than 19 weeks, the Institute has to pay, in addition to the salary, health insurance and pension reserves for the worker. The amount is 6% of the worker's salary.

   Update definition of *payment* in the form:

   $$\begin{aligned} &\mathit{payment} :: \mathsf{Int} \to \mathsf{Int} \\ &\mathit{payment}\ \mathit{weeks} \mid \mathit{weeks} > 19 = \ldots \\ &\phantom{\mathit{payment}\ \mathit{weeks}} \mid \mathit{otherwise} \;\; = \ldots \end{aligned}$$

You may need a function *fromIntegral* to convert Int to Float, and a function *round* that rounds a floating point number to the nearest integer.

In this case, should you use **let** or **where**?

4. More on **let**.

   (a) Guess what the value of *nested* would be. Type it into your working file and evaluated in in GHCi to see whether you guessed right. Note that indentation matters.

```
nested :: Int
nested = let x = 3
         in (let x = 5
             in x + x) + x .
```

   (b) Guess what the value of *recursive* would be. Try it in GHCi.

```
recursive :: Int
recursive = let x = 3
            in let x = x + 1
               in x .
```

5. Type in the definition of *smaller* into your working file.

```
smaller :: Int → Int → Int
smaller x y = if x ⩽ y then x else y .
```

   Then try the following:

   (a) In GHCi, type `:t smaller` to see the type of *smaller*.

   (b) Try applying it to some arguments, e.g. *smaller* 3 4, *smaller* 3 1.

   (c) Use `:t` to see the type of *smaller* 3 4.

   (d) Use `:t` to see the type of *smaller* 3.

   (e) In your working file, define a new function *st3* = *smaller* 3.

   (f) Find out the type of *st3* in GHCi. Try *st3* 4, *st3* 1. Explain the results you see.

6. More practice on curried functions.

   (a) Define a function *poly* such that $poly\ a\ b\ c\ x = a \times x^2 + b \times x + c$. All the inputs and the result are of type *Float*.

   (b) Reuse *poly* to define a function *poly1* such that $poly1\ x = x^2 + 2 \times x + 1$.

   (c) Reuse *poly* to define a function *poly2* such that $poly2\ a\ b\ c = a \times 2^2 + b \times 2 + c$.

7. Type in the definition of *square* in your working file.

   (a) Define a function *quad* :: Int → Int such that *quad x* computes x^4.

(b) Type in this definition into your working file. Describe, in words, what this function does.

$$\begin{aligned} &\mathit{twice} && :: (a \to a) \to (a \to a) \\ &\mathit{twice}\ f\ x && = f\ (f\ x) \ . \end{aligned}$$

(c) Define *quad* using *twice*.

8. Replace the previous *twice* with this definition:

$$\begin{aligned} &\mathit{twice} && :: (a \to a) \to (a \to a) \\ &\mathit{twice}\ f && = f \cdot f \ . \end{aligned}$$

(a) Does *quad* still behave the same?

(b) Explain in words what this operator $(\cdot)$ does.

9. Functions as arguments, and a quick practice on sectioning.

(a) Type in the following definition to your working file, without giving the type.

$$\mathit{forktimes}\ f\ g\ x = f\ x \times g\ x \ .$$

Use $:t$ in GHCi to find out the type of *forktimes*. You will end up getting a complex type which, for now, can be seen as equivalent to

$$(t \to \mathit{Int}) \to (t \to \mathit{Int}) \to t \to \mathit{Int} \ .$$

Can you explain this type?

(b) Define a function that, given input x, use *forktimes* to compute $x^2 + 3 \times x + 2$. **Hint**: $x^2 + 3 \times x + 2 = (x + 1) \times (x + 2)$.

(c) Type in the following definition into your working file: $\mathit{lift2}\ h\ f\ g\ x = h\ (f\ x)\ (g\ x)$. Find out the type of *lift2*. Can you explain its type?

(d) Use *lift2* to compute $x^2 + 3 \times x + 2$.

10. Let the following identifiers have type:

$$\begin{aligned} f &:: \mathit{Int} \to \mathit{Char} \\ g &:: \mathit{Int} \to \mathit{Char} \to \mathit{Int} \\ h &:: (\mathit{Char} \to \mathit{Int}) \to \mathit{Int} \to \mathit{Int} \\ x &:: \mathit{Int} \\ y &:: \mathit{Int} \\ c &:: \mathit{Char} \end{aligned}$$

Which of the following expressions are type correct?

1. $(g \cdot f)\ x\ c$

2. $(g\ x \cdot f)\ y$
3. $(h \cdot g)\ x\ y$
4. $(h \cdot g\ x)\ c$
5. $h \cdot g\ x\ c$

You may type the expressions into Haskell and see whether they type check. To define f, for example, include the following in your working file:

$$f :: Int \rightarrow Char$$
$$f = undefined$$

However, it is better if you can explain why the answers are as they are.