Lab 1

Functional Programming (ITI0212)

2023-02-03

This course is about dependently typed functional programming. This is a vibrant subject, some innovations from which have already made their way from academic research to mainstream software engineering, with others still on the way. Think of this course as a sneak-peek of the future. `;-)`

As the language for this course we will be using Idris (language version 2, release 0.6).

Task 1 (install Idris)
If you haven't already done so, install Idris on your computer by following the instructions at `https://compose.ioc.ee/courses/2023/functional_programming/reference/install_idris.html`.

Once Idris is installed you should be able to run the interactive interpreter or REPL using the command `idris2`, and see the following:

```
     ____    __     _         ___
    /  _/___/ /____(_)____   |__ \
    / // __  / ___/ / ___/   __/ /     Version 0.6.0
  _/ // /_/ / /  / (__  )   / __/      https://www.idris-lang.org
 /___/\__,_/_/  /_/____/   /____/      Type :? for help

Welcome to Idris 2.  Enjoy yourself!
Main>
```

You can quit the REPL by typing `:quit` (or its abbreviation, `:q`).

Task 2 (install interactive editor integration)
After installing Idris you should install an interactive editing integration for your program editor of choice as described in the installation instructions linked above.

Task 3
The general workflow for programming in Idris is to have an editor open with your program source and a REPL session in a console where you can test things out.

- In the directory where you plan to store your lab exercises, create a new empty file called `Lab1.idr`.
- In a terminal shell, change to this directory and load the file in the Idris REPL with the command `idris2 Lab1.idr` (or `rlwrap idris2 Lab1.idr` to enable command-line history).
- Open this file in your program editor and enter the following lines:

  ```
  module Lab1

  small : Integer
  small = 4
  ```

```
large  :  Integer
large  =  small * 6
```

Then ask the Idris interactive mode to save and type check your file. The key-chord for this depends on your operating system and editor. Consult the documentation for the specific interactive mode that you chose above.

- In the REPL, ask Idris to reload the file with the command `:reload` (or its abbreviation, `:r`). Idris should respond that the file was loaded successfully. Now type `large` at the prompt and press enter. You should see the value assigned to that variable displayed.
- Still in the REPL, ask Idris to tell you the type of the variable `large` with the command `:type large` (or its abbreviation, `:t large`). Idris should respond that `large` is an `Integer`.

Task 4
Now you will interactively write a function that computes the average of two `Integer`s (truncating any halves).

- Type the following line into your program:

  ```
  average  :  Integer -> Integer -> Integer
  ```

- Place the cursor somewhere over the function name and ask Idris to automatically add a definition clause. The key-chord for this depends on your operating system and editor, usually it is some command-key combination together with "a" for "add definition". The following line should be automatically added to your file:

  ```
  average x y  =  ?average_rhs
  ```

- Place the cursor somewhere over the goal `?average_rhs` and ask Idris to inspect this goal. The key-chord for this is usually some command-key combination together with "t" for "type-in-context". Idris should tell you that the goal must be an `Integer`, and that there are two `Integer` bound variables in scope.
- Complete the definition of the `average` function. If you don't know the name of a function that you think should be in the standard library you can search for it by its type in the REPL using `:search <type_of_function>`. When you are done, reload it in the editor to make sure there are no errors, then reload the file in the REPL and test your function on some inputs.

Your function should behave as follows:

```
Lab1> average 1 9
5
Lab1> average 1 10
5
```

Task 5
Back in your program file, declare an `Integer` variable called `medium`, and assign to it the average of `small` and `large`, as determined by the function you just wrote.

Task 6
Write a function `average'` that returns the average of two `Double`s as a `Double`.

Your function should behave as follows:

```
Lab1> average' 1 9
5.0
```

```
Lab1> average' 1 10
5.5
```

Hint: try using the following workflow:

- start by entering the type signature for the function,
- use interactive editing to add a clause,
- use interactive editing to inspect the goal and think about how to get what you want from what you have,
- if you want to use a function that you think should be in the standard library but don't know its name, try searching for it by its type.