Lab 1

Functional Programming

2024-01-26

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 0.7).

Task 1 (install Idris)
The easiest way to install Idris is using the homebrew package manager (`https://brew.sh`).

On MacOS first make sure that you have Apple's developer tools installed:

- `xcode-select --install`

then install homebrew and run:

- `brew install idris2`
- `brew install rlwrap` (for command history – recommended)
- `brew install visual-studio-code` (to use this program editor – recommended)

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

```
     ____     __      _          ___
    /  _/___/ /____(_)____    |__ \
    / // __  / ___/ / ___/    __/ /      Version 0.7.0
  _/ // /_/ / /  / (__  )    / __/       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`).

If your shell reports that it can't find `idris2`, then make sure that it is installed and located in your search path (`echo $PATH`).

Task 2 (install interactive editor integration)
After installing Idris, install an Idris integration for your program editor of choice.
For `Visual Studio Code` (recommended) use the `idris-vscode` extension by meraymond. After installing the editor integration, tell it the path to your `idris2` executable (`Settings -> Extensions -> Idris`) and open the *keyboard shortcuts* (`Keyboard Shortcuts -> Idris`).

Task 3
The general workflow for programming in Idris is to have your source file open in your program editor and a REPL session running in a terminal so you can test things out interactively.

- Create a new empty file called `Lab1.idr`, open it in your program editor, and enter the following lines:

```
module Lab1

small  :  Integer
small  =  4

large  :  Integer
large  =  small * 6
```

  Then save your file. This should trigger the Idris editor integration to read in your file and highlight any errors.

- If VSCode reports an error saying that *Multiple workspaces are not currently supported*, then close the file, open the folder containing it as an "implicit workspace" (`File -> Open Folder...`) and re-open the file from there.
- In a terminal shell, change to the directory containing your lab file and load the file in the Idris REPL using the command `idris2 Lab1.idr` (or `rlwrap idris2 Lab1.idr` to enable command history).
- 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 (mean) of two `Integer`s (truncating any halves).

- Type the following line into your program file:

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

- Place the cursor somewhere over the function name and ask Idris to add a definition clause for you. Use the key-chord that you set for the command `idris: Add Clause`. 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. Use the key-chord that you set for the command `idris: Type At`. 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 type in the REPL using `:search <type_of_function>`. When you are done, save your file to let Idris check it for 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`, using the function that you just wrote.

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

To learn what a `Double` is, ask Idris using `:doc Double` (just ignore the `Hints` for now).

Your function should behave as follows:

```
Lab1> average' 1 9
5.0
Lab1> average' 1 10
5.5
```

Notice that because Idris knows the *type* of the function, it is able to correctly parse the numeric literal arguments as `Double`s.

Hint: try using the following workflow:

- start by giving the *type specification* for the function,
- use interactive editing to add a *definition 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, *search* for it by type.

Task 7 (clone course assignments repository)

- If you have not already done so, create a GitHub account and associate your Bowdoin email address with it.
- (optional, but strongly recommended) Configure your GitHub account with SSH access: `https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account`
- Sign into GitHub Classrooms (`https://classroom.github.com`).
- Accept the invitation to join the classroom `bowdoin-csci-2520` and clone the starter assignments repository (link in email) to your computer.

Task 8 (push your work to your assignments repository)
Copy your `Lab1.idr` file to the `lab` directory of your course assignments repository, then commit your changes and push your work to GitHub.