Lab 1

Functional Programming (ITI0212)

2021.01.26

This course is about 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 (`https://www.idris-lang.org/`), version 2.

Installing Idris 2

Idris 2 is a re-implementation of the Idris programming language in Idris itself, together with a new, much faster runtime system based on Chez Scheme. Unfortunately, Idris 2 has not yet reached 1.0 release status and binary installers are not yet available, so you will need to build it from source. Fortunately, this is a fairly easy process on most systems, and the course staff is here to help.

The dependencies to build Idris 2 are the following:

- a UNIX-derived compiler toolchain, including `bash`, `make`, etc.

  Linux: Most distributions include this by default. If you don't have basic development tools installed, consult the package manager for your distribution.

  MacOS: You will need the Apple command-line tools. If you don't already have them installed, run `xcode-select --install` from a terminal and follow the instructions. Unfortunately, Apple's command-line tools are missing one command needed to build Idris 2, called `realpath`. The easiest way install this (as well as the next dependency) is with the *homebrew* package manager, found at `https://brew.sh/`. With homebrew installed, you can install `realpath` using the command `brew install coreutils`.

  MS Windows: There are several ways to install these tools in Windows, including *MSYS2* and *Cygwin*. If you already have one of these installed, try using it in the following steps.

- Chez Scheme (`https://cisco.github.io/ChezScheme/`)

  Linux: You should be able to install this using your system's package manager. If for some reason you need/want to build it from source, see the instructions at `https://github.com/cisco/ChezScheme/blob/master/BUILDING`.

  MacOS: If you have homebrew installed, just run the command `brew install chezscheme`, otherwise, see the build instructions above.

  MS Windows: The Chez Scheme project page offers a binary installer for Windows at `https://github.com/cisco/ChezScheme/releases/`. If that doesn't work for you, the build instructions above contain a section specific to Windows.

Once you have the compiler tools and Chez Scheme installed and in your search path you can install Idris 2 itself. Currently, the latest release is version 0.3, which is the version we will be using. It is downloadable from:

`https://www.idris-lang.org/pages/download.html`

Detailed build instructions are found at:

`https://github.com/idris-lang/Idris2/blob/master/INSTALL.md`

But you should only need to follow steps 1 (choosing an installation location and adding it to your path) and 2 (compiling and installing the Idris 2 executable from the pre-generated Scheme code).

If there are no errors then you should be able to run the Idris 2 REPL with the command `idris2`, and see the following:

```
     ____     __      _          ___
    /  _/___/ /____(_)____    |__ \
    / // __  / ___/ / ___/    __/ /      Version 0.3.0
  _/ // /_/ / /  / (__  )    / __/       https://www.idris-lang.org
 /___/\__,_/_/  /_/____/    /____/       Type :? for help

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

You can quit using `:q`.

Note: Because the Idris run-time has only recently been ported to Chez Scheme, the command-line history and tab-completion functionality previously provided by the `readline` C library has not yet been implemented in the REPL. Command-line history can be recovered by using the `rlwrap` command, which is available from most Linux package managers, homebrew for MacOS, and MSYS2 for Windows.

Once you have installed Idris 2, you should install an interactive editor mode for your program editor of choice. Idris supports interactive editing with (at least) the following program editors:

Atom (package `language-idris`): see `https://atom.io/packages/language-idris`

VSCode (package `idris-vscode`): see `https://github.com/meraymond2/idris-vscode`

Vim (package `idris-vim`): see `https://github.com/idris-hackers/idris-vim`

Emacs (package `idris-mode`): see `https://github.com/idris-hackers/idris-mode`

Most of these interactive editor modes have not yet been updated for full compatibility with Idris 2, so some of their more advanced features may not work property. We can live with this for now.

Interacting with Idris

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

Task 1

- 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`).

- Open this file in your program editor of choice. Enter the following lines:

```
small  :  Integer
small  =  7

large  :  Integer
large  =  small * 6
```

  Then ask the Idris interactive mode to save, type check, compile, and reload your file. The key-chord for this depends on your operating system and editor. Usually it is some command-key combination together with "r" for "reload". Consult the documentation for the specific interactive mode that you chose above.

- Back 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. 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 it is an `Integer`.

Task 2
Now you will interactively write a function that computes the average of two `Double`s.

- Type the following line into your program and reload:

```
average  :  Double -> Double -> Double
```

- Place the cursor somewhere over the function name and ask Idris to automatically add a definition clause. Again, 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". You should see something like the following:

```
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 a `Double`, and that there are two `Double` bound variables in scope.

- Complete the definition of the `average` 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.

Task 3
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. If you encounter difficulties, try using goals in order to see the relationship between what you need and what you have at hand. The `cast` function from lecture (`:doc cast`) may be helpful here.