CMPT 383: Vitamin #0

Anders Miltner
miltner@cs.sfu.ca

Introduction

This "Vitamin" is here just to help you set up and install a Haskell development environment, set up and install your IDE, and learn some commands for interacting with Haskell.

There is no submission or grading for this Vitamin. You should attend office hours if you have difficulties with setup.

1 Installing Docker Engine and Docker Compose

We need to install the Docker Engine and Docker Compose. The easiest way to do this is to just get Docker Desktop.

2 Prepare and Install Image

The `docker/synced` directory will be where all your code goes. Place the full docker folder in a permanent location.

Navigate in a terminal to the `docker` folder. Run `docker compose up`. This may take a few hours to finish.

3 Installing VS Code

Throughout the class, we'll use Visual Studio Code as our default text editor. I will be demoing on the VS Code editor throughout the class. If you want to use a different text editor, that is fine, but we won't be able to provide much "Tech Support" for installation and use.

Download VS Code from https://code.visualstudio.com/download.

4 Setting up a Haskell Environment Within VS Code Docker Image

Follow the below instructions exactly. Do not skip any steps, or you may mess up your environment.

1. Open VS Code.
2. Ensure your Docker image is running.
3. Click the "Extensions" tab
4. Search "Dev Containers"
5. Click "Install" on the "Dev Containers" extension, by developer "Microsoft"
6. Click the lower-left "><" symbol
7. Click "Attach to Running Container" and attach to the Haskell docker image
8. Open the `~/synced/a0` folder.
9. Click the "Extensions" tab

10. Search “Dev Containers”
11. Click “Install” on the “Haskell” extension, by developer “Haskell”
12. Navigate to the “src” directory, and open “Lib.hs”
13. The extension will request permission to download various files, which should be granted
14. Wait for the extension to install all files and set up the project. This may take a few minutes for the first time. The setup is done when there are small “Refresh” clickable items underneath `helloWorldString`.

5 Running Your First Haskell Program

Like many languages, executing a Haskell program runs the code in the “main” function. We can compile and run a Haskell program in our IDE. Open up a new terminal in VS Code (Terminal > New Terminal).

The terminal should print out `v0 $`. Type `stack build` into the terminal and press enter. You should see a few lines of build information. To run the newly compiled program, type `stack run v0-exe`, which should output “Hello, World!”.

Note, some of your Vitamins can only run ‘stack test‘ as they don’t have an executable.

6 Running Haskell Test Suites

The Haskell build system we are using has built-in support for running test suites. In the open terminal, type `stack test`. This should output “(Empty) Test suite ran!”.

7 Getting Evaluation and Type Information via Haskell Language Server

The Haskell Language Server (HLS) that underlies the VS Code Haskell extension is quite powerful. The way the HLS permits human interaction is through comments. In this section, we will show how to extract evaluation and type information from the HLS using comments.

After the `main` function are 4 other comment snippets that interact with the HLS. The first one should look like `-- >>> helloWorldString`. The `--` at the start initiates a single-line comment, similar to `//` in Java and C. The `>>>` that follows initiates an interaction with the HLS. What type of information you get back depends on what follows.

In the first comment, we simply write some Haskell code after `>>>`. This signals to the HLS that you wish to evaluate that code, then print it out. In this example, it appends `!!` to `helloWorldString`. Click “Refresh” to evaluate that code snippet, and see the result.

The next comment begins with `-- >>>`, but then has `:t` before the Haskell code. This means that we wish to get the type of the code that follows. Click “Refresh” type-check that code snippet, and see the result.

The next comment is a different type of comment. It is a multi-line comment, similar to `/*` and `*/` in Java and C. Within multi-line comments, you can also interact with the HLS by adding `>>>`.

In the last line, we do another type-checking. But this doesn’t check a single variable, but a full expression. Just as you can evaluate expressions with the HLS, you can also type-check them! Try adding your own!