Hassling with the IDE - for Haskell

This guide is for students that are new to the Haskell language and are in need for a proper **I**ntegrated **D**evelopment **E**nvironment (IDE) to work on the projects of the course.

This guide is intended for Windows users. It may work similarly on Linux and iOS. For more information, see:

- https://www.haskell.org/platform/mac.html for iOS
- https://www.haskell.org/downloads/linux/ for Linux

For any errors, please contact me at w.b.wijnia@uu.nl.

Changelog

04/09/2019	Initial version
24/08/2020	Updated contents for the year 2020 / 2021, thanks to Bernadet for pointing it out

1. Installing Visual Studio Code

Visual Studio Code (VSC) is a lightweight, but rather extensive, code editor. We'll talk about its features throughout this guide. You can download VSC at:

```
https://code.visualstudio.com/download
```

Once installed, take note of the sidebar on the left. From top to bottom, it contains:

- **Explorer** (Ctrl + Shift + E)
- Search (Ctrl + Shift + F)
- **Source control** (Ctrl + Shift + G)
- Debug (Ctrl + Shift + D)
- **Extensions** (Ctrl + Shift + X)

The **bold** options are important to remember. They will be mentioned throughout this document.

2. Installing GHC (and stack / cabal) on Windows

Download the Glasgow Haskell Compiler (GHC). This compiler is used throughout this course and any other course that uses Haskell. You can find its download instructions at:

```
https://www.haskell.org/platform/windows.html
```

The download link recommends you to use Chocolatey. Chocolatey is an open source project that provides developers and admins alike a better way to manage Windows software. Follow the instructions provided by the Haskell platform, those instructions are quite concise. A few common mistakes:

- Do not use the Command Prompt (CMD) - use PowerShell as mentioned
- For the installation of Chocolatey and GHC administrator privileges are required, ensure that you open up PowerShell with administrator privileges

Open up a console once the installation is complete and type the following commands:

```
// in some PowerShell session

stack --version
cabal --version
ghc --version
```

These commands will check the version of the programs on your system. If these execute successfully then the installation process has completed successfully.

Once again inside a console, type the following commands:

```
// in some PowerShell session

stack update
stack upgrade
```

These commands update the list of packages available to stack and upgrades all packages (including stack itself) already installed. This will prevent errors down the road, such as outdated or unknown packages.

The ‘stack upgrade’ command may fail. Typically this happens because it doesn’t have the administration rights to move the new stack executable over the old one. If this does happen, read carefully what the operations tried to do and then do these manually. This involves a copy and a move operation, overwriting the stack provided by chocolatey.

3. Installing hlint

The tool hlint is used by DomJudge to check for your coding style. It is a great tool to increase your understanding of your code and experiment with your understanding of the type system of Haskell. You can find more information at:

```
https://hackage.haskell.org/package/hlint
```

Take note that the tool is **not** perfect. Do not follow it blindly: always try to understand the hints and tips that HLint is providing for you.

You can install hlint globally with:

```
// in some PowerShell session

stack install hlint
```

This installation process can take some time. If it fails, make sure you have upgraded Stack properly. A few useful commands you can run after installing:

```
// in some PowerShell session

hlint <path-to-file>.hs
hlint <path-to-folder>
hlint .
```

The output of hlint is directed to the console. The first command applies hlint to a given file, the second to all files in a specific folder and the latter to all files in the current folder.

Take note that environment variables (such as where to find hlint) are changed. Always restart any PowerShell session before attempting to use new commands that are recently added to your environment variables. In the case of Visual Studio Code, restart the entire application and not just the terminal.

4. Setup of Visual Studio Code workspace

We'll initialise a workspace and generate a local git repository. The git repository can be used as a versioning system for your own convenience.

4.1. Workspace

Choose or create a folder in which you'll do all your local Haskell development. Inside this folder you can create a hierarchy for the assignments, but I'll leave that to you.

Inside Visual Studio Code (VSC) you can open up this folder as your workspace. In turn, VSC can assist you with your development.

```
// in the toolbar of Visual Studio Code (VSC)

File -> Open Folder                                    CTRL + K + O
```

About time to add in some Haskell files in our workspace. If your explorer is hidden, use 'CTRL + Shift + E' to open it up. Right click on the explorer and create a new file called 'scratchpad.hs'. We'll use this file to toy around with our options.

Open up the file by clicking on it twice. Clicking on the file twice will permanently open the file inside VSC, instead of only temporarily when you click only once. Add in the following definition to the file and save it:

```
sum' :: [Int] -> Int
sum' (x:xs) = x + sum' xs
sum' [ ] = 0
```

We can open up a PowerShell inside VSC. Any terminal opened inside VSC is automatically navigated towards your current folder.

```
// in the toolbar of Visual Studio Code (VSC)

// to make a new terminal
Terminal -> New Terminal                               CTRL + Shift + ~
// to open / close the terminal window
View -> Terminal                                       CTRL + ~
// to view multiple terminals
Terminal -> Split terminal                             CTRL + Shift + 5
```

From the PowerShell session we can start an interactive GHC session, called ghci.

```
// in some PowerShell session

ghci
```

Inside this session we can dynamically compile and run code. It is extremely useful to debug individual functions throughout the first few weeks of the course. As an example, try to load in the file we previously created.

```
// in some ghci session

:load scratchpad.hs
```

And then call the function we defined with some list as its argument.

```
// in some ghci session

sum’ [1,2,3]
```

The GHCi session provides more useful functions.

```
// in some ghci session                  // shortcuts

:load <file>                            :l <file>
:reload <file>                          :r <file>
:quit                                   :q
:type <function>                        :t <function>
:import <library / package>             :i <library / package>
```

You can load and reload files to check whether they compile and, if they do, test their functionality. Take note that GHCi only loads the Prelude package by default. If you want to toy with functions from other packages you first need to load them. All packages from files are loaded accordingly.

Take note that you cannot use hlint functionality inside a GHCi session. Open up an additional terminal or split your current terminal. This allows you to easily access both GHCi for debugging and hlint for linting purposes.

4.2. Local git repository

Assuming that you have a folder selected you can start a local repository from VSC, allowing you to have versioning control on your code. Go to the **Source Control** tab and click on the '+' button to the right of 'Source Control'. Choose your current folder and initialize a repository.

From here on you can do everything that you expect from Git, such as staging changes (right click inside the Source Control tab -> Stage Changes), Commit your changes (the 'checkmark' button) and all other functionality available through Git under the '...' button.

Git is the industry standard to version control. If you're unfamiliar with Git then rest assured that you do not need it to pass the course. Making yourself familiar with Git can therefore only be beneficial to you: it will help you work faster, collaborate better and become more professional.

5. Visual Studio Code extensions and tips

5.1 Extensions

A useful extension is the 'Haskell Syntax Highlighting' extension. It adds all kinds of pretty colors in VSC. Go to the **Extensions** tab, search for it and install it.

Take note that there are a few auto-completion extensions available. Extensions such as Haskero or Haskelly. Installing these can be difficult and system dependent therefore we do not provide support for installing these extensions.

You can program in Haskell without auto completion and it is better for the sake of learning.

5.2 Useful hotkeys

`Ctrl + D`	`Select a keyword, then press CTRL + D. Press the D repeatedly. Selects all keywords that match. You can change them at once.`
`Ctrl + H`	`Search for the selected keyword and change them all.`
`Ctrl + F`	`Search for the selected keyword.`
`Ctrl + Shift + F`	`Search for the selected keyword through all files in your workspace.`
`Ctrl + B`	`Open / close the sidebar.`
`Alt + CLICK`	`Repeats the cursor at the given location.`
`Alt + UP / DOWN`	`Moves the current line up or down.`
`CTRL + ~`	`Open or close the terminal window`
`CTRL + Shift + 5`	`Open up a split terminal`
`CTRL + Shift + ~`	`Open up a new terminal`

And don't forget to *just click* on buttons within the IDE to better understand what it can offer you.