Introduction to Programming Utilities

Lecture 2: Text Editors and Compilers

Course Syllabus

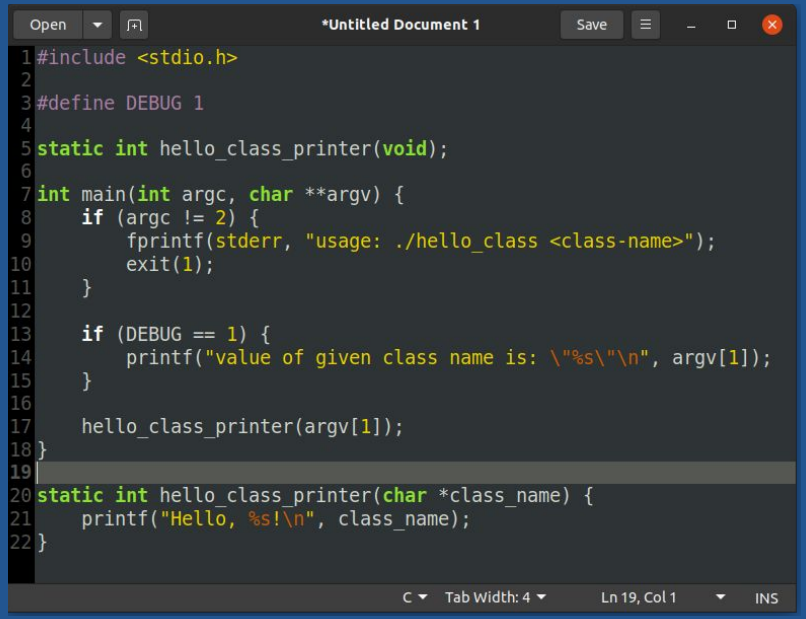
1. **Linux** and **The Command Line**
2. **Text editor** and **Compiler** (*This Lecture*)
3. Basics of **Git** and **GitLab**
4. Integrated Development Environment (**IDE**)
5. **Advanced Git** for Group Projects

This Lecture: **Text Editor** and **Compiler**

- What is a text editor?
- How do I choose which text editor to use?
- How to set up a text editor on your own computer
  - Demo using Atom
- What is a compiler and why do we need it?
- How do we install and use a compiler?
  - Demo using GHC/GHCi

Text Editors

What is a Text Editor?



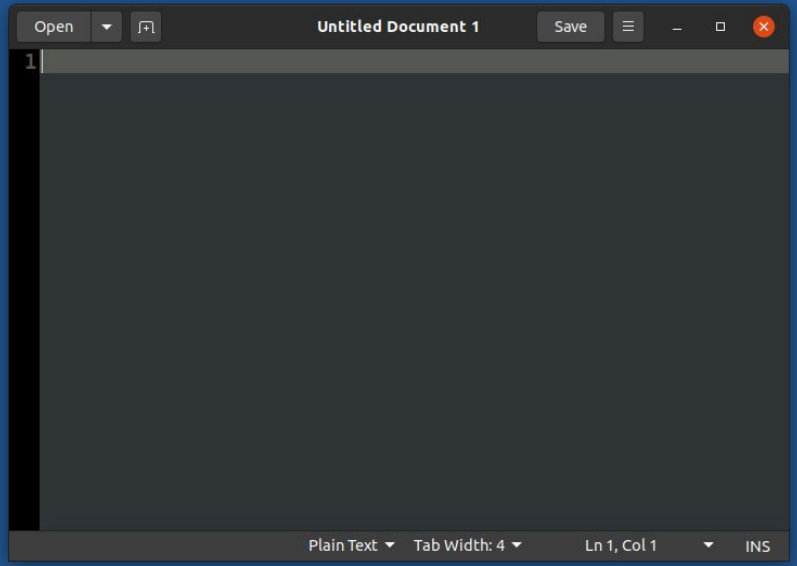

```c
#include <stdio.h>

#define DEBUG 1

static int hello_class_printer(void);

int main(int argc, char **argv) {
    if (argc != 2) {
        fprintf(stderr, "usage: ./hello_class <class-name>");
        exit(1);
    }

    if (DEBUG == 1) {
        printf("value of given class name is: \"%s\"\n", argv[1]);
    }

    hello_class_printer(argv[1]);
}

static int hello_class_printer(char *class_name) {
    printf("Hello, %s!\n", class_name);
}
```

- *"Computer program that edits plain text documents"*
- Source code is also just a plain text file!
  - You ***can*** technically use any text editor
- But there are some text editors that are better than others

Some preferable features for programming

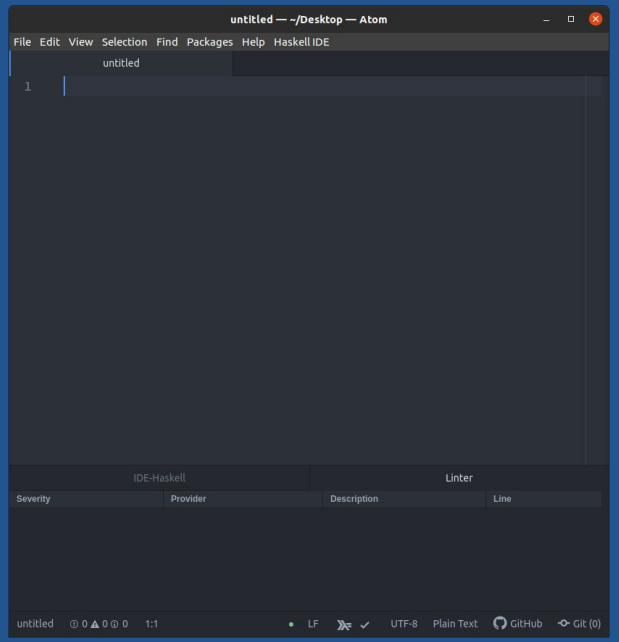
- Syntax highlighting
- Line numbering
- Auto-close brackets and quotes
- Automatic indentation
- Automatically convert tabs to spaces (“soft tabs”)
- Multi-cursor
- Vertical rulers for maximum line length
- Customisable user interface
- Keyboard shortcuts
- Some refactoring features
  - Search and replace
  - Renaming identifiers of variables, functions, etc.
  - Linter support
  - Comment multiple lines w/ shortcut
- Version control (i.e. Git) integration
- Plugin support

gedit



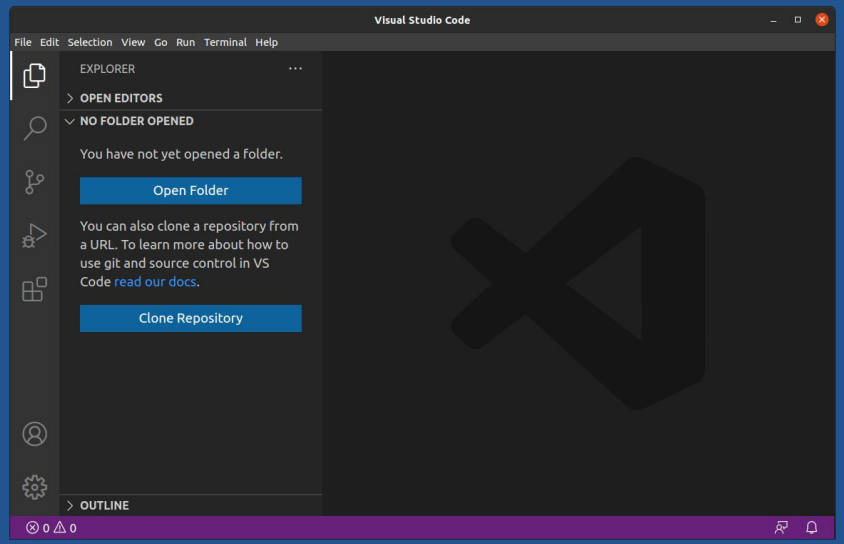
- Default text editor for Ubuntu
- Not that sophisticated
- Some basic features for programming
  - Syntax highlighting
  - Line numbering
  - Vertical ruler
  - Soft tabs
- Not great for any serious programming

Atom



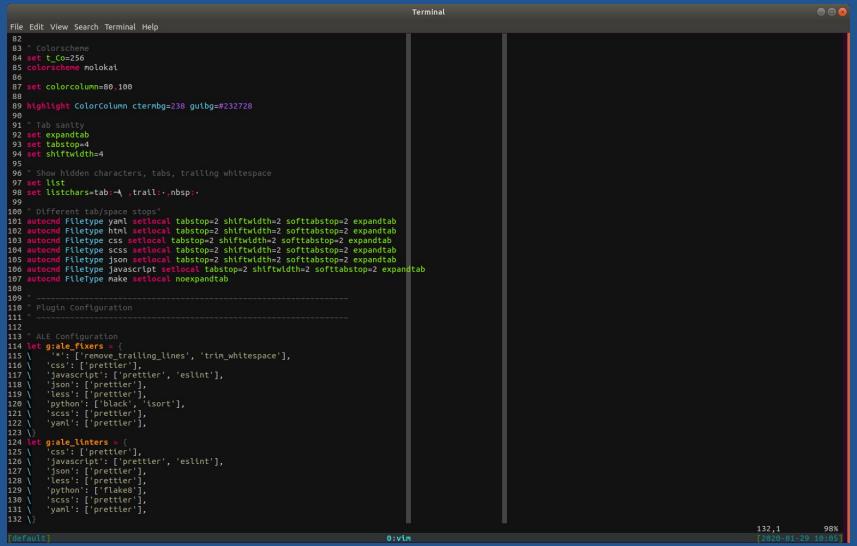
- Developed by GitHub
  - *"Hackable Text Editor for the 21st century"*
- Has essentially all features preferable for programming
- *Many* plugins you can install extend its feature set
- Also highly customisable
- Installed by default on all DoC machines
  - Reason why I recommend Atom for Haskell
- Configuring it for Haskell can get complicated
  - We will go through it in a demo later
- Relatively slow and resource-intensive

Visual Studio Code (VSCode)



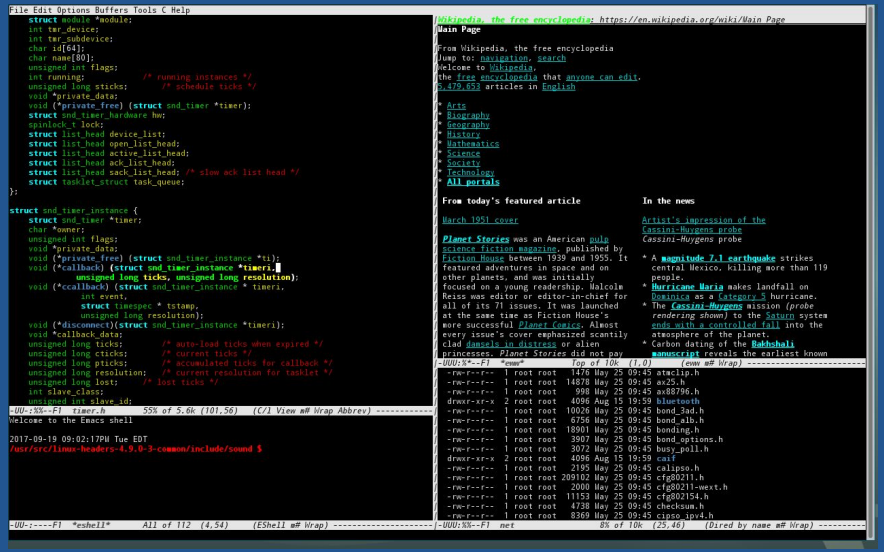
- Developed by Microsoft
  - Free, open-source source code editor
  - Stripped-down version of Visual Studio
- Has all features for programming
- Extensive list of plugins
- One of the best source code editors, especially for web development
- Also doesn’t support Haskell by default

Vim



- Command-line text editor
- *Very* steep learning curve at first
- But also *very* customizable
  - Increase workflow speed significantly
- Not approachable for beginners
- Not useful for Haskell

Emacs



- Command-line text editor
- *Very* steep learning curve at first
- But also *very* customizable
  - Increase workflow speed significantly
- Not approachable for beginners
- Not useful for Haskell

So which text editor should I choose for Haskell (at Imperial)?

- Installed by default on all DoC machines
- What the department endorses for Haskell
- Easy (enough) to use
- But very hard to set up without guidance
  - Not enough documentation on Atom for Haskell

Reference sheet - Installing Atom w/ Haskell

1) Install **Atom**: https://atom.io/ - download .deb package
2) Install **Haskell Platform** (for Haskell support) - https://www.haskell.org/platform/
3) Install **Stack** (for installing Haskell binary dependencies of Atom plugins) by running the following commands on terminal:

```
wget -qO- https://get.haskellstack.org/ | sh
echo "export PATH=$HOME/.local/bin:$PATH" >> ~/.bashrc
```

(refer https://docs.haskellstack.org/en/stable/install_and_upgrade/ for details)

4) Install **binary dependencies** by running the following commands on terminal

```
stack install stylish-haskell
stack --resolver lts-9 install ghc-mod
```

(refer https://atom-haskell.github.io/installation/installing-binary-dependencies/ for details)
Then get the executable's path (e.g. "/home/<your-username>/.local/bin") and save it

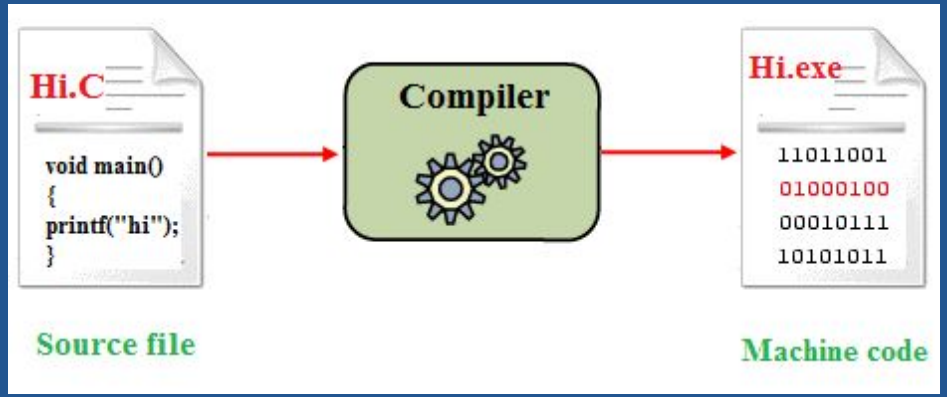
5) Get Atom **packages** (or plugins)
Find atom-haskell package OR run `apm install atom-haskell`

6) Go to haskell-ghc-mod settings and **add the executable's path** to "additional path directories"

Demo 1: Text Editors

Compilers

How source code is converted into a runnable program



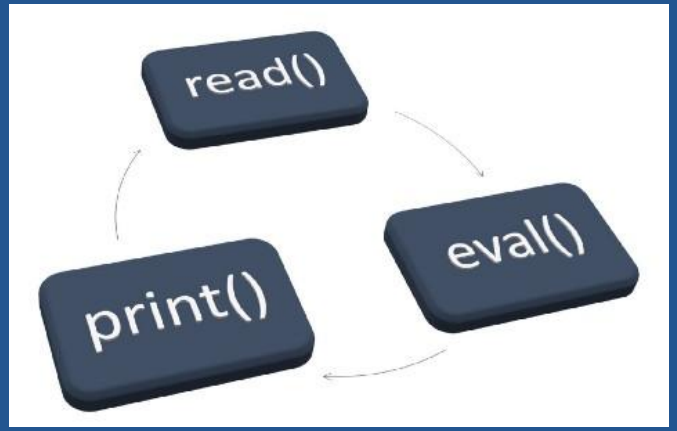
- **Compiler** : converts source code into an executable program
- Source code is just a text file
- Computers cannot understand text
  - “Computers don’t speak English”
  - They only read 0s and 1s (Machine Code)

```
print(“Hello, world!”)
=>
000100100111010110110 00.....
```

GHC (Glasgow Haskell Compiler)

- Compiler for Haskell source code (.hs files)
- Gets the “main” function in a file
- Does the action of whatever the main function does
- Realistically never used for this course
  - You never really create an executable program for Tutorial exercises and LEXIS Tests
- Instead create program “modules” that you load into an interactive REPL
  - Module: a collection of data and functions wrapped up in a single source code file/package
  - *REPL*???

The REPL (Read-Evaluate-Print Loop)



- Repeatedly reads code, evaluates it, and prints the result
  - Read : get user input (code statement)
    - e.g. calling a function
  - Evaluate : evaluates the given code and run it
  - Print : print the result of what was evaluated

GHCi (GHC interactive)

```
hk619@hk619-ThinkPad-X1C7: ~/Docume...
hk619@hk619-ThinkPad-X1C7:~/Documents/lecture-2
/calculator-demo$ ghci
GHCi, version 8.6.5: http://www.haskell.org/ghc
/  :? for help
Prelude> :l
Calculator.hs  Tests.hs       IC
Prelude> :l Calculator.hs
[1 of 1] Compiling Calculator       ( Calculato
r.hs, interpreted )
Ok, one module loaded.
*Calculator> add 1 2
3
*Calculator>
```

- **`ghci`**
- Haskell's REPL Interactive environment
- Load modules you created into GHCi
- "Prelude" module is loaded by default
  - Contains all basic functions you need
- Call functions
- See the printed result

Some useful commands for GHCi

- **`:l`** *`<module/file>`*
  - load a file/module into GHCi to use
- **`:r`**
  - Reload all modules loaded on the current environment
- **`:browse`** *`<module>`*
  - List all functions with its types in the module
  - Very useful for LEXIS tests
- **`:q`**
  - Quit the current GHCi session

Demo 2: Compilers

Department of Computing Society

Imperial College London

docsoc.co.uk | fb.com/icdocsoc | twitter.com/icdocsoc