COMP20003 Workshop Week 2

Welcome to the First Workshop!

- Networking...
- Algorithms & Programs
- Remember C? Data Structures?
- Computing environment for this course
- Lab: make, gdb and Notebook exercises

Algorithms: How search engines work?

- Input:
  - a HUGE collection of text docs (say, each is an article, a book) - think about a digital library, or a collection of web pages.
  - a query, such as:
    - `"algorithm" AND ”data structures”`
    - `data structures in C`
- Output: find documents that best match the query.
- Algorithm = ? Data structures= ?
- What’s important for ALD?

C Programming

- `scanf` & `printf`
- `if`, `while`, `for`, and functions
- basic date types, strings, arrays, `struct`
- **pointers**

C Programming

- `scanf` & `printf`

type	`int`	`float`	`double`	`char`	*`string`*
`printf` format	`%d`	`%f`	`%f`	`%c`	`%s`
`scanf` format	`%d`	`%f`	`%lf`	`%c`	`%s`
`scanf` for **`v`**	`&v`	`&v`	`&v`	`&v`	`v`

- `if`, `while`, `for`, and functions
- basic date types, strings, arrays, `struct`
- **pointers**

C Programming

- `scanf` & `printf`
- `if`, `while`, `for`, and functions
- basic date types, strings, arrays, `struct`
- **pointers == what?**

Computing Environment: Lab's PCs

Window 1	Window 2

A text editor.
Recommended: `jEdit`
Better: Atom, but not stable
 	
Note: save files in
`H:COMP20003/week2/`
 	
`Do it now: start jEdit/Atom for this Window, start mobaXterm/minGW for the other window. Arrange the 2 windows so that you can see both. Then, create helloworld.c.`
	A terminal in `dimefox` or `nutmeg`.
	
	Recommended procedure:
	
	(1) run `mobaXterm` or `minGW`, then:
	
	`ssh you@dimefox.eng.unimelb.edu.au`
	
	(2) run some `Unix` commands like:
	
	`$ mkdir COMP20003`
	`$ cd COMP20003`
	`$ mkdir week2`
	`$ cd week2`
	`$ ls`
	
	(3) compile, run, debug, test your program directly on `dimefox`. See LMS for more `Unix` commands.

Computing Environment: Your Laptop

Window 1	Window 2
Install `jEdit` or Atom or … but don't use IDEs Note: save files in local driver such as `C:COMP20003/week2/` **Not ready? You'd better to do at home.**	For Windows only: install **mobaXterm** or **minGW**. For all laptops: • make sure you have **ssh**, **scp**, **gcc**, **gdb**, **valgrind** • install **Docker** • install **VPN** Test programs in your local drivers, or copy them to **H:** and use **dimefox**/**nutmeg**

copy folders/files between your laptop and uni's driver **H:**

scp –r <name_of_folder> **you@dimefox.eng.unimelb.edu.au:**

Example:

scp –r week2 bob@dimefox.eng.unimelb.edu.au:comp20003/

ssh bob@dimefox.eng.unimelb.edu.au

cd comp20003/week2

Try some unix commands

`pwd`	display the name of current folder (CF)
`cd`	change CF to my home folder
`ls`	list the content of CF
`mkdir comp20003`	make a sub-folder `comp20003` under CF
`mkdir comp20003/week2`	
`cd comp20003/week2`	
`echo Hello! >test.txt`	make a new file with content "`Hello`!"
`more test.txt`	display the content of file `test.txt`
`cp test.txt test1.txt`	make a copy `test1.txt` of `test.txt`
`ls`	
`rm *.txt`	remove all files ended with `.txt` **[dangerous]**
`man rm`	display info about command `rm`
`man scp`	

Download from github.com

- In your browser, navigate to `github.com/anhvir/c203`, and follow the instruction there.
- Then, move or copy the directory `Downloads/c203-master` to under `comp20003/week2/`

make & Makefile

- Explore the downloaded `Makefile` then try:

```
cd ~/comp20003/week2/c203-master
cp Makefile1 Makefile
./afriend
make
make clean
make
```

- To learn more about Makefile: explore `Makefile2, Makefile3` (in that order). They are all equivalent to Makefile1. To use, say, `Makefile2` just run:

```
cp Makefile2 Makefile
make clean
make
```

Debugging

- Debugging == finding bugs in (faulty) programs
- How?
  - Insert some print statements in questionable places to examine value of some variables…
  - Use debugging sofware such as **gdb**
- Resources for **gdb**: in **LMS** or
  - https://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf
  - **google** it
- Now, try to compile and run **random_friend.c** Bugs? use **gdb**
- Note: We'll talk about **malloc()** and **valgrind** next week.

Debugging with gdb

- Two steps:
  - compile with –g flag, for example:

```
$ gcc –g –o afriend random_friend.c
```

  - `$ gdb afriend`

- Inside `gdb`:
  - **r(un)** (**run** or **r**) : running program test
  - **b(reak)** <function name> : set a break point
  - **b(reak)** <line number> : set a break point
  - **p(rint)** <expr> : print value of expr
  - **c(ont)** : continue prog execution
  - **n(ext)** : run next statement
  - **help**
  - **q(uit)**

Lab

- Play with "`Makefile`" and "`afriend`" to understand how make/Makefile work
- Have a look at `Makefile1,2,3`: they are all equivalent!
- Use `gdb` to debug `random_friend.c` and `debug_me.c`
- Do exercises in Notebook [OR follow instructions in the .PDF file and do exercises in your system]

Remember

- Algorithm is fun! Stay active, stay happy!
- Arm your laptop/desktop with necessary working tools (`VPN`, `jEdit`/`Atom`, `mobaXterm`/`minGW`, `gcc`, `gdb`…)
- `Google` is our good friend!
- Keywords:
  - **ssh**, **ls**, **pwd**, **cd**, **mkdir**, **rm**, **cp**, **more**, **cat**
  - **mobaXterm**, **minGW** & **ssh**, **jEdit** & **Atom**
  - **make**, **Makefile** (targets, dependencies, recipe, CC= gcc…)
  - **gdb**: **b**(**reak**), **r**(**un**), **p**(**rint**), **s**(**tep**), **w**(**atch**)