COMP20007 Workshop Week 2

Welcome to the First Workshop!

1	• When waiting: do networking (chat with, and get to know classmates, talk about some interesting stuffs)
2	• Arrays, Linked Lists, ADT: Stacks, Queues Tutorial questions 1→5
3	• LAB: C revision with functions.c and (time permitted) listops.c

Networking

- Know your mates and their programming experiences, favourite data structures, and interesting/funny stuffs
- Tutor: Anh Vo == Vo Ngoc Anh, email avo@unimelb.edu.au with subject starting with "COMP20007" or just "C207". Note: you should use uni's email.

Workshop Format

- At home, before each workshop:
  - attend/watch and review the previous week lectures, and
  - from LMS: download, read and attempt tutorial and lab problems.
- First hour: theoretical/algorithmic problems
  - Have papers/booklets and pens/pencils ready 🤪🤪
  - Q&A,
  - solve workshop problems individually and collectively,
  - Motto: to understand, we you should attempt the problems ourselves.
- Second hour: *mainly* algorithmic/programming problems, lab
  - have fun with (multi-file) C programming,
  - ask algorithmic/programming questions, and
  - learn algorithmic and C skills from classmates and tutors.

Problems, Algorithms, Pseudocode, C code

Big-O: O(n), O(log n), O(1)

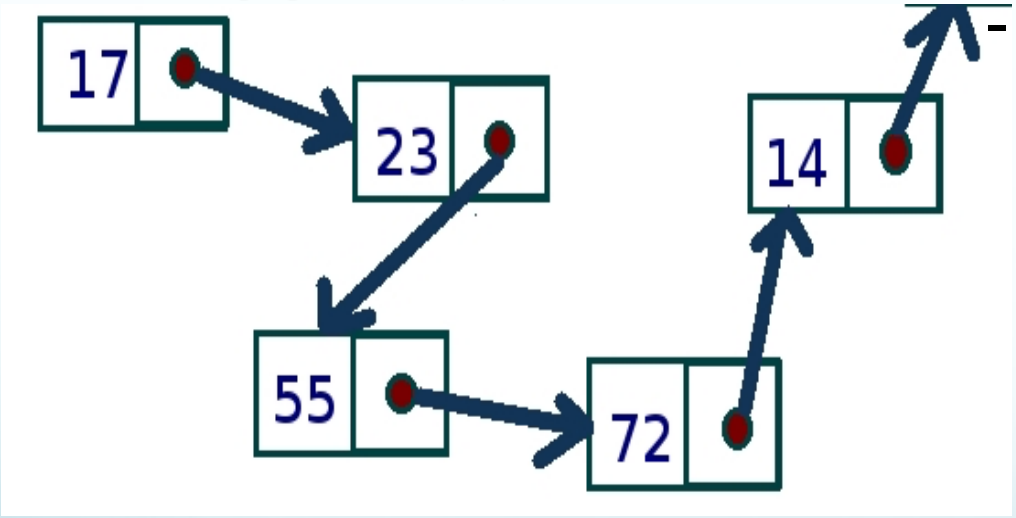
Data Types & ADT

- A *concrete data type*, such as array or linked list, specifies a representation of data, and programmers can rely on that to implement operations (such as `insert`, `delete`).
- An *abstract data type* specifies possible operations, but not representation. Examples: stacks, queues, dictionaries.
  - When implementing an ADT, programmers use a concrete data type. For example, we might attempt to employ array to implement stack.
  - When using an ADT, programmers just use its facilities and ignore the actual representation and the underlined concrete data type.

Two concrete data types: Arrays & Linked Lists

Access A[k] in O(1) time!

Access L[k] in O(n) time!



In C:

- How to specify an array? How to traverse it?
- How to specify a linked list? How to traverse it?

Problem 1: Arrays & Linked Lists

- Describe how you could perform the following operations on sorted and unsorted arrays, and decide if they are *O(1)*, *O(log n)*, or *O(n)*, where *n* is the number of elements initially in the array. Assume that there is no need to change the size of the array to complete each operation.

Operation	Unsorted Arrays	Sorted Arrays
Searching for a specified element		
Inserting a new element		
Deleting the final element		
Deleting a specified element		

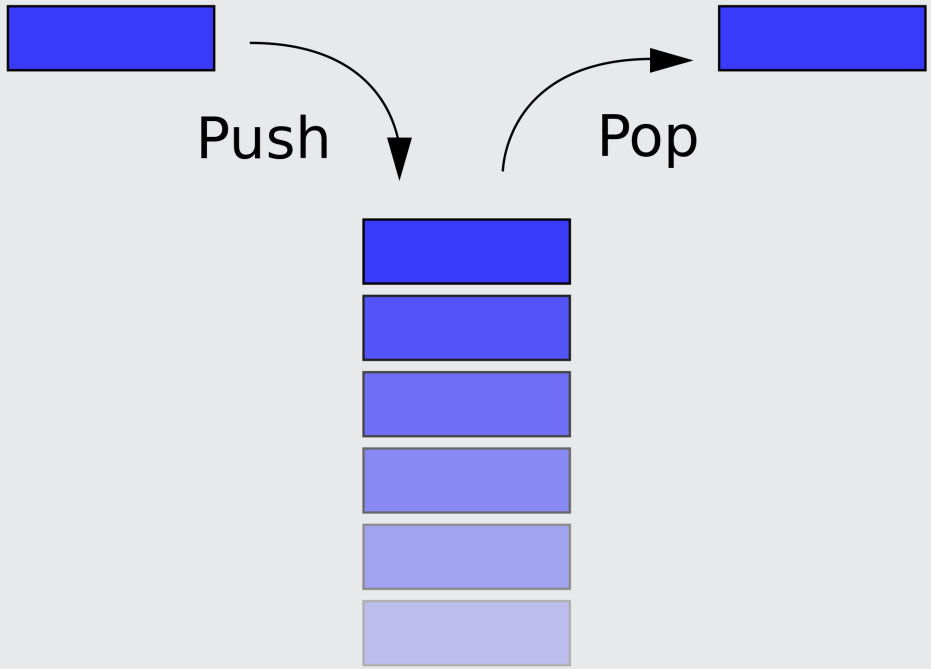
Problem 2: Linked Lists

- Describe how you could perform the following operations on singly-linked and doubly-linked lists, and decide if they are *O(1)*, *O(log n)*, or *O(n)*, where *n* is the number of elements initially in the linked list. Assume that the lists need to keep track of their final element.

Operation	Singly	Doubly
Inserting a node at the start		
Inserting a node at the end		
Deleting the first node (at the start)		
Deleting last node (at the end)		

ADT: Stack (LIFO)

http://www.123rf.com/stock-photo/tyre.html



https://simple.wikipedia.org/wiki/Stack_(data_structure)

Stack Operations

push: add an element into stack

pop: remove an element from stack

isEmpty: check if stack is empty, or

size: return number of elements in stack

Example of using Stacks ?

Stack is widely used in implementation of programming systems. For example, compilers employ stacks for keeping track of function calls and execution.

Stack for :

fact(4)

```
int fact( int n ) {
  if ( n<=1 )
      return 1;
  return n*fact(n-1);
```

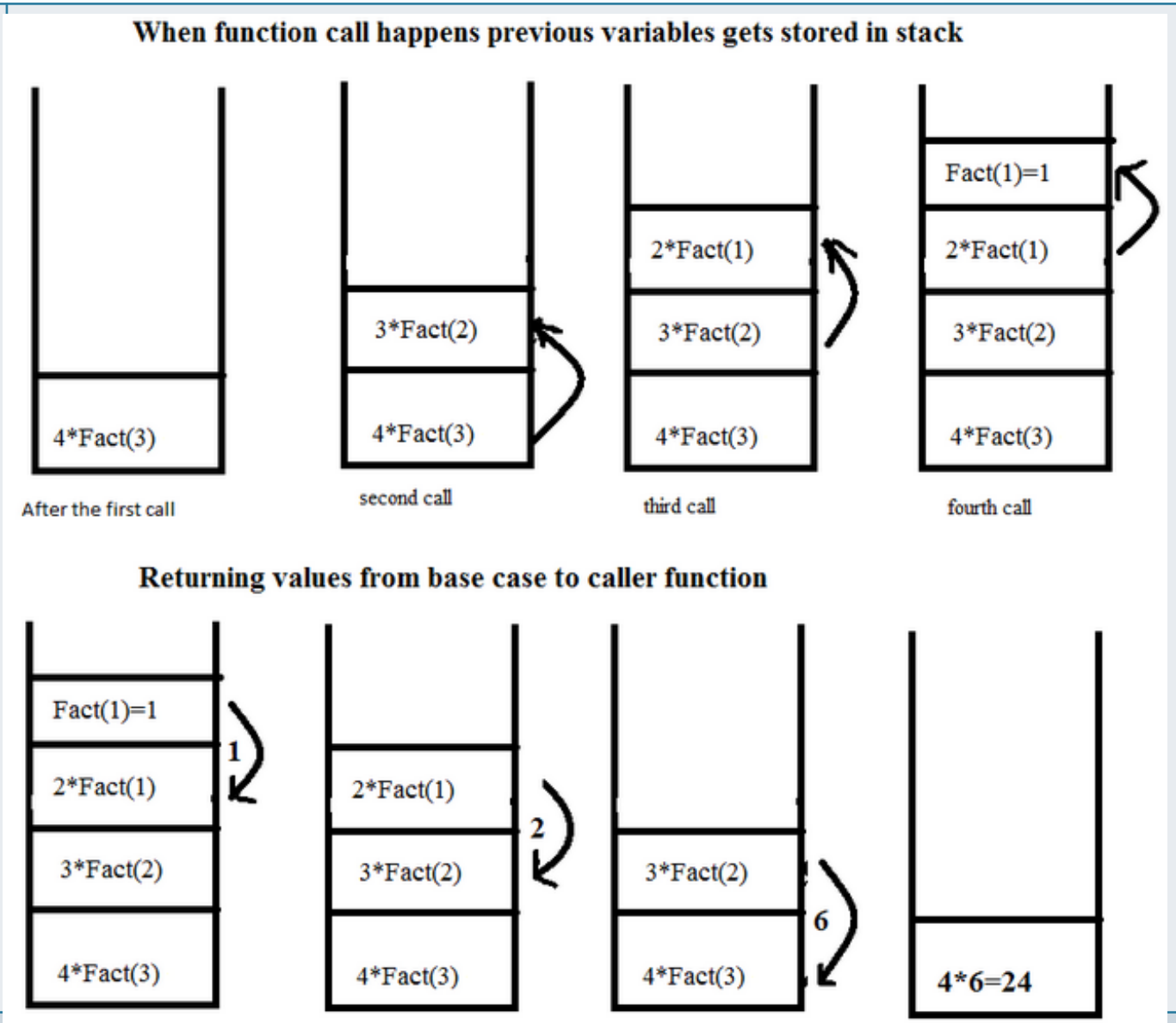


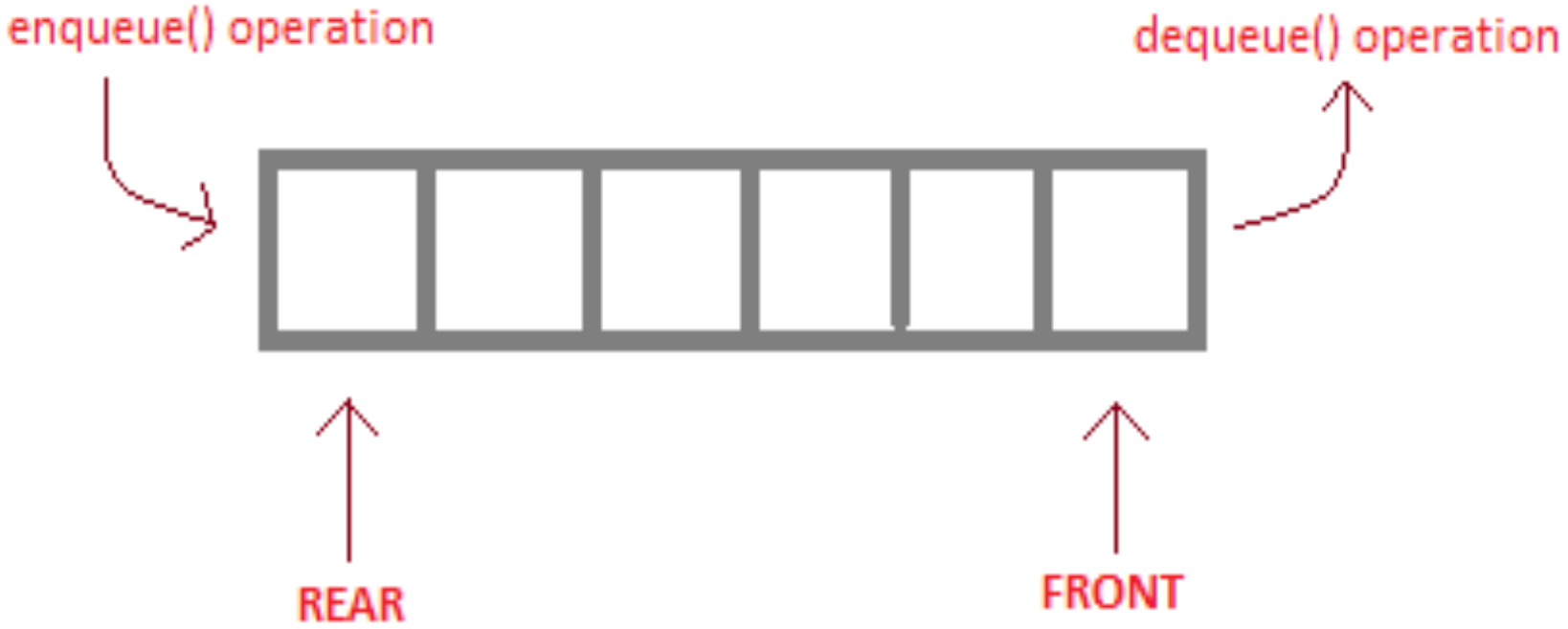
Image source: http://stackoverflow.com/questions/19865503/can-recursion-be-named-as-a-simple-function-call

Problem T3: Stacks

- Describe how to implement `push` and `pop` using an unsorted array, and using a singly-linked list.

Using arrays	Using linked lists

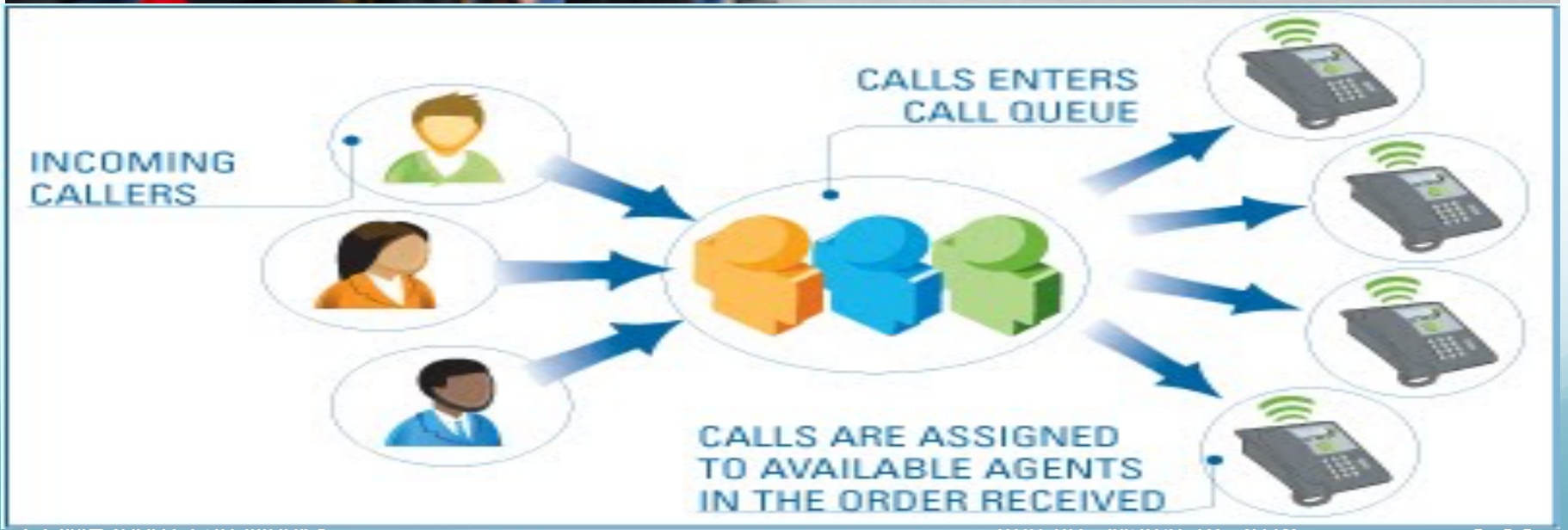
ADT: Queue (FIFO)



enqueue( ) is the operation for adding an element into Queue.

dequeue( ) is the operation for removing an element from Queue .

ADT: Queue (FIFO)



Problem T4: Queues

- Describe how to implement **`enqueue`** and **`dequeue`** using an unsorted array, and using a singly-linked list. Is it possible to perform each operation in constant time?

Using arrays	Using linked lists

Problem T5 [optional]: Stacks & Queues

If you have access only to stacks and stack operations, can you faithfully implement a queue?

How about the other way around?

You may assume that your stacks and queues also come with a size operation, which returns the number of elements currently stored.

5-minute break: making friends OR making an algorithm for making friends

- physical exercises please!

Lab: read instructions in (page 3 of) the workshop sheet

1. *Start* with `helloworld.c`
2. (Together) Implement functions in `functions.c`, which reviews *function and function parameters*
3. *dynamically resizing arrays* with `malloc`/`calloc` and `free`. Forgot `malloc`? Try command "`man malloc`" in your terminal.
4. *Optional*: download Alistair's `listops.c` (just Google to find), then use it to make a *least-effort* implementation of stack with:
   - declaration of data type `lstack_t` (warning: don't use `stack_t`),
   - functions `make_empty_stack`, `free_stack`, `is_empty_stack`, `push`, `pop`. Note: except for the last one, each function should have a single-line body!

Convince yourself that you can implement queue quickly!

Summary

- concrete data types: array, linked lists.
- ADT stack, queue and operations; implementation using array and linked list. Quick implementation of `lstack_t`.
- functions, pointers, `malloc`/`realloc`, `free`.
- dynamically resizing arrays.
- Technical stuffs:
  - using LMS and LMS discussion forum,
  - Editors: `jEdit`, `Atom`, `emacs` or …,
  - Terminal to run `gcc` and programs, simple unix commands: `cd`, `ls`, `cp`, `mv`, `mkdir`, `more`
  - using `man` and `Google`.

ALGORITHMS ARE FUN,

as so will be our future jobs!

Case 1: edit/run functions.c on lab PC

- Suppose you already downloaded `functions.c` from LMS, and it is now under Download folder
- Open that `functions.c` in `jEdit` and use Save As to save to H:
- Now change `functions.c` as required and Save
- Run `MinGW`, and in `minGW` run:

Command	Notes
`cd H:`	use your uni's folder as current directory (CDIR)
`ls`	display content of CDIR, you should see functions.c
`gcc –o functions functions.c`	compile
`./functions`	run functions
	then, back to jEdit to change/save and then compile, run again.

- Note: In this way, `functions.c` will be kept permanently on your uni's folder `H:`. If you didn't save to `H:`, the file will be automatically deletted.

make your own laptop/desktop useful

Aim: be able to edit and run C programs off-line on your own laptop/desktop.

If you can do that already, skip this page! Otherwise:

- Make sure to have an editor (such as `jEdit`) installed.
- For Windows: make sure to install `minGW`. When installing minGW, remember to also mark `open_ssh_bin` for installation. Note: if you installed `minGW` already, run `minGW installation manager` to add `open_ssh_bin`

Case 2a: edit/run functions.c on your Windows laptop

- Suppose you already downloaded `functions.c` from LMS, and put it your working directory (say. C:\comp20007)
- Open that `functions.c` in `jEdit`
- Run `MinGW`. In this `minGW` terminal run:

Command	Notes
`cd C:` `cd comp20007`	use your C:\comp20007 as current directory (CDIR)
`ls`	display content of CDIR, you should see functions.c
`gcc –o functions functions.c`	compile
`./functions.exe`	run functions
	then, back to jEdit to change/save and then compile, run again.

Note: In this way, `functions.c` will be kept on laptop. If something wrong happens to your laptop (oh, I didn't mean that, just if ☺) you will lose `functions.c` and all of your other precious works.

Case 2b: edit/run functions.c on your MacBook

- Suppose you already downloaded `functions.c` from LMS, and opened it in `jEdit`
- Then open a Terminal and run

Command	Notes
`cd`	use home directory (/Users/your_name) as current directory (CDIIR)
`mkdir comp20007` `cd comp20007`	make a new directory change CDIR to comp20007 now, on jEdit, use Save As to save in comp20007
`ls`	display content of CDIR, you should see functions.c
`gcc –o functions functions.c`	compile
`./functions`	run functions
	then, back to jEdit to change/save and then compile, run again.

Note: In this way, `functions.c` will be kept on laptop. If something wrong happens to your laptop (oh, I didn't mean that, just if ☺) you will lose functions.c and all of your other precious works.

Common for both Case 2a and 2b: backup your files using your uni's H: driver

- You can copy your file to your uni's folder and have a good backup. For that you need to run command scp on your Terminal windows

```
scp functions.c your_uni_login_name@dimefox.eng.unimelb.edu.au:
```

(note: there is an colon : at the end). The file will be copied to your uni's H: driver. You can also copy the whole directory comp20007 using:

```
cd ..
cp –r comp20007 your_uni_login_name@dimefox.eng.unimelb.edu.au:
```

You will have a copy of your valuable works in a uni's server! And of course, if you use a lab PC you can edit and run these files directly.

Case 3: use your laptop to edit and run programs in uni's H: directly

Tools: 3a. connect to dimefox/nutmeg, why?

Aim: use your laptop/desktop to access the CIS's servers `dimefox` (or `nutmeg`) and also access your uni's folder `H:`

Note:

- Your uni's folder `H:` is a good place to keep your files. You can access the folder from anywhere (with an Internet connection, of course).
- After accessing `dimefox`, you can edit and run C programs, submit assignments, and use various necessary tools for this course such as `make`, `valgrind`, `gdb`.

Tools: 3a. connect to dimefox/nutmeg, how?

- if you are not on uni's ground, make sure to run `VPN` first
- open Terminal (such as `minGW` terminal, or Mac's `Terminal`)
- on the `Terminal` run:

```
ssh login_name@dimefox.eng.unimelb.edu.au
```

then, your terminal will work with dimefox.

Try some unix commands such as:

- `ls` (list the content of current directory)
- `cd` (change current directory),
- `cp` (copy files, run `man cp` for details)
- `mkdir` (make new directory).

At home: use Google to learn some basic Unix commands.

Sample session on dimefox:

```
cd
ls
echo Hello Linux > hello.txt
ls
cp hello.txt hello_1.txt
more hello.txt
ls
man cp
emacs helloworld.c
```

Tools: 3b. Remote editing programs in H:

- *Aim*: Avoid time-consuming filecopy from your laptop to `H:`, be able to edit your uni's files remotely from anywhere.
- Tools: using `emacs`/`nano`/`vim` etc
- lazy use of `emacs` (run `emacs hello.c` on dimefox's terminal)
  - relatively simple and easy: just remember Ctrl-x Ctrl-s for "Save", Ctrl-x Ctrl-c for "Quit".
  - Cut/Copy/Delete a chunk of code inside the emacs window: Ctrl-Space to mark the start, arrows to expand, Esc-w for copy, Ctrl-w to cut, Ctrl-y to paste the chunk
  - Undo: Ctrl-x Ctrl-u
  - ESC a few times to exit from some mode/command if having troubles
  - how to use `emacs` more effectively and professionally: https://www.digitalocean.com/community/tutorials/how-to-use-the-emacs-editor-in-linux