

COMP10002 – Foundations of Algorithms

Welcome to the First Workshop!

The main workshop content starts at ...

While waiting, why not turn on the video and have a chat with friends?

A chance for us to communicate. A chance to have fun!

Note:

Algorithms are fun!

*The last lecture in W1: lec03-c in Lec Capture,
covering up to the end of chapter 2 textbook*

Greetings from Anh Vo (Vo Ngoc Anh)

- email avo@unimelb.edu.au with subject “COMP10002” or just “C102”

About You

- Have a chat, introduce, know about each other
- Tell about your interest, your funny things, your biggest programming/algorithim experience so far
- [optional] together, design an algorithm $\text{GCD}(a,b)$ to find the greatest common divisor of 2 natural numbers a and b
- **Zoom Poll:** our backgrounds

Keep Algorithms Fun

- Visit Canvas frequently and explore all the content there
- Use grok and discussion forum
- Be prepared for the workshops
- Be active in Zoom, start & follow discussions.
- Have fun, be engaged

A Problem to Solve

- What steps a computer must do to help us to solve an equation $ax+b=0$?
- Build a program, and then a C program for that.

C program: equation.c

Opening	#include <stdio.h> int main (int argc, char *argv[]) {
Declaring Inputting	double a, b, x; printf ("a, b = "); scanf("%lf%lf", &a, &b);
Computing Outputting	if (a != 0) { x= -b/a; printf("Solution x= %lf\n", x); } else { printf("a must be non-zero\n"); }
Closing	return 0; }

Think about the differences with a Python version!

Full C program: equation.c

Documentation	<pre>/* Solving equation ax + b = 0 Author: Anh Vo - anhvirl@gmail.com Last updated: 01 Aug 2020 */</pre>
Opening	<pre>#include <stdio.h> int main (int argc, char *argv[]) {</pre>
Declaring Inputting Computing Outputting	<pre> double a, b, x; ...</pre>
Closing	<pre> return 0; }</pre>

Why documentation and indentation? Programs are not just for computers to execute, but also for people to read, understand, and make changes.

C data types

- A variable must belong to a datatype
- `int`
- `float` and `double`
- `char`

The **char** data type

- a char represents a single character and is not a string
- example of constant char: 'A' '1' 'c' '+' *Note:* single quotes are used here, double quotes are used for strings.

```
1  char letter, digit= '1';
2  printf("Enter a letter:");
3  scanf("%c", &letter);

4  printf("letter= %c, ");
5  printf("next letter= %c\n", letter+1);

6  printf("digit= %c, ", digit);
7  printf("its ASCII value= %d\n", digit);
```

How scanf does it job?

Remember

- `printf` and `scanf` need a bit of remembrance
- Simple rules:

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

Quiz 1

If we execute the following fragment:

```
int i=8;    float x=9;  
scanf("%d%f", &i, &x);  
printf("i= %d, x= %.1f\n", i, x);
```

with the input stream (data from keyboard) of:

100.0 3.4

The output is:

A:

i= 100, x= 3.4

B:

i= 8, x= 9.0

C:

i= 100, x= 0.0

D:

i= 100 x= 3.4

Quiz 2

If we execute the following fragment:

```
int i=8; float x=9; char c= 'A';
scanf("%d%c%f", &i, &c, &x);
printf("i= %d, c= %c, x= %.1f\n", i, c, x);
```

with the input stream (data from keyboard) of:

100AB 3.4

The output is:

A:

i= 100, c= AB, x= 3.4

B:

i= 100, c= A, x= 9.0

C:

i= 100, c=A, x= 3.4

D:

i= 100, c= A, x= 3.4

Quiz 3

If we execute the following fragment:

```
int i;  char c;  float x;  
scanf("%d%c%f", &i, &c, &x);
```

with the input stream (data from keyboard) of:

100.1A200.2

Then, the value of **i**, **c**, and **x** become respectively:

A:

100 A 200.2

B:

100.1 A 200.2

C:

100 . 1

D:

(something else)

Quiz 4: **scanf** is actually a function, returning the number of data items it *successfully read*

If we execute the following fragment:

```
int i;  char c; float x; int n1, n2;  
n1= scanf("%f%d", &x, &i);  
n2= scanf("%c%d", &c, &i);  
printf("n1= %d, n2= %d, i= %d\n", n1, n2, i);
```

with the input stream (data from keyboard) of:

100.1A200.2

The output is:

A:

n1= 1, n2= 2, i= 200

B:

n1= 2, n2= 2, i= 200

C:

n1= 2, n2= 2, i= 2

D:

(something else or errors)

What needed to run C programs on my computer?

- You need to have some software installed:
 - A text editor such as [jEdit](#)
 - A unix-style terminal such as Mac's [Terminal](#), or Windows' [minGW](#), which include a C compiler such as [gcc](#)
- Then, you need to organize a neat working environment, including:
 - Make a dedicated directory (ie file folder) comp10002, and sub-folders Week02, Week03, ... under it
 - Arrange the app windows in your screen neatly so that you can [see](#) both editor and Terminal windows at the same time, and can easily switch to your browser when needed.

How to run my equation program on a computer?

1. Type, edit, and save it as `equation.c` using your text editor.
2. Using Terminal and:

 - a) Run `gcc` to compile `equation.c` to have an executable file with a name such as `equation` or `equation.exe`
 - b) Run the executable file
3. If `equation.c` has some errors, go back to step 1

Practice : help yourself, mates, and me. Each should have a neat working screen that includes a Zoom window

- **Zoom Poll:** how ready are we?
- How to work in your breakout room?
 - One person takes a lead, sharing the screen and do the typing etc.
 - Other people discuss and advise the leader on what to do while also doing the job in their own computer
 - Change the leader role after around 10 minutes or so
- Then, doing together:
 - practice some Unix commands if possible,
 - try equation.c together
 - Peer-programming exercises for 2.08 and 3.07

Practice 2: Peer-programming on Exercises 2.08 and 3.07

- The exercises are in grok, in case if you don't have the book yet
- You can use **jEdit+Terminal** or **grok**
- Work in breakout rooms (suggestions):
 - One person takes a lead, sharing the screen and do the typing etc.
 - Other people discuss and advise the leader on what to do while also doing the job in their own computer
 - Change the leader role after around 10 minutes or so

Lab implementation: Ex. 2.08 & 3.07

2.8: To convert from degrees Fahrenheit to degrees Celsius, you must first subtract 32, then multiply by 5/9. Write a program that undertakes this conversion.

3.7: Extend 2.8 program for additional units. Use F for Fahrenheit, C – Celsius, M – miles, K – kilometers, P – pounds, and G for kilograms. ($1M = 1.609K$; $1P = 0.454G$).

For example:

H:>converter

Enter a quantity: 100M

The distance 100.0 miles converts to 160.9 kilometers.

Remember

- Programming is fun! Stay active, stay happy, stay safe!
- Program, talk, ask friends, tutors and Mr Google
- Use Canvas, jEdit, minGW/Terminal, grok
- Programs: structure, editing, compiling, running, testing
- Variables: names, data types, values
- `if (?) {...} else {...}`
- Input with `scanf`, output with `printf`:
- Data types and formats for `printf`, `scanf`:

type	<code>int</code>	<code>float</code>	<code>double</code>	<code>char</code>	<code>string</code>
printf format	<code>%d</code>	<code>%f</code>	<code>%lf</code>	<code>%c</code>	<code>%s</code>
scanf format	<code>%d</code>	<code>%f</code>	<code>%lf</code>	<code>%c</code>	<code>%s</code>
scanf for <code>v</code>	<code>&v</code>	<code>&v</code>	<code>&v</code>	<code>&v</code>	<code>v</code>