COMP1511 Programming Fundamentals Lecture 1

The Beginning

Today's Lecture

- Important details about the lecture format
- Who to contact if you need help
- How COMP1511 works
- How to get help when you need it
- What is programming?
- Working in Linux
- A first look at C

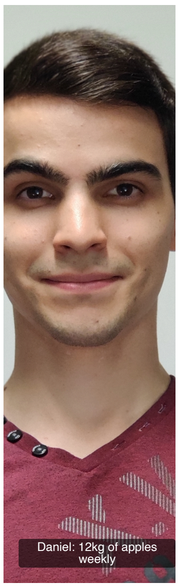
Who am I?

Really, who am I?

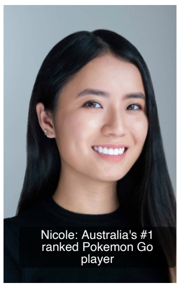
- Software Engineer
- Tennis lover
- Coffee aficionado
- Favourite languages (right now): Typescript, Python, C!

Course admins!

Sofia: Multi-Award winning travel documentarian


Daniel: 12kg of apples weekly

Tammy: "Happiest person of the year" 5 times champ


Nicole: Australia's #1 ranked Pokemon Go player

We have Lecture Moderators!

And we can't wait to meet you all <3

Let's take 5 mins to introduce yourself to your neighbours (physical or virtual)

Important Resources

The Course page: https://cgi.cse.unsw.edu.au/~cs1511/24T1/

- All important course information is on this page
- We don't use Moodle!
- ***New*** Course Outline has moved!

Contacts

- Administration issues: cs1511@unsw.edu.au
- Enrolment issues: https://nucleus.unsw.edu.au/en/contact-us
- Equitable Learning Plan: jake.renzella@unsw.edu.au

Getting help with Programming

The Forum

- https://edstem.org/au/
- Post any content-related questions here!

Details on Help Sessions, Revision Classes, and more coming soon

Course Format

- Weekly lectures
- Weekly tutelabs
- 2x Major Assignments
- 1x Final Exam

Lecture Format

- **Monday**: 11:00 - 13:00 in Ainsworth G03
- **Wednesday**: 11:00 - 13:00 in Ainsworth G03
- Youtube Live, or come alone to the theatre

Tutorials/Labs

- Tutelabs are scheduled as a single 3-hour block
- Go further into topics we cover in the lecture
- hands-on and practical!

Jake's Major Assignment pro-tips

– Start it as early as possible
– Don't plagiarise, we'll get ya
– Assignment 1 - 20% (Monday 8pm Week 7)
– Assignment 2 - 25% (Friday 8pm Week 10)

What to do if you can't COMP1511

Feeling unwell? Need to travel back home for an emergency? Dog ate your assignment?

– **special considerations**: https://student.unsw.edu.au/special-consideration

Code of Conduct
We are here to learn

Plagiarism, Contract Cheating, ChatGPT, My Neighbour worked on a C compiler

Quick break

COMP1511

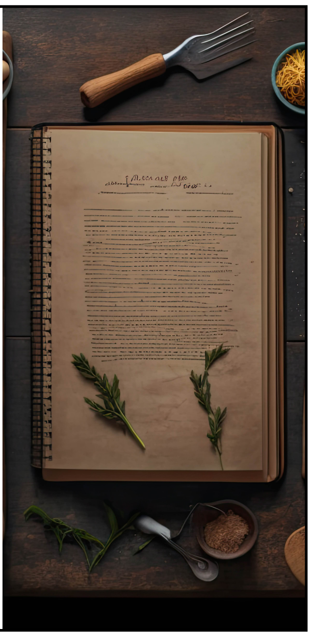
Computers, compilers, programs, C, operating systems, UNIX, Linux, Terminal, Files, functions, oh my...

What is a computer?

What is Programming?
Producing a set of instructions and/or data to achieve a task

Writing a program is like writing a recipe

- You provide the steps required to solve the task
- The computer executes the program, completing it step by step
- Any mistakes in your recipe will alter the final product (and probably ruin it!)



How do these *programs* run?

- Computers are made up of many programs, many executing at the same time!
- Imagine if your kitchen was used to prepare tens, hundreds of recipes all at once

We need a head chef (operating system)!

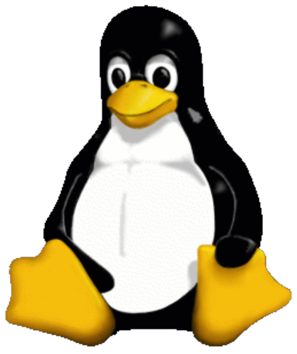
An Operating System is the interface between the user and the computer hardware

Operating Systems:

- Execute user programs
- Make sure programs do what they're supposed to
- Schedules access to limited resources (hardware)
- Make the computer system convenient to use

The Linux Operating System

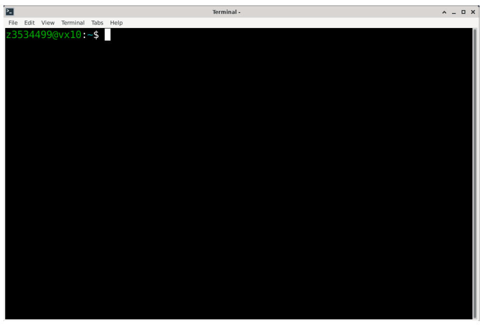
- A UNIX-based operating system
- Open-Source, reliable, lightweight and secure



How do programmers interact with a computer?

The Terminal

- Send text-based commands to our shell
- Terminal handles user input, rendering shell output



```
z3534499@vx10:~$
```

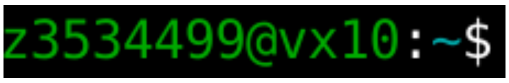
The Shell

The shell, (bash, zsh) is a program that executes commands, and has its own syntax. It returns output which the terminal can display

```
z3534499@vx10:~$
```

The Prompt

The prompt is controlled by the shell, and is the line of text which displays some information



```
z3534499@vx10:~$
```

How do I use this thing?

Important terminal commands

- `ls` : Lists all the files in the current directory:
- `mkdir <dir name>` Makes a new directory called directoryName:
- `cd <dir name>` : Changes the current directory to directoryName:
- `cd ..` : Moves up one level of directories (one folder level):
- `pwd` : Tells you where you are in the directory structure at the moment:

File operations

- `cp <source> <destination>` : Copy a file from the source to the destination
- `mv <source> <destination>` : Move a file from the source to the destination (can also be used to rename)

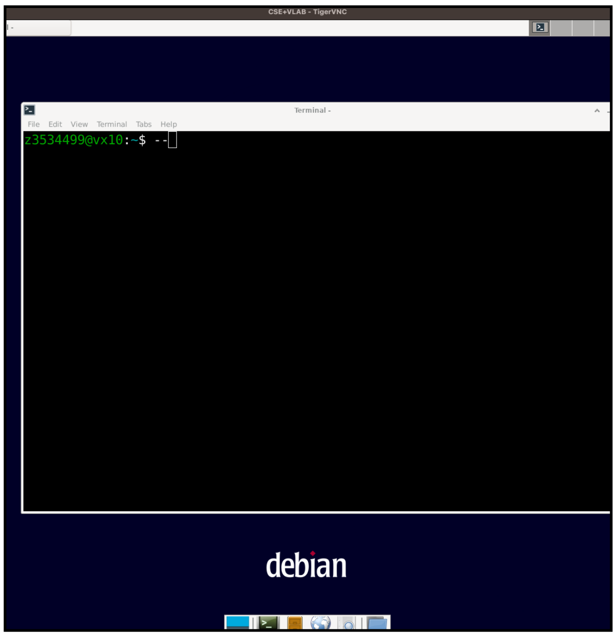
`rm filename` : Remove a file (delete)

The -r tag can be added to cp or rm commands to recursively go through a directory and perform the command on all the files

```
cp -r <source> <desitnation>
```

But Jake! I don't have a Linux computer!!!

Don't worry! We have one for you <3



```
z3534499@vx10:~$ --
```

Let's get set up together

- Log into VLAB
- Open the Terminal
- Run `1511 setup`

Now we have the tools, so can we write out first program yet?

- Computers execute *precise* instructions described in a *native language* to computers
- This language is not easy for us to understand:

```
00000000: 0100 0000 0000 0000
0000 0000 0000 0000
00000010: 1011 0110 0000 0000
0000 0000 0000 0010
00000020: 0000 0100 0110 0000
1001 0000 0000 0000
```

Computers need precision!

So machine code is too precise...

Why can't we just say "Hey computer! Add two numbers together!

Programming

Precise enough to be translated to machine code

Simple enough that a human can (sometimes) understand it.

A *shared* language

Programming in C
Why C?

And what a beautiful language

```
#include <stdio.h>

int main(void)
{
    printf("Hello
world");
    return 0;
}
```

Demo (follow along if you can)

1. Create a .c file using the Terminal
2. Write our hello world program using VSCode
3. Save it

Let's break it down

```
// loads the standard
input/output library
#include <stdio.h>

// the main function, the
starting point of our program
int main(void) {
    // prints the string to the
standard output
    printf("Hello world");

    // returns 0 to the operating
system
    return 0;
}
```

`#include <stdio>`

- Some tasks are so common, that it would be wasteful to have to write them every time
- Common code is available for us, in the standard C library
- We need to tell the compiler which libraries to use

`#include <stdio>`

- In this case, we want the Standard Input Output Library

  This allows us to make text appear on the terminal

  Almost every C program you will write in this course will have this line

The main block

```
int main(void) {
    ...
}
```

- The **main function**
- Every C program must have 1 main function! It's where our program starts!
- Program runs in sequence, line-by-line starting inside the main block

Blocks of code

```
{
    ...
}
```

Between each `{` and `}` are a block, or group of instructions.

Blocks are very important! They are how we organise code

The `printf`

```
{
    printf("Hello
world!");
}
```

printf() makes text appear on the screen. It is a function from stdio.h which we included.

`return 0`

return is a C keyword that tells the computer that we are now delivering the output of a function.

A main function that returns 0 is signifying a correct outcome of the program back to the operating system

Comments!

– We place “comments” in programs explain to our future selves or our colleagues what we intended for this code

`//` in front of a line makes it a comment`

If we use `/*` and `*/` everything between them will be comments

The compiler will ignore comments, so they can be anything you want really!

Compiling

Remember, C is a shared language, so we can be productive

Computers can't understand C

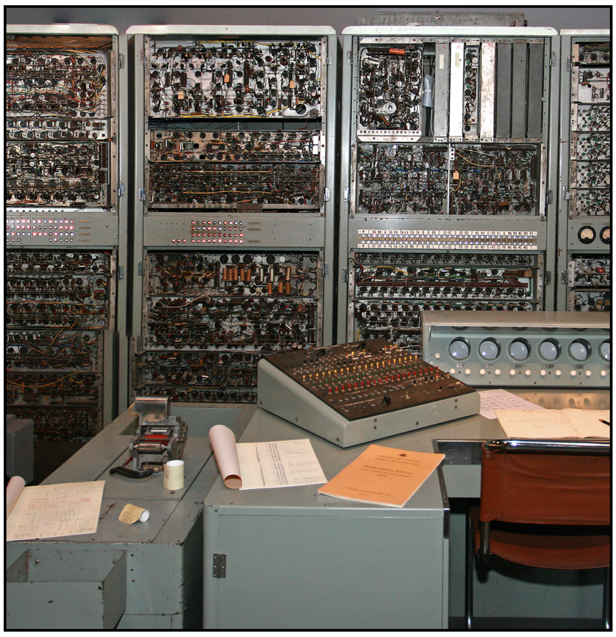
We need to turn our C code into machine code using a compiler

Compilers are programs

That turn code into machine code.

```
dcc program.c -o
helloWorld
./helloWorld
```

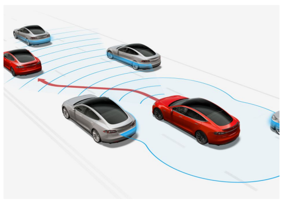
This compiles a C program into an executable called helloWorld, and runs it

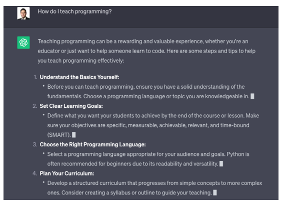


Modern technology has changed a lot

But what hasn't changed

Is computers executing instructions described by humans

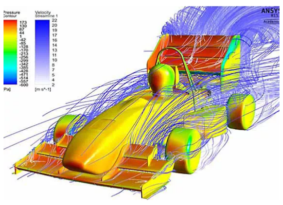




How do I teach programming?

Teaching programming can be a rewarding and valuable experience, whether you're an educator or just want to help someone learn to code. Here are some steps and tips to help you teach programming effectively:

1. **Understand the Basics Yourself:**
   - Before you can teach programming, ensure you have a solid understanding of the fundamentals. Choose a programming language or topic you are knowledgeable in.
2. **Set Clear Learning Goals:**
   - Define what you want your students to achieve by the end of the course or lesson. Make sure your objectives are specific, measurable, achievable, relevant, and time-bound (SMART).
3. **Choose the Right Programming Language:**
   - Select a programming language appropriate for your audience and goals. Python is often recommended for beginners due to its readability and versatility.
4. **Plan Your Curriculum:**
   - Develop a structured curriculum that progresses from simple concepts to more complex ones. Consider creating a syllabus or outline to guide your teaching.



What will you build?