CSC209 Lecture 1: Introduction

David Liu, Department of Computer Science, University of Toronto

*Navigation tip for web slides: press **?** to see keyboard navigation controls.*

Getting to know each other

Say hi to your neighbours

Some things to talk about:

- What other classes are you taking?
- Favourite spot on campus so far?
- Clubs you might be interested in joining?

Who is David Liu?

Born in Ottawa, grew up in Toronto

Studied math, CS, and education

Teaching-stream faculty member in CS

Call me David or Professor Liu

1 22

News **Finally some appreciation for Prof. David Liu**

submitted 2 hours ago by [redacted]

3 comments share save hide report

About CSC209

Course Information

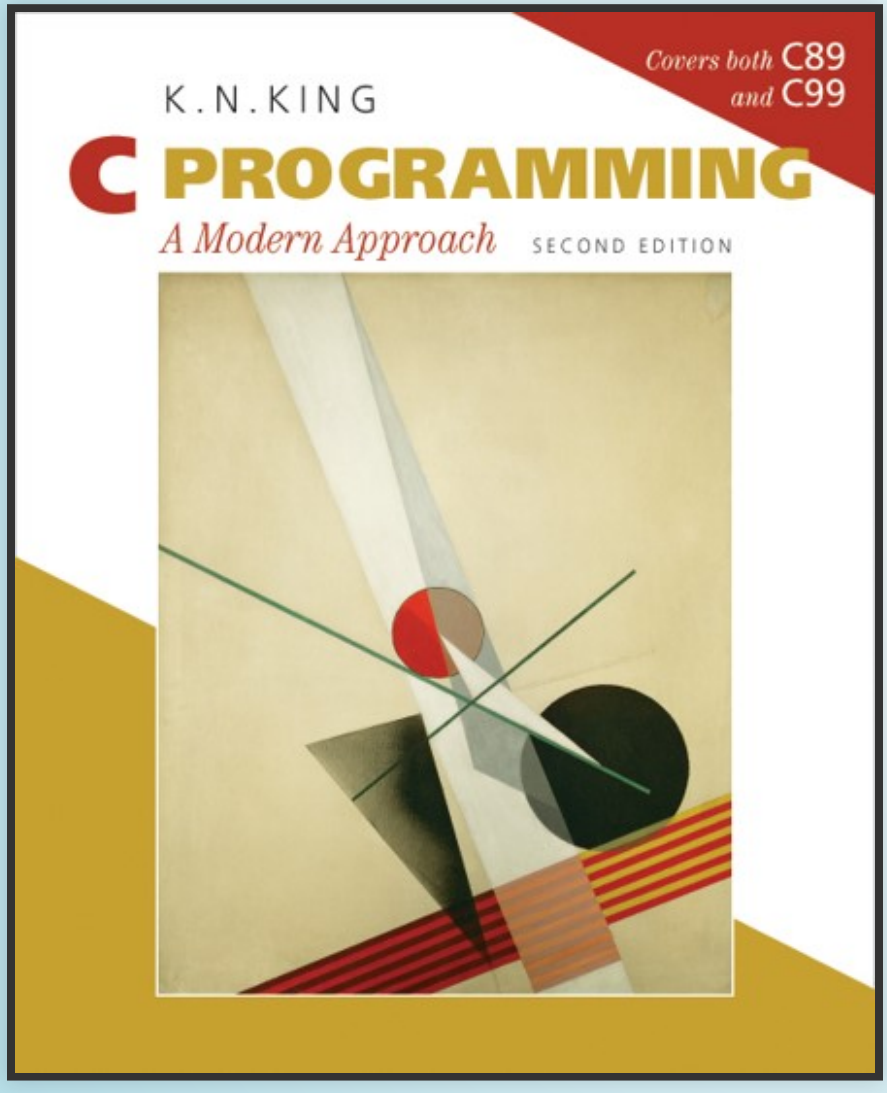
- All important information will be on Quercus
  - https://q.utoronto.ca/courses/204484/
- Syllabus: deadlines, course policies, prerequisite requirements
- Lectures & Labs: schedule, notes, worksheets, etc.
- Announcements: you are responsible for reading all announcements!

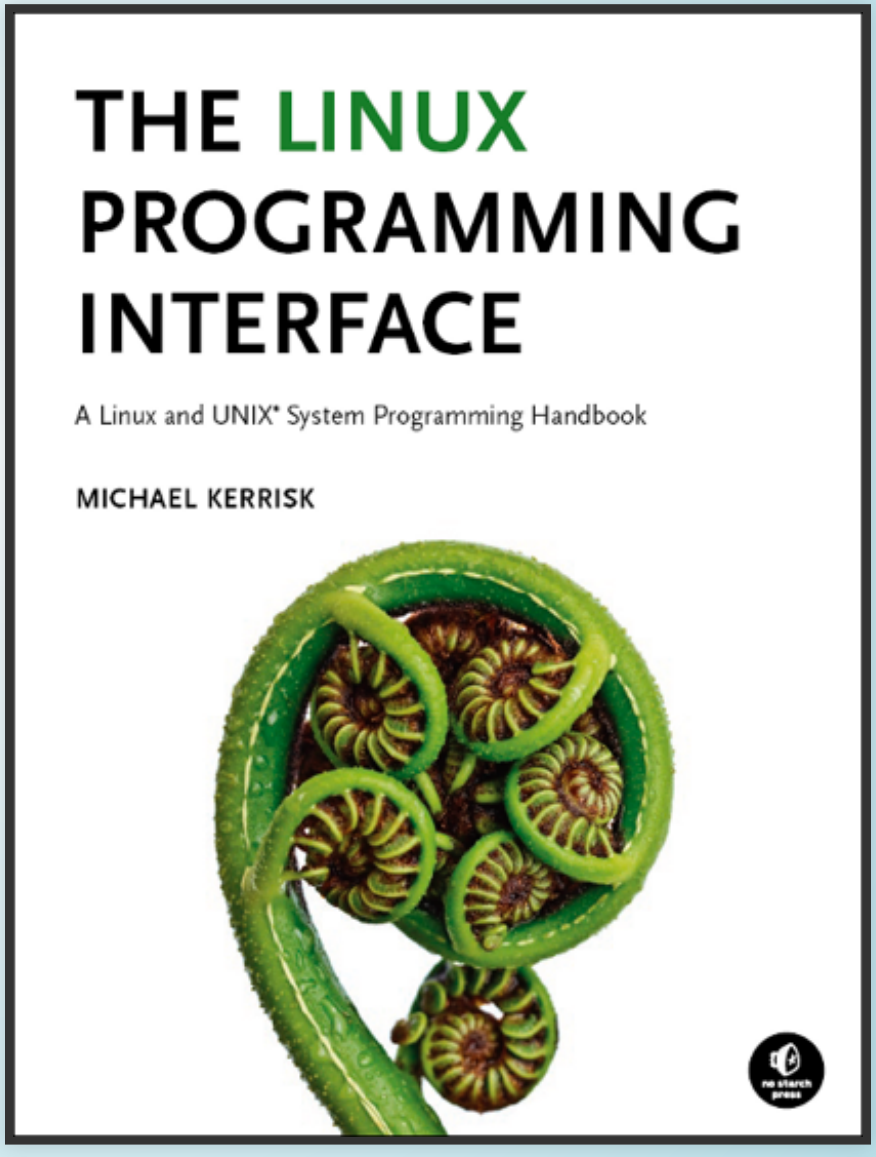
Lectures

CSC209 uses an inverted classroom model.

- (Graded) preparation before class
  - videos & exercises
- Hands-on activities in class
  - should be ready to do some programming during class

Course textbooks





Assessments

Work	Weight	Deadline
Lecture Preparation (PCRS)	5% (best 10 of 11)	Tuesdays before 10:00am (weeks 2 - 12)
Lab Exercises	10% (best 10 of 11)	Fridays before 6:30pm (weeks 1 - 11)
A1	5%	Thursday 26 January before 6:30pm
A2	10%	Wednesday 15 February before 6:30pm
Midterm Test	10%	Tues 28 February during class time
A3	10%	Wednesday 15 March before 6:30pm
A4	9%	Wednesday 5 April before 6:30pm
Research Surveys	1% (0.1% x 10 surveys)	Fridays before 6:30 pm (weeks 2 - 11)
Final exam	40%	Minimum grade of 40% required to pass this course

Assignments

The **four** assignments will give you opportunity to apply and extend your learning to solve new programs.

- Fairly time-intensive, so start early!
- Lots of programming
  - All code must work on teach.cs to receive full marks.
  - Code that does not compile on teach.cs will get **0**.
- You'll be using the `git` version control system to manage and submit your assignments
- See Course Syllabus for late penalties and remark policies.

Weekly lab exercises

We'll be posting a weekly lab exercise to help reinforce course concepts regularly.

- Due Fridays 6:30pm
  - No late submissions accepted
- Your tutorials are designed to help you complete the labs with the support of your TA
  - See tutorial information on Quercus
- First lab is due **this Friday!**

Software installation

- Follow the Software Setup instructions on Quercus
  - Ask questions on Piazza if you get stuck
- Choose a **text editor** to write code for this course
  - e.g., VSCode, Sublime Text, vim, emacs, nano
- Learn to connect remotely to teach.cs and compile, edit, run programs
  - Learning to use `ssh` from the command-line will really pay off
  - https://www.teach.cs.toronto.edu/using_cdf.html

Academic integrity

> *"The work you submit must be your own, done without participation by others. It is an academic offence to hand in anything written by someone else without acknowledgement."*

- You are hurting your friend when you give them a copy of your assignment.
- You are hurting your friend when you ask them for a copy of their assignment.

Academic integrity, continued

It is an academic offense to:

- copy parts or all of another student's assignment
- include code from books, websites, other courses without attribution
- get someone else to do substantial parts of your assignment
- give someone else your solution
- use code generated by an AI (e.g., ChatGPT)

It is not an academic offense to:

- help each other understand documentation or example code
- refer to course materials (e.g. lectures, lab exercises)
- provide help debugging (but be careful)

Communication

Piazza:

- use first for non-personal communication
- informative subject lines help

Email:

- csc209-2023-01@cs.toronto.edu
- use for personal communication, such as a request for special consideration

Office hours:

- Held in person (see Course Syllabus for schedule)
- Use for questions about course material and help with course work

What is CSC209 About?

“Software Tools and Systems Programming”

Software tools:

- Efficiently use the **Unix command line**
- Understand what the **shell** is, and write basic **shell scripts**
- Use the **`make`** tool by creating `Makefile`s

Systems programming:

- C programming
- files
- processes
- process communication (e.g. signals, sockets)

Unix Principles

1. **Do one basic thing well**
    - with some basic variations
2. **Simple input formats**
    - plain text
    - don’t require interactive input
    - stdin to stdout/stderr
3. **Simple output format**
    - plain text
    - expected to be input to another tool

The Unix Command Line

Demo of some tools

- `ls`
- `wc`
- `sort`

Shells

A **shell** is a program that acts as an interface between a human and your computer's operating system.

```
$ wc hello.c
```

- The $ is a shell **prompt**.
- The text `wc hello.c` is a command for the shell.
- The first word, `wc`, is the name of an executable file (program) to run
- The remaining text (`hello.c`) is an argument to the program.
- The shell:
  - Finds the executable program on your computer
  - Interprets the arguments
  - Runs the program with the given arguments

Shell commands

Note: shells typically have some built-in commands that aren't executable programs. E.g., `echo`, `test`.

We'll do some shell programming at the end of the course.

Standard output, standard input, and pipes

Standard output: the (default) place where programs print text

Standard input: the (default) place where programs read in text (user input)

It is possible to **redirect** standard input and output to files using < and >.

It is possible to **pipe** the standard output of one program into the standard input of another program.

Basic tools to learn

Working with the filesystem:

- `cd`, `ls`
- `mkdir`, `cp`, `mv`, `rm`
- `chmod`

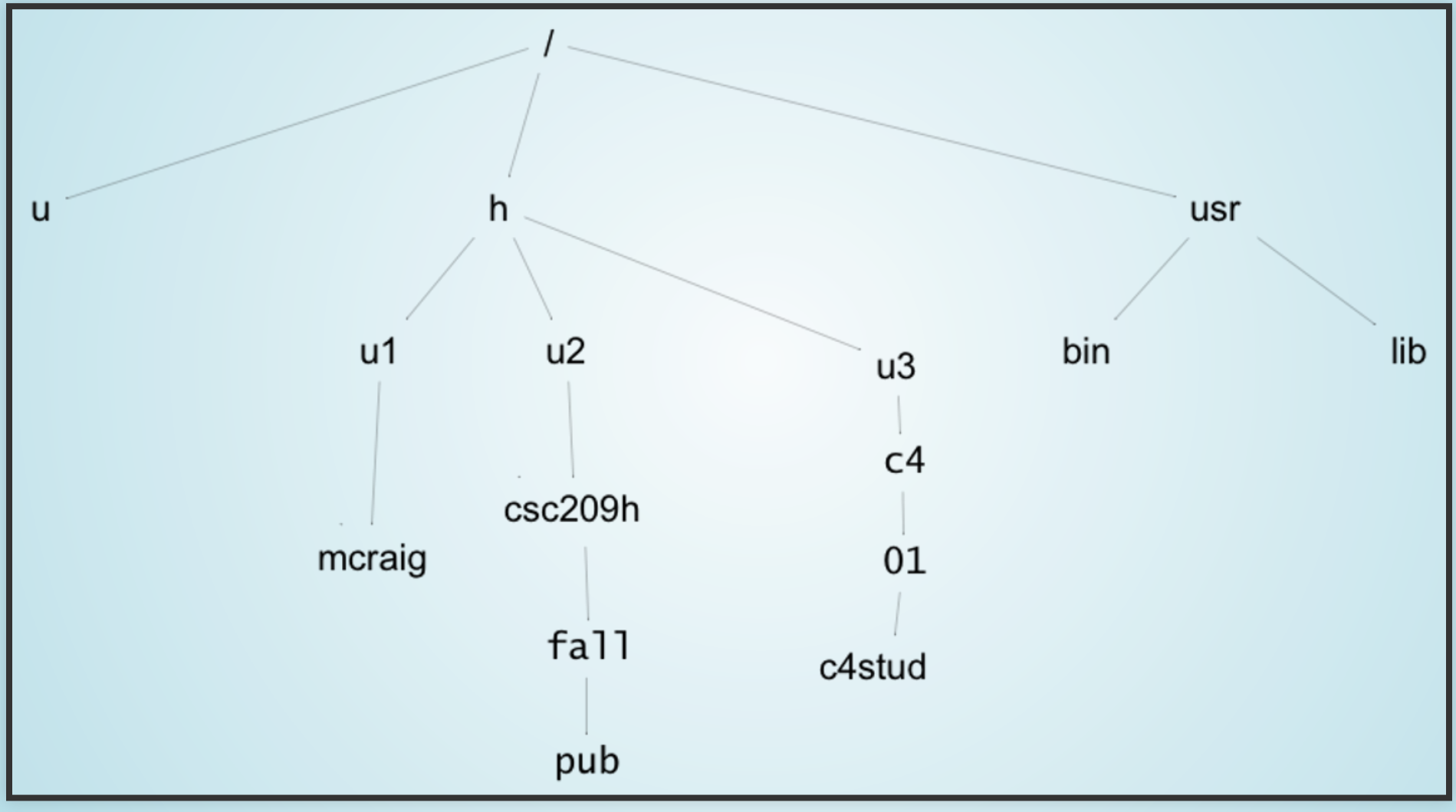
Inspecting files:

- `cat`, `head`, `tail`
- `wc`
- `grep`

Don't memorize, look it up! (**Demo**: `man`)

An introduction to the filesystem

The filesystem as a tree



Some details

- An **inode** is a data structure that contains information about a file
  - Metadata (e.g. owner, creation time)
  - Where the file contents are stored on disk
- A **directory** is a special type of file that contains **directory entries**.
  - A **directory entry** is a mapping from file name to inode.
  - Directories can also contain other directories, creating a directory hierarchy
- The **root directory** is the top of the hierarchy
  - The name of the root directory is simply `/`

Inspecting directory contents with `ls`

Demo!

Permissions

Since multiple users can share the same filesystem, we need a way to restrict file/directory access to particular users.

Three levels of user access:

- **user** who owns the file
- **group (of users)** that the file is associated with
- **other** users

Three types of access:

Type	**For a file**	**For a directory**
read	View contents of file	See contents of directory
write	Edit/delete file	Add/remove files from directory
e**x**ecute	Execute a file (as a program)	“Pass through” directory to access subdirectories

Inspecting permissions with `ls -l`

```
-rwx--x--x 1 liudavid instrs 16880 Jan  4 13:10 hello
-rw------- 1 liudavid instrs   166 Jan  4 12:21 hello.c
```

Changing permissions with `chmod`

```
$ chmod MODE FILE ...
```

`MODE` can be:

- Three numbers between 0 and 7, for user/group/other
  - Each number represents three bits $(b_0 b_1 b_2)_2$ where each bit is `r`/`w`/`x`
  - Example: `chmod 741 hello`
- `<u/g/o><+/-><r/w/x>`
  - Example: `chmod u+x hello` or `chmod g-w hello`

Creating programs

Consider Python

In Python, we write our programs as code in `.py` files, e.g.

```
# hello.py
if __name__ == '__main__':
    print('David is cool')
```

How do we run this program in the shell? (**Demo!**)

```
$ python3 hello.py
```

The program being run is `python3`, the Python interpreter.

It takes as an argument a path to a file (`hello.py`), which it then executes.

Now, in C (Demo!)

Suppose we have a file `hello.c`, written in the C programming language. How do we "run" this file?

First, we **compile** the code into an executable file.

```
$ gcc -Wall -g -std=gnu99 -o hello hello.c
```

Then, we can **execute** that new executable directly.

```
$ ./hello
```

CSC209 programming workflow

Throughout CSC209, you'll be following this workflow:

1. Write C code in `.c` files
2. Compile the code into an executable file
3. Run that executable file (often with some arguments)