CPSC 201: Midterm 2 Review

Spring 2025 😄

Hosted by

Deanna DeCarlo and Miranda Selin

Midterm Exam 2

Tuesday, April 1st at 7:00 PM

in Davies Auditorium (SAS: Becton C031)

Agenda

1. Exam resources and topics
2. Boolean expressions
3. Gates and circuits
4. UNIX Principles 3 & 4
5. Tail recursion
6. Q&A (time permitting)

Agenda

1. **Exam resources and topics**
2. Boolean expressions
3. Gates and circuits
4. UNIX Principles 3 & 4
5. Tail recursion
6. Q&A (time permitting)

General Class Resources

- Course website
  - Lecture notes
  - UNIX Guide
  - Jupyter Notebook (best resource IMO)
- Practice materials for Midterm 2
  - Practice exam and solutions (Ignore problems 3, 4, 5a, and 5e)
  - UNIX transcript and solutions
- Racket guide
- UNIX tutorial (more info in a following section)
  - ssh into the Zoo; then in your home folder, type the following command:

    ```
    python3 /c/cs201/www/unixtutorial.py
    ```
- Ed Discussion
- cs201help@cs.yale.edu (automatically emails Professor Slade and the ULAs)
- Office hours

On the exam: ✅

- Boolean functions
- Gates/circuits
- Tail recursion
- UNIX Principles 1 - 4

Not on the exam: 🙏

- The halting problem
- Writing Turing machines
- TC-201

Relevant Topics from Midterm 1

1. General Racket functions
2. Recursion
3. UNIX Principles 1 & 2

See Owen’s Midterm Exam 1 review slides here.

Agenda

1. Exam resources and topics
2. **Boolean expressions**
3. Gates and circuits
4. UNIX Principles 3 & 4
5. Tail recursion
6. Q&A (time permitting)

Key Ideas

- Truth tables
- Operations:
    - and (•)
    - or (+)
    - not (‘)
- Sum of products algorithm
    - (optional) Boolean algebra (NOT ON EXAM)

What is a Truth Table?

- Table of all the possible truth values returned by a boolean expression from all the possible inputs
- Number of possible sets of inputs doubles for every variable (every variable can be either 0 or 1)
- You can always find a corresponding boolean expression for every truth table using the sum of products method

Sum of Products Method

1. Isolate rows where the output is 1/true and ignore rows where output is 0/false
2. For each true row, write a sub-expression that takes the AND of all the variables together while taking the NOT for any variable whose input value is 0
3. Take the OR of all the sub-expressions together to produce a final expression
4. (Optional) Simplify the expression if possible

Sum of Products

How I approach truth tables:

x	y	z	f(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Sum of Products

x	y	z	f(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

How I approach truth tables:

1. Find all the true values in the output column

Sum of Products

x	y	z	f(x,y,z)	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	x’•y•z
1	0	0	0	
1	0	1	1	x•y’•z
1	1	0	1	x•y•z’
1	1	1	1	x•y•z

How I approach truth tables:

1. Find all the true values in the output column
2. Write Boolean expressions for the corresponding rows

Sum of Products

x	y	z	f(x,y,z)	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$x'\bullet y\bullet z$
1	0	0	0	
1	0	1	1	$x\bullet y'\bullet z$
1	1	0	1	$x\bullet y\bullet z'$
1	1	1	1	$x\bullet y\bullet z$

How I approach truth tables:

1. Find all the true values in the output column
2. Write Boolean expressions for the corresponding rows
3. Add these expressions together to get your final sum of products:

$$(x'\bullet y\bullet z)+(x\bullet y'\bullet z)+(x\bullet y\bullet z')+(x\bullet y\bullet z)$$

Equivalently:

$$(x\bullet y)+(y\bullet z)+(x\bullet z)$$

Sum of Products Practice

x	y	z	f(x,y,z)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Sum of Products Practice

x	y	z	f(x,y,z)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Solution:

$(x'\bullet y'\bullet z')+(x'\bullet y\bullet z')+(x\bullet y'\bullet z')+(x\bullet y'\bullet z)$

OR

$(x'\bullet z')+(x\bullet y')$

Boolean Algebra (NOT ON EXAM)

- Sum of products is reliable but not always efficient
  - Writing a sub-expression for every valid truth table row can get messy if there are many true rows
- Luckily boolean algebra has many laws and rules that work similar to normal algebra to help reduce large expressions to shorter, equivalent statements
- Order of operations: NOT, AND, OR
- NOTE: The next 4 slides are NOT tested on the exam but knowing it makes life much easier (especially for circuits)

Useful Laws to Know (NOT ON EXAM)

Annulment Law:

$X \bullet 0 = 0 \qquad X + 1 = 1$

Identity Law:

$X \bullet 1 = X \qquad X + 0 = X$

Idempotent Law:

$X \bullet X = X \qquad X + X = X$

Complement Law:

$X \bullet X' = 0 \qquad X + X' = 1$

Double Negation Law:

$(X')' = X$

XOR Gate:

$X \text{ XOR } Y = X'Y + XY'$

Useful Laws to Know (NOT ON EXAM)

Commutative Law:

$$X \bullet Y = Y \bullet X$$

$$X + Y = Y + X$$

Distributive Law:

$$X \bullet (Y + Z) = XY + XZ$$

$$X + (Y \bullet Z) = (X + Y) \bullet (X + Z)$$

Associative Law:

$$X \bullet (Y \bullet Z) = (X \bullet Y) \bullet Z$$

$$X + (Y + Z) = (X + Y) + Z$$

Redundancy Law:

$$(X + Y') \bullet Y = X \bullet Y$$

$$(X \bullet Y') + Y = X + Y$$

Reducing an Expression Example

$x'yz + xy'z + xyz$	
$x'yz + xy'z + xyz = (x' + x)(yz) + xy'z$	Distributive Law
$(x' + x)(yz) + xy'z = 1(yz) + xy'z$	Complement Law
$1(yz) + xy'z = yz + xy'z$	Identity Law
$yz + xy'z = z(y + xy')$	Distributive Law
$z(y + y'x) = z(x + y)$	Redundancy Law
$z(x + y)$ OR $zx + zy$	Final Answer

Extra Resources for Boolean Expressions

Truth Table Generator: gives a truth table for a given boolean/logic expression

Boolean Algebra Calculator: reduce a given boolean expression to its simplest form with steps

Boolean Algebra Laws: short table to reference general boolean algebra rules and laws

You don’t always have to use sum-of-products

Find an expression for each of the following: f, g, and h

x	y	z	f(x,y,z)	g(x,y,z)	h(x,y,z)
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	1	0	1	1

You don’t always have to use sum-of-products

Find an expression for each of the following: f, g, and h

x	y	z	y’	x + yz	x + y + z
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	1	0	1	1

Agenda

1. Exam resources and topics
2. Boolean expressions
3. **Gates and circuits**
4. UNIX Principles 3 & 4
5. Tail recursion
6. Q&A (time permitting)

Circuits

Circuits are basically Boolean logic expressions with an additional dimension of time (aka gate delays).

In **combinational circuits** (no “loops”) that additional dimension only matters for figuring out how long it will take the circuit to produce its final output.

In **sequential circuits**, wherein the output is determined by both the current input and prior states of the circuit, time really does matter.

Combinational vs. Sequential Circuits

Combinational circuits:

- No loops of wires and gates
- In a combinational circuit, the eventual final outputs of the circuit are completely determined by the values of the circuit inputs

Examples:

- Full-adder
- Half-adder

Sequential circuits:

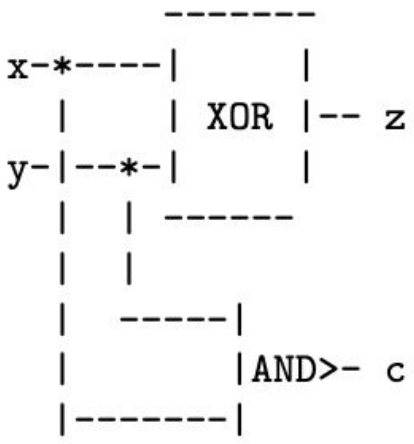
- “Loop-y”
- The outputs of a sequential circuit may depend on both the inputs and the past values of the wires of the circuit

Examples:

- “Garden of Eden” circuit
- NAND latch

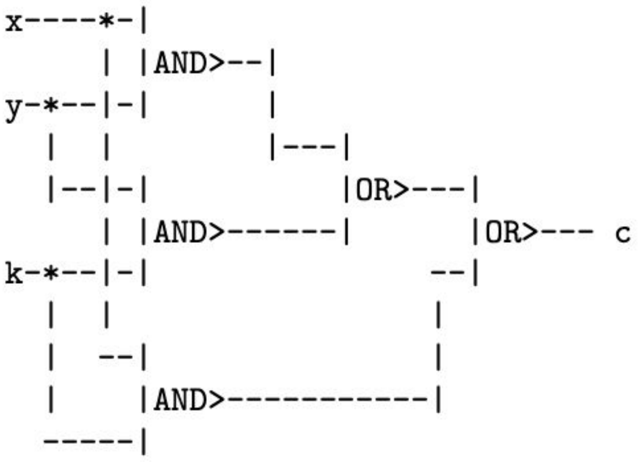
Combinational Circuits

Half-Adder



```
       -------
x-*----|     |
  |    | XOR |-- z
y-|--*-|     |
  |  | ------
  |  |
  |  -----|
  |       |AND>- c
  |-------|
```

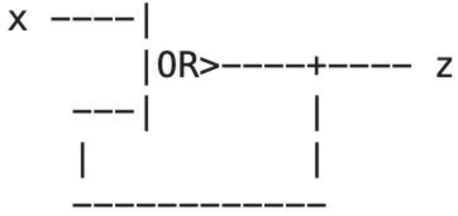
Full-Adder



```
x----*-|
     | |AND>--|
y-*--|-|      |
  |  |        |---|
  |--|-|          |OR>---|
     | |AND>------|      |OR>--- c
k-*--|-|               --|
  |  |                 |
  |  --|               |
  |    |AND>-----------|
  -----|
```

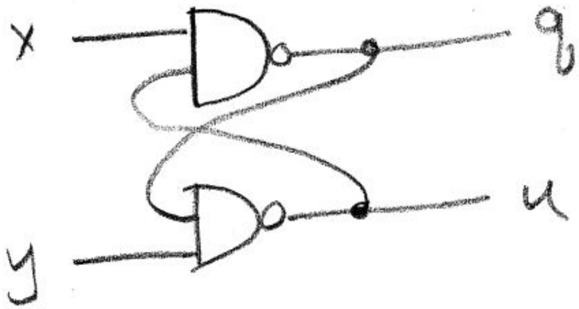
Sequential Circuits

Garden of Eden



```
x ----|
      |OR>----+---- z
   ---|       |
   |          |
   ------------
```

Full-Adder



Practice designing circuits (from sample exam)

Draw a combinational circuit with

- inputs r, a, b
- outputs x, y

that computes the following:

- if r = 0 then x = a and y = b
- if r = 1 then x = b and y = a

You may use NOT and 2-input AND, OR, XOR. Make sure you label the input and output wires of your circuit, and label your NOT, AND, OR and XOR gates (which can be represented by rectangles with the correct labels.)

Step 1: Draw out the corresponding truth table

r	a	b	x	y
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Step 1: Draw out the corresponding truth table

r	a	b	x	y
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Recall: if r = 0 then x = a and y = b

Step 1: Draw out the corresponding truth table

r	a	b	x	y
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

Recall: if r = 1 then x = b and y = a

Step 2: use sum-of-products algorithm to find expression for x

r	a	b	x	y
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

r’a

rb

$$x = r'a + rb$$

Step 3: use sum-of-products algorithm to find expression for y

r	a	b	x	y
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

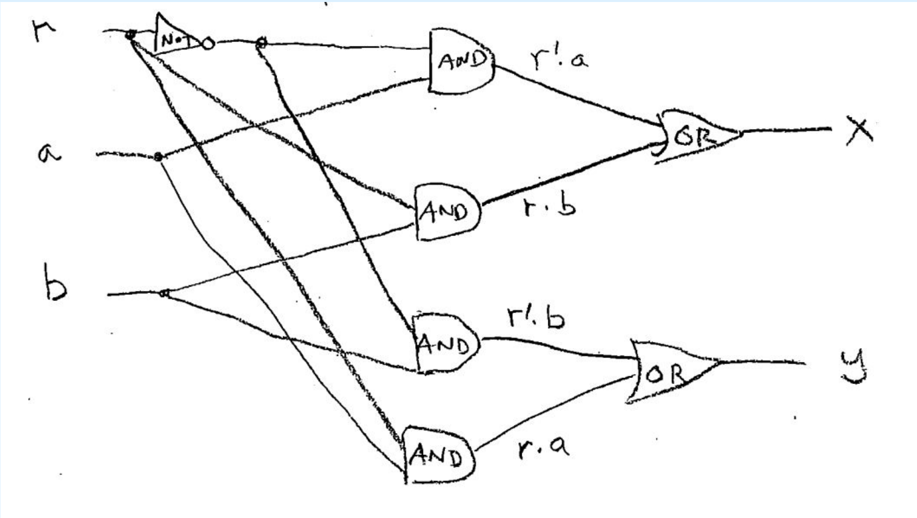
r'b

ra

$$y = r'b + ra$$

Step 4: translate your expressions for x and y into a circuit!

```
x = r'a + rb
y = r'b + ra
```



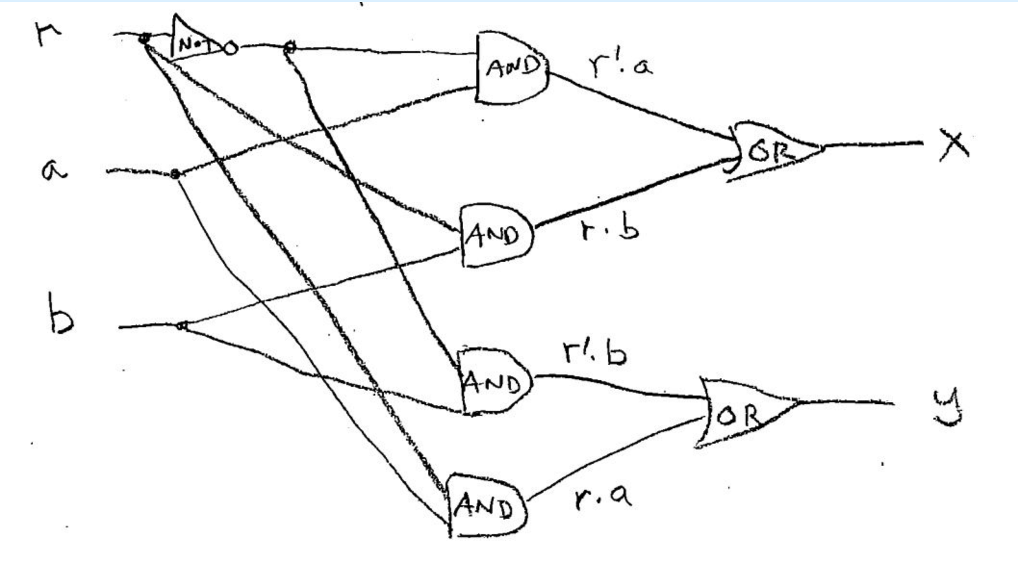
Gate Delays

- For every combinational circuit, there is a set number of gate delays before the circuit outputs the final values
- The total gate delays required does NOT depend on the total number of gates
  - This is because gates can run in parallel
- Instead it depends on the number of gates in the **longest path** between any input wire and any output wire

Gate Delays

How many gate delays are required to solve the final output?

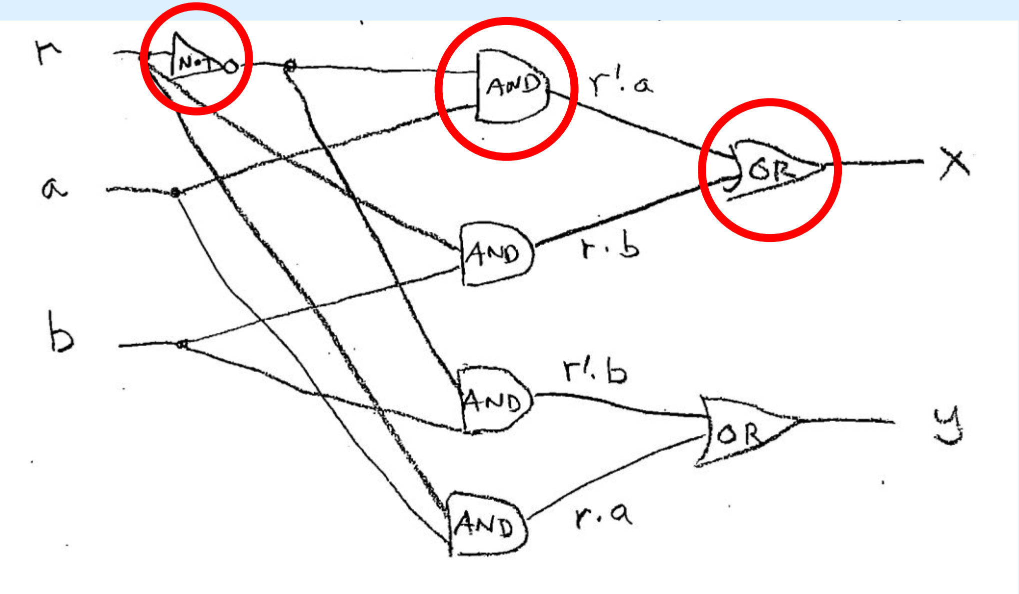
Remember to find the longest path first



Gate Delays

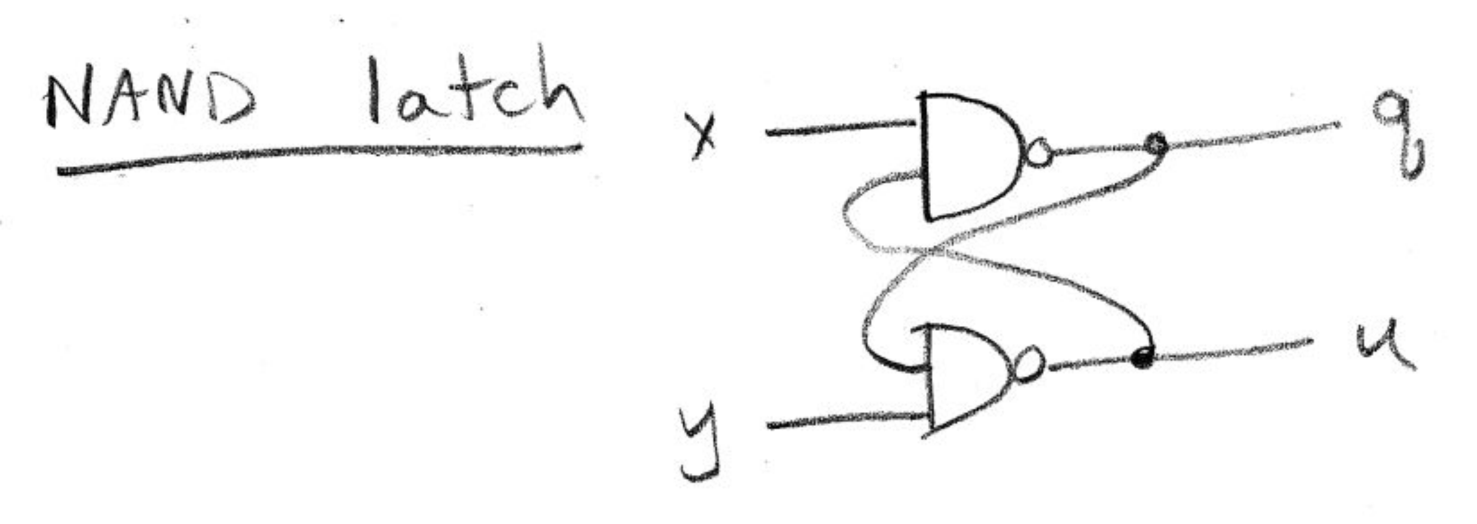
Answer: 3 gate delays

Longest path is from r to either x or y and passes through a NOT, AND, and OR gate



Good circuits to know

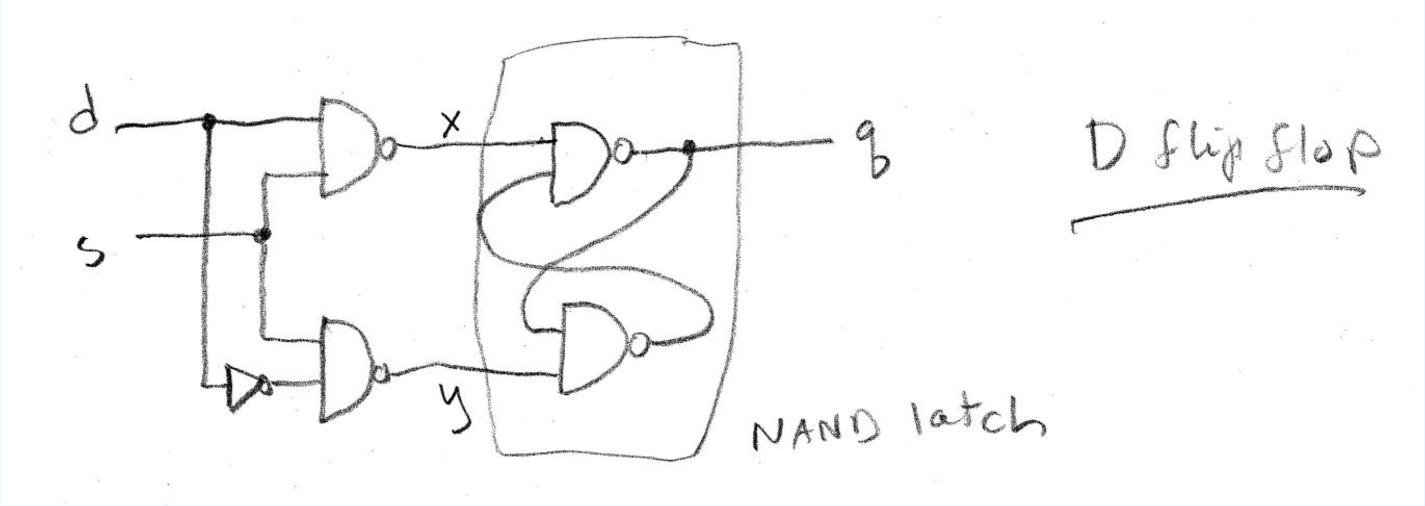
- D flip-flop and NAND latch
- Half-adder, full-adder, and ripple-carry adder



NAND Latches/D Flip-Flop

Your key takeaway should be that a D Flip Flop enables the storage of information through the use of a NAND latch.

How?



NAND Latches/D Flip-Flop

Your key takeaway should be that this type of circuitry enables the storage of information.
How?

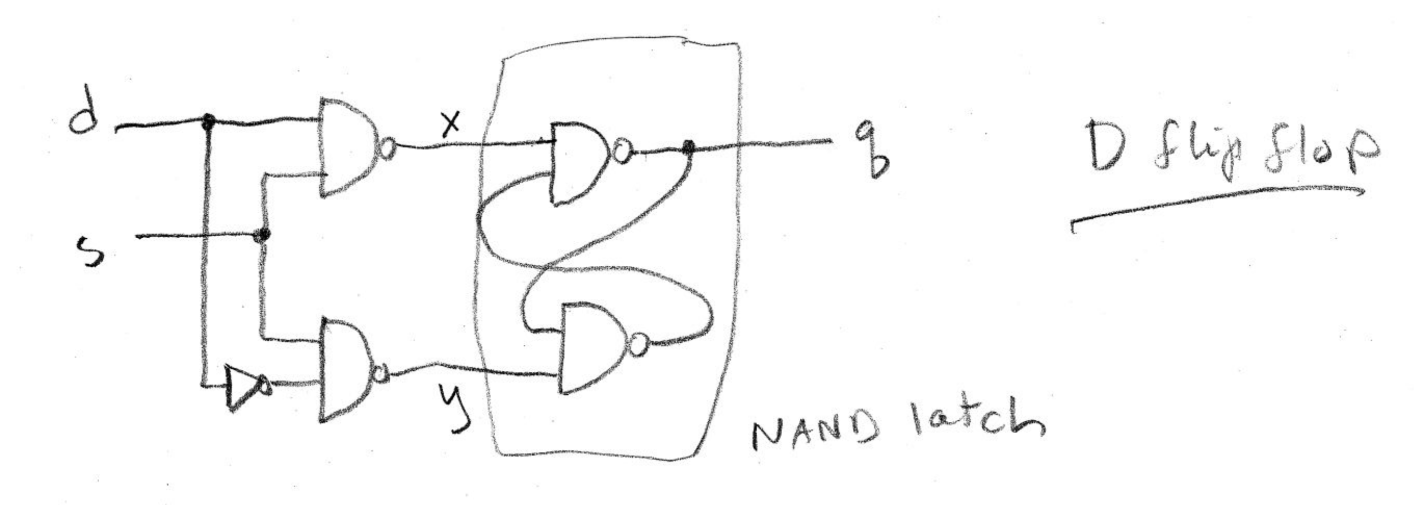
There are 5 stable states in a NAND latch. If we only move between some subset of them in a well-defined way, we can ensure that we never reach an unstable state—and, as such, we can store information. The D flip-flop is a bit of extra circuitry that allows us to use a selector wire to adjust the state of the NAND latch. If the selector is high, the NAND latch will change state in accordance with the input wire. If the selector is low, the NAND latch will not change state.

*Stable: wires are not different after one gate delay.

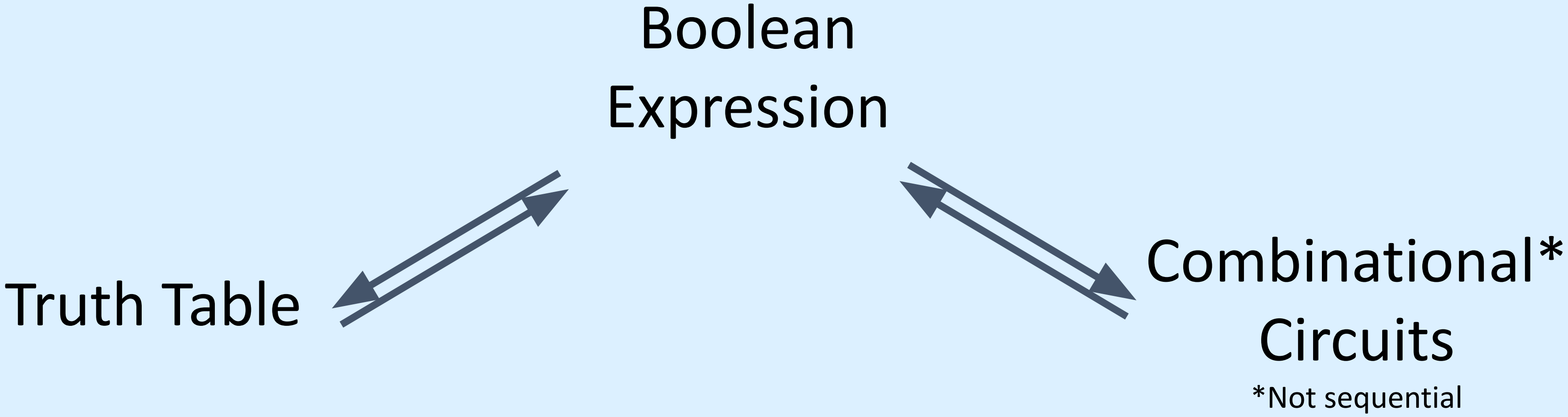
NAND Latches/D Flip-Flop

Basically, when s=0, the D-flip flop just remembers the previous value of q.

And, when s=1, q is set to the value of d.



Main Takeaway: Equivalent Forms



Boolean Expression

Truth Table

Combinational* Circuits

*Not sequential

Questions?

Extra Practice Problem (Booleans and Circuits)

Write an expression for f(x,y,z). Translate your expression into a circuit.

x	y	z	f(x,y,z)
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Answer - Part 1

x	y	z	f(x,y,z)
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Sum of Products:

$$(X'Y'Z') + (XY'Z) + (XYZ)$$

Or simplified:

$$(X'Y'Z') + (XZ)(Y' + Y)$$

$$=$$

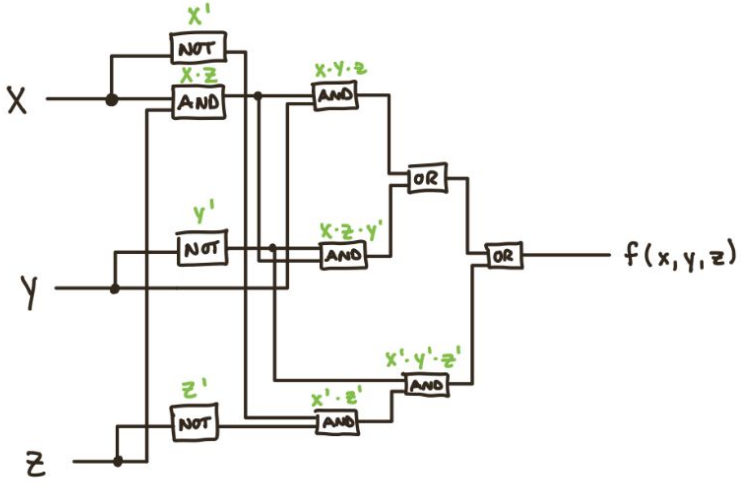
$$(X'Y'Z') + (XZ)$$

Identity Law:

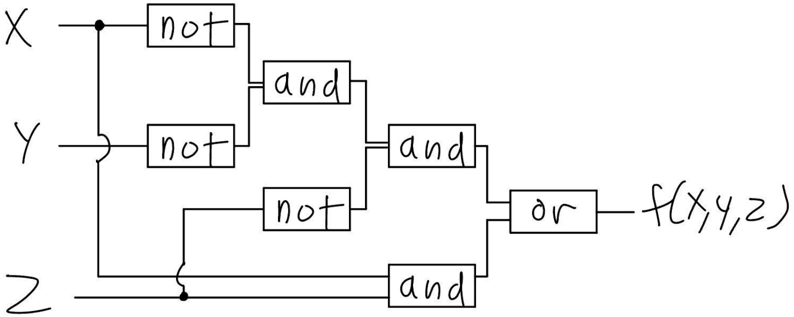
$$X + X' = 1$$

Answer - Part 2

(X’Y’Z’) + (XY’Z) + (XYZ)



(X’Y’Z’) + (XZ)



Agenda

1. Exam resources and topics
2. Boolean expressions
3. Gates and circuits
4. **UNIX Principles 3 & 4**
5. Tail recursion
6. Q&A (time permitting)

How can I get better at UNIX?

1. UNIX tutorial on the Zoo! ssh into the Zoo; then in your home folder, type the following command:

   ```
   python3 /c/cs201/www/unixtutorial.py
   ```

1. Practice typing commands on the Zoo

General tips:

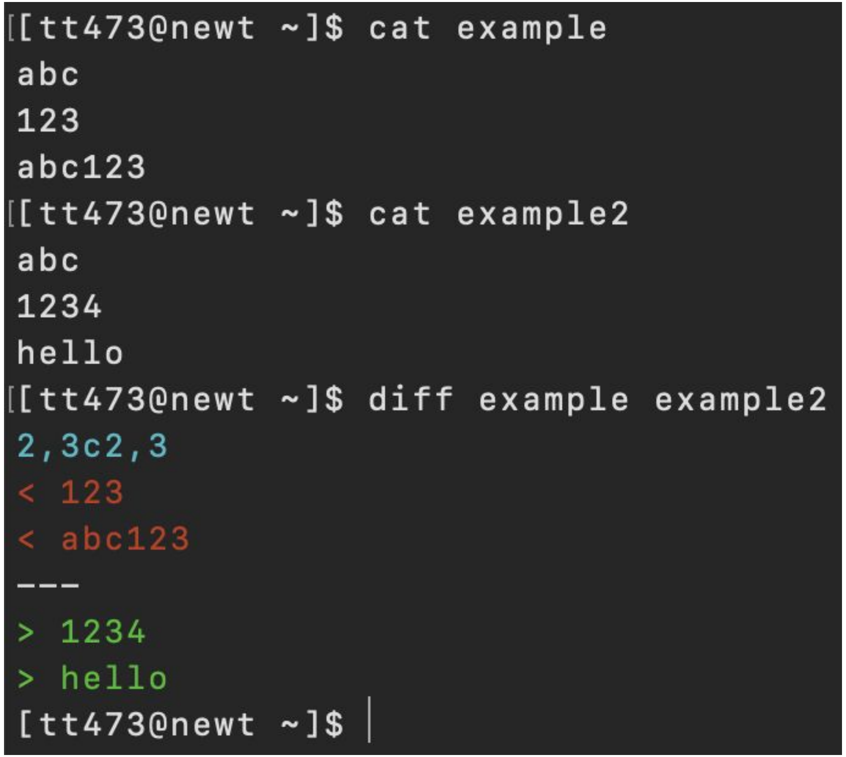
- Be familiar with the *output* of each command (important in context of the transcript!)

Useful UNIX commands (Principles 3)

- diff
- grep
- file
- --help

UNIX Highlight: diff

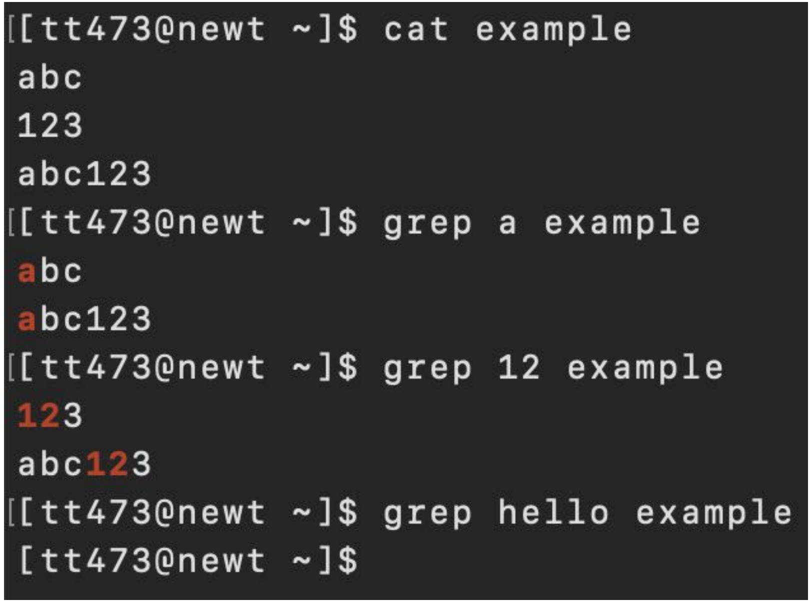
- Find difference between 2 files



```
[[tt473@newt ~]$ cat example
abc
123
abc123
[[tt473@newt ~]$ cat example2
abc
1234
hello
[[tt473@newt ~]$ diff example example2
2,3c2,3
< 123
< abc123
---
> 1234
> hello
[tt473@newt ~]$ 
```

UNIX Highlight: grep

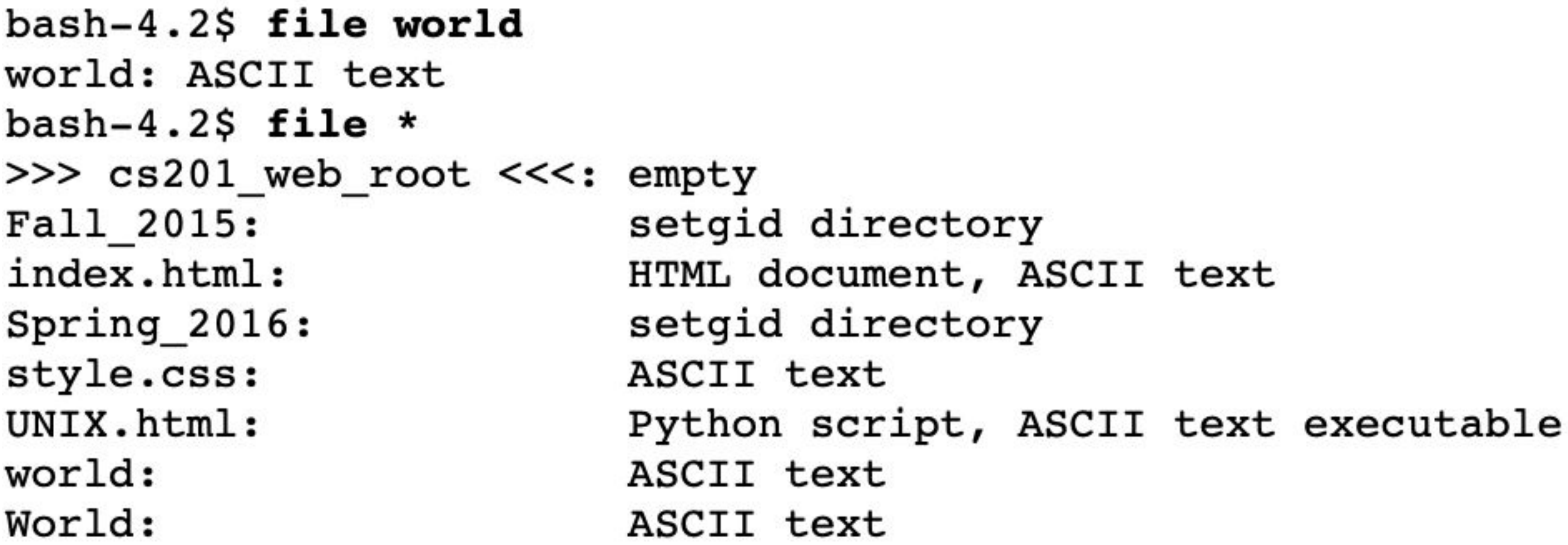
- Search an input with a given regular expression and return lines that match the pattern



```
[[tt473@newt ~]$ cat example
abc
123
abc123
[[tt473@newt ~]$ grep a example
abc
abc123
[[tt473@newt ~]$ grep 12 example
123
abc123
[[tt473@newt ~]$ grep hello example
[tt473@newt ~]$
```

UNIX Highlight: file

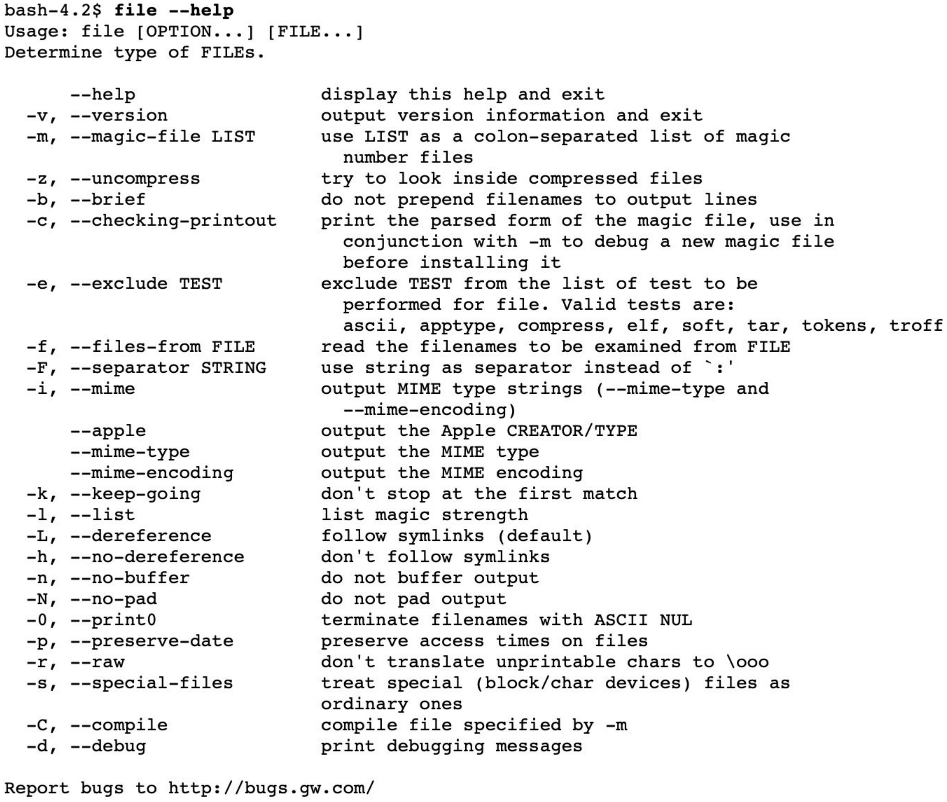
- Returns the file type of a given file or files



```
bash-4.2$ file world
world: ASCII text
bash-4.2$ file *
>>> cs201_web_root <<<: empty
Fall_2015:              setgid directory
index.html:             HTML document, ASCII text
Spring_2016:            setgid directory
style.css:              ASCII text
UNIX.html:              Python script, ASCII text executable
world:                  ASCII text
World:                  ASCII text
```

UNIX Highlight: --help

- Similar to man, but gives a succinct summary of the command’s functionality



```
bash-4.2$ file --help
Usage: file [OPTION...] [FILE...]
Determine type of FILEs.

      --help                 display this help and exit
  -v, --version              output version information and exit
  -m, --magic-file LIST      use LIST as a colon-separated list of magic
                               number files
  -z, --uncompress           try to look inside compressed files
  -b, --brief                do not prepend filenames to output lines
  -c, --checking-printout    print the parsed form of the magic file, use in
                               conjunction with -m to debug a new magic file
                               before installing it
  -e, --exclude TEST         exclude TEST from the list of test to be
                               performed for file. Valid tests are:
                               ascii, apptype, compress, elf, soft, tar, tokens, troff
  -f, --files-from FILE      read the filenames to be examined from FILE
  -F, --separator STRING     use string as separator instead of `:'
  -i, --mime                 output MIME type strings (--mime-type and
                               --mime-encoding)
      --apple                output the Apple CREATOR/TYPE
      --mime-type            output the MIME type
      --mime-encoding        output the MIME encoding
  -k, --keep-going           don't stop at the first match
  -l, --list                 list magic strength
  -L, --dereference          follow symlinks (default)
  -h, --no-dereference       don't follow symlinks
  -n, --no-buffer            do not buffer output
  -N, --no-pad               do not pad output
  -0, --print0               terminate filenames with ASCII NUL
  -p, --preserve-date        preserve access times on files
  -r, --raw                  don't translate unprintable chars to \ooo
  -s, --special-files        treat special (block/char devices) files as
                             ordinary ones
  -C, --compile              compile file specified by -m
  -d, --debug                print debugging messages

Report bugs to http://bugs.gw.com/
```

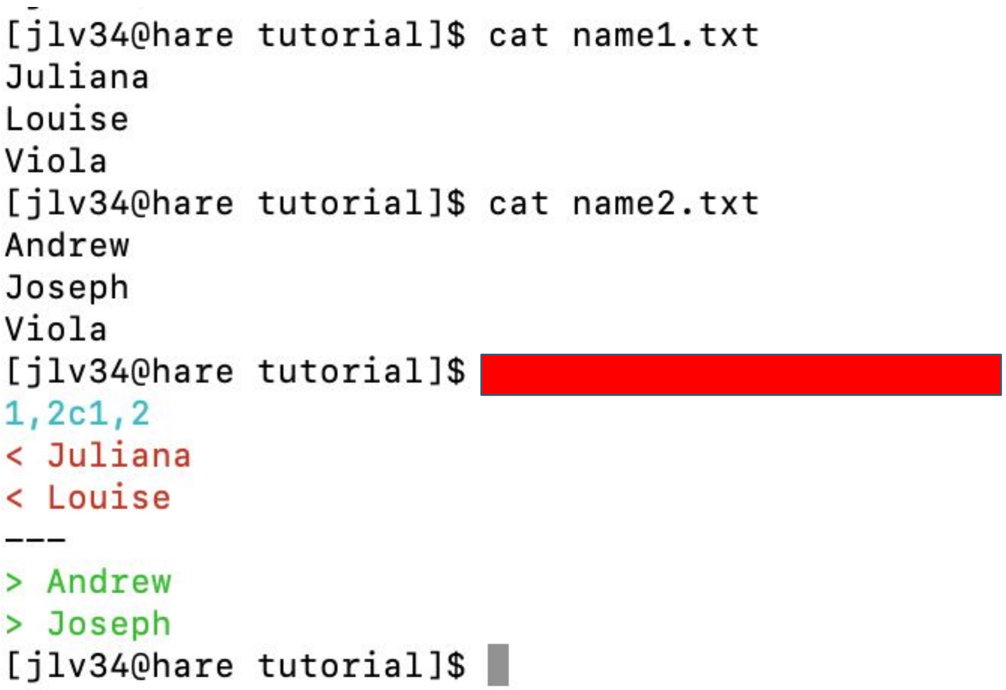
Other commands (Principle 3)

- whoami
- id
- uptime
- who
- w
- last
- info
- uname
- lsb_release
- du
- quota
- free
- finger

A few scenarios

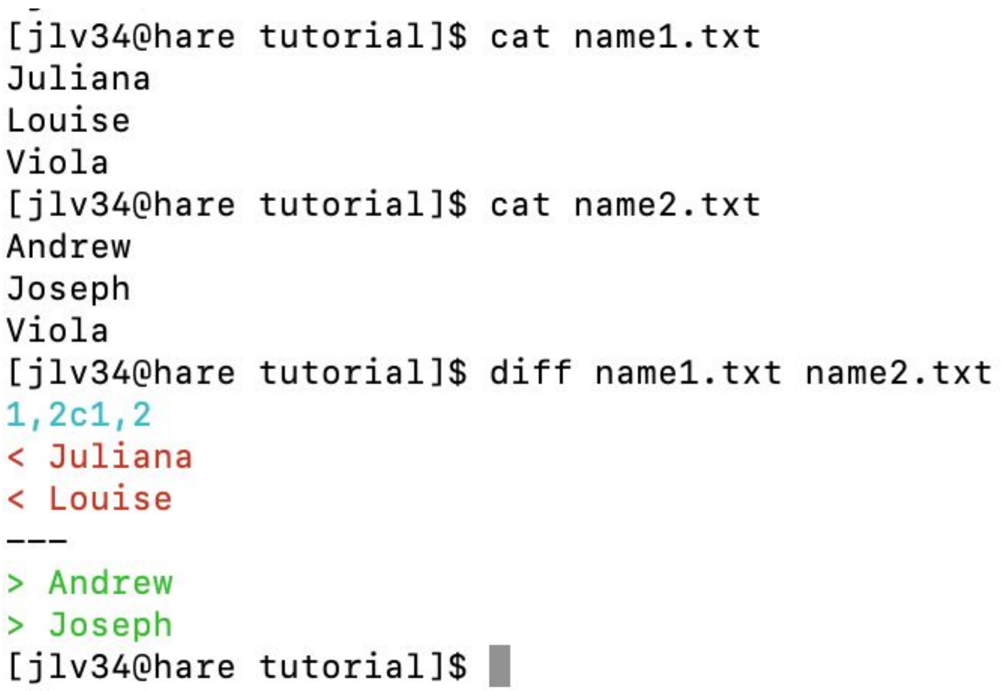
We will mostly focus on Principle 3 during this review session, but be sure to review principles 1-2 as well!

scenario 1



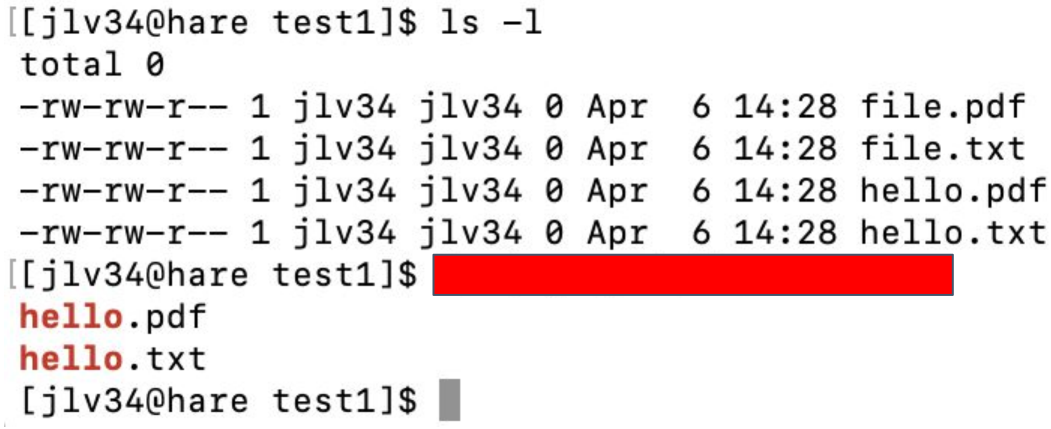
```
[jlv34@hare tutorial]$ cat name1.txt
Juliana
Louise
Viola
[jlv34@hare tutorial]$ cat name2.txt
Andrew
Joseph
Viola
[jlv34@hare tutorial]$ [redacted]
1,2c1,2
< Juliana
< Louise
---
> Andrew
> Joseph
[jlv34@hare tutorial]$
```

scenario 1



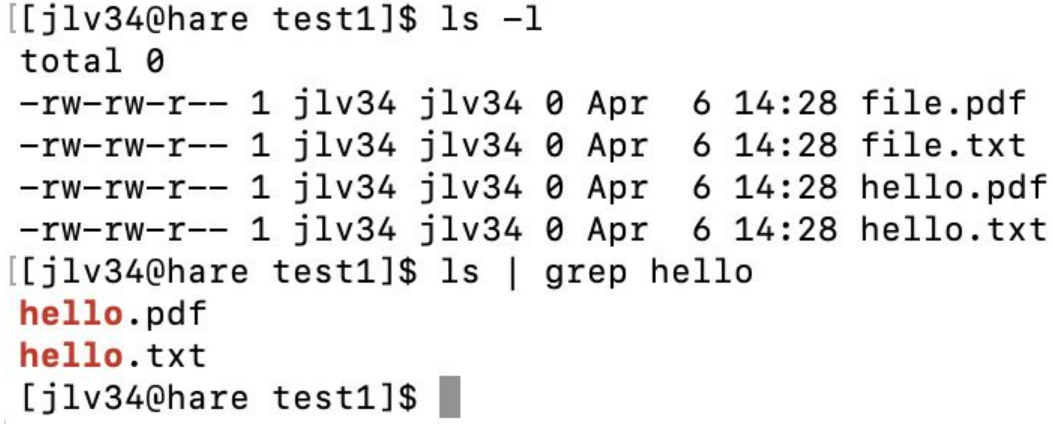
```
[jlv34@hare tutorial]$ cat name1.txt
Juliana
Louise
Viola
[jlv34@hare tutorial]$ cat name2.txt
Andrew
Joseph
Viola
[jlv34@hare tutorial]$ diff name1.txt name2.txt
1,2c1,2
< Juliana
< Louise
---
> Andrew
> Joseph
[jlv34@hare tutorial]$ 
```

scenario 2



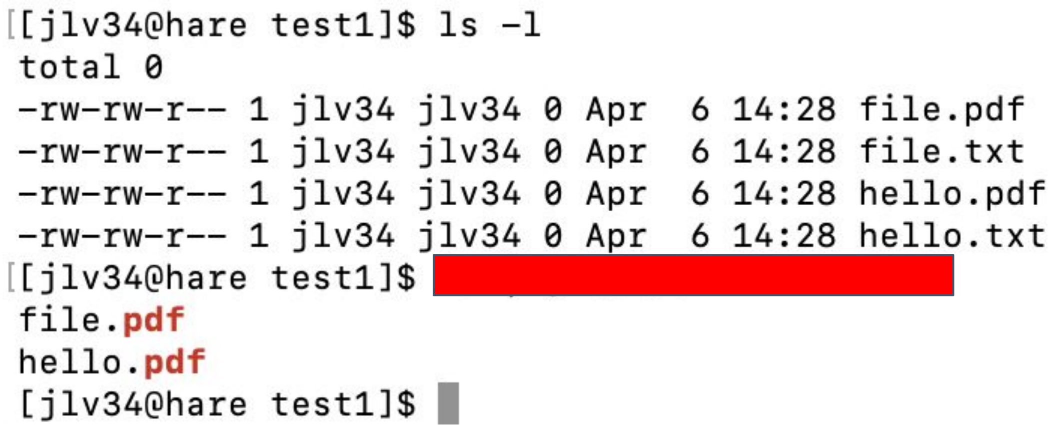
```
[[jlv34@hare test1]$ ls -l
 total 0
 -rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 file.pdf
 -rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 file.txt
 -rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 hello.pdf
 -rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 hello.txt
[[jlv34@hare test1]$ [redacted]
 hello.pdf
 hello.txt
 [jlv34@hare test1]$ 
```

scenario 2



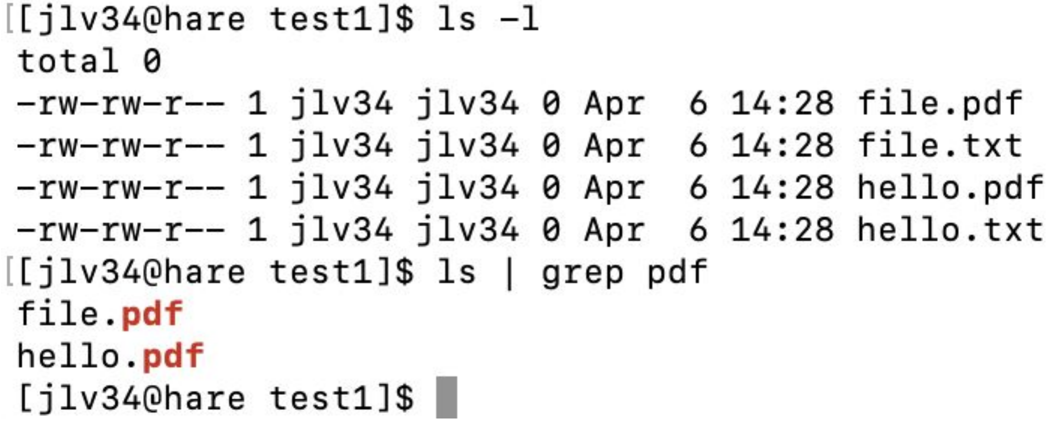
```
[[jlv34@hare test1]$ ls -l
total 0
-rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 file.pdf
-rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 file.txt
-rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 hello.pdf
-rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 hello.txt
[[jlv34@hare test1]$ ls | grep hello
hello.pdf
hello.txt
[jlv34@hare test1]$ 
```

scenario 3



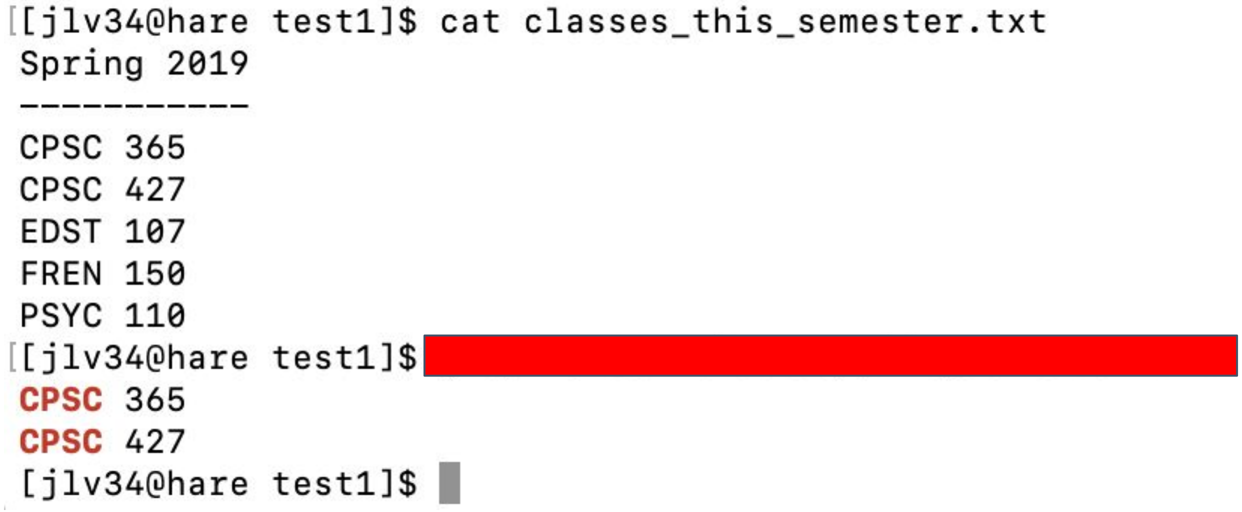
```
[[jlv34@hare test1]$ ls -l
 total 0
 -rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 file.pdf
 -rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 file.txt
 -rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 hello.pdf
 -rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 hello.txt
[[jlv34@hare test1]$ [redacted]
 file.pdf
 hello.pdf
 [jlv34@hare test1]$ 
```

scenario 3



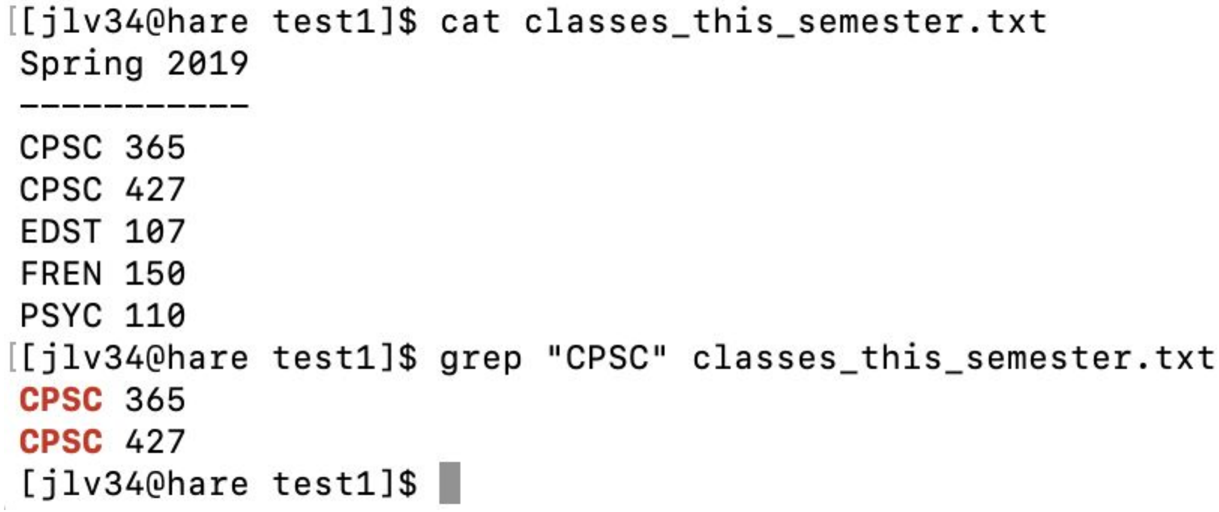
```
[[jlv34@hare test1]$ ls -l
 total 0
 -rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 file.pdf
 -rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 file.txt
 -rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 hello.pdf
 -rw-rw-r-- 1 jlv34 jlv34 0 Apr  6 14:28 hello.txt
[[jlv34@hare test1]$ ls | grep pdf
 file.pdf
 hello.pdf
 [jlv34@hare test1]$
```

scenario 4



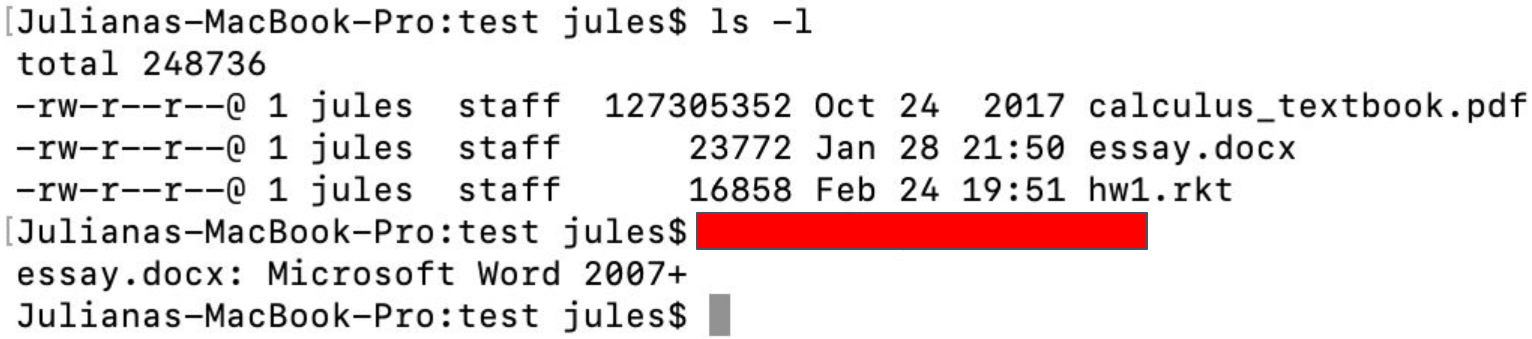
```
[[jlv34@hare test1]$ cat classes_this_semester.txt
 Spring 2019
 -----------
 CPSC 365
 CPSC 427
 EDST 107
 FREN 150
 PSYC 110
[[jlv34@hare test1]$ [redacted]
 CPSC 365
 CPSC 427
 [jlv34@hare test1]$ █
```

scenario 4



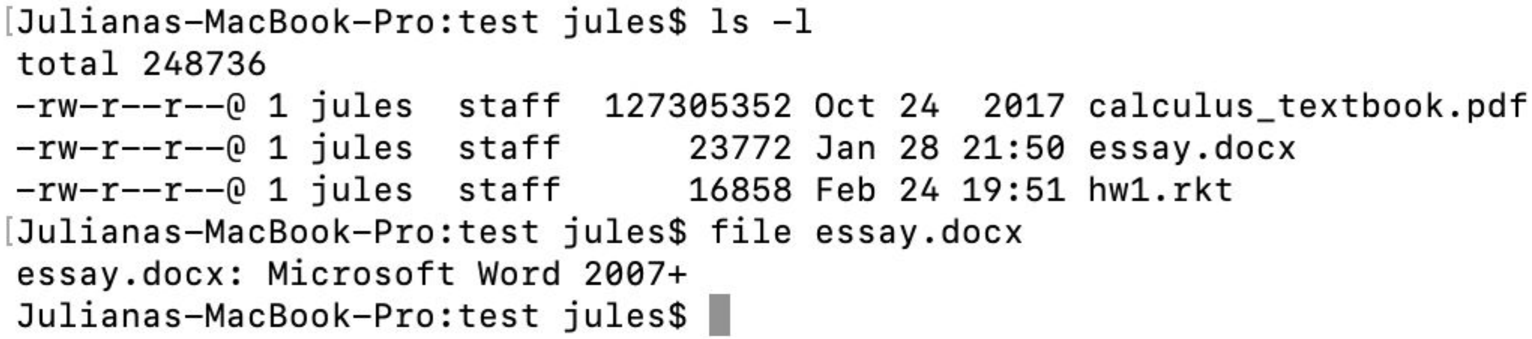
```
[jlv34@hare test1]$ cat classes_this_semester.txt
Spring 2019
-----------
CPSC 365
CPSC 427
EDST 107
FREN 150
PSYC 110
[jlv34@hare test1]$ grep "CPSC" classes_this_semester.txt
CPSC 365
CPSC 427
[jlv34@hare test1]$ 
```

scenario 5



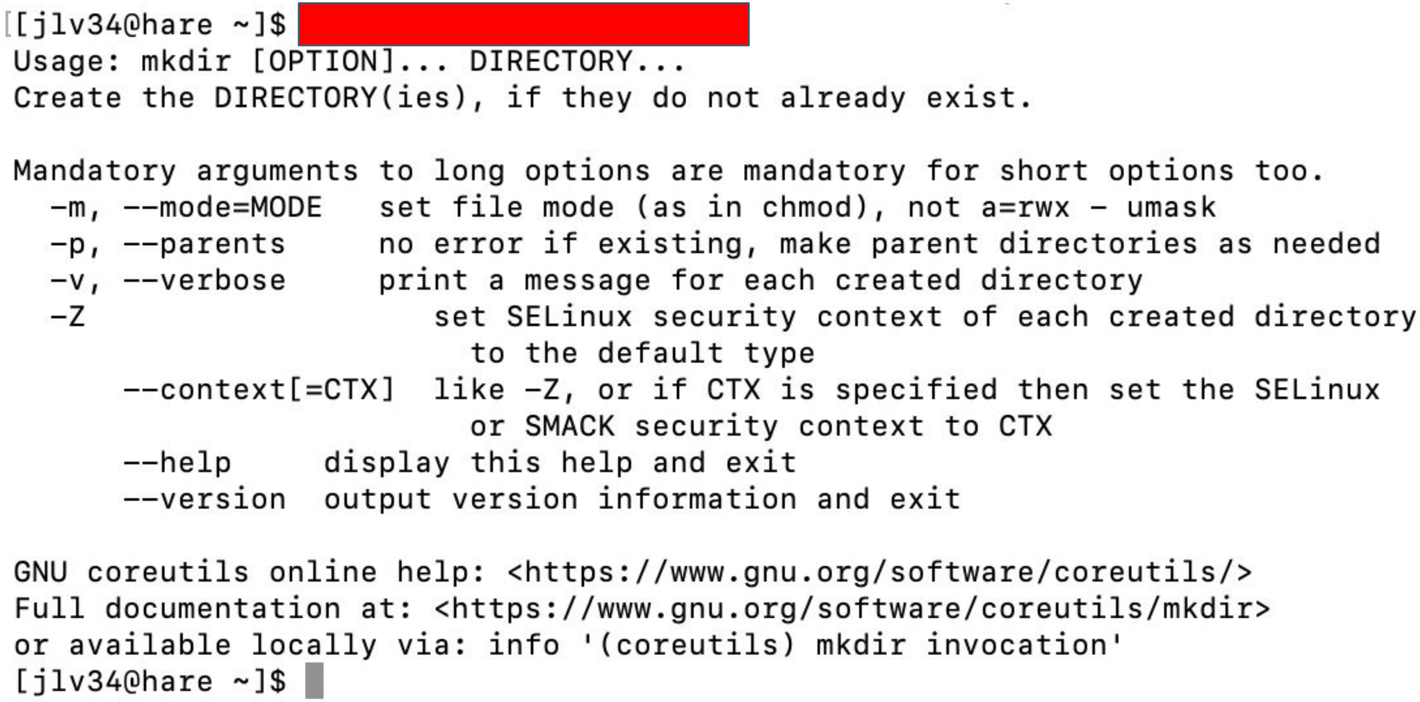
```
[Julianas-MacBook-Pro:test jules$ ls -l
 total 248736
 -rw-r--r--@ 1 jules  staff  127305352 Oct 24  2017 calculus_textbook.pdf
 -rw-r--r--@ 1 jules  staff      23772 Jan 28 21:50 essay.docx
 -rw-r--r--@ 1 jules  staff      16858 Feb 24 19:51 hw1.rkt
[Julianas-MacBook-Pro:test jules$ [redacted]
 essay.docx: Microsoft Word 2007+
 Julianas-MacBook-Pro:test jules$ 
```

scenario 5



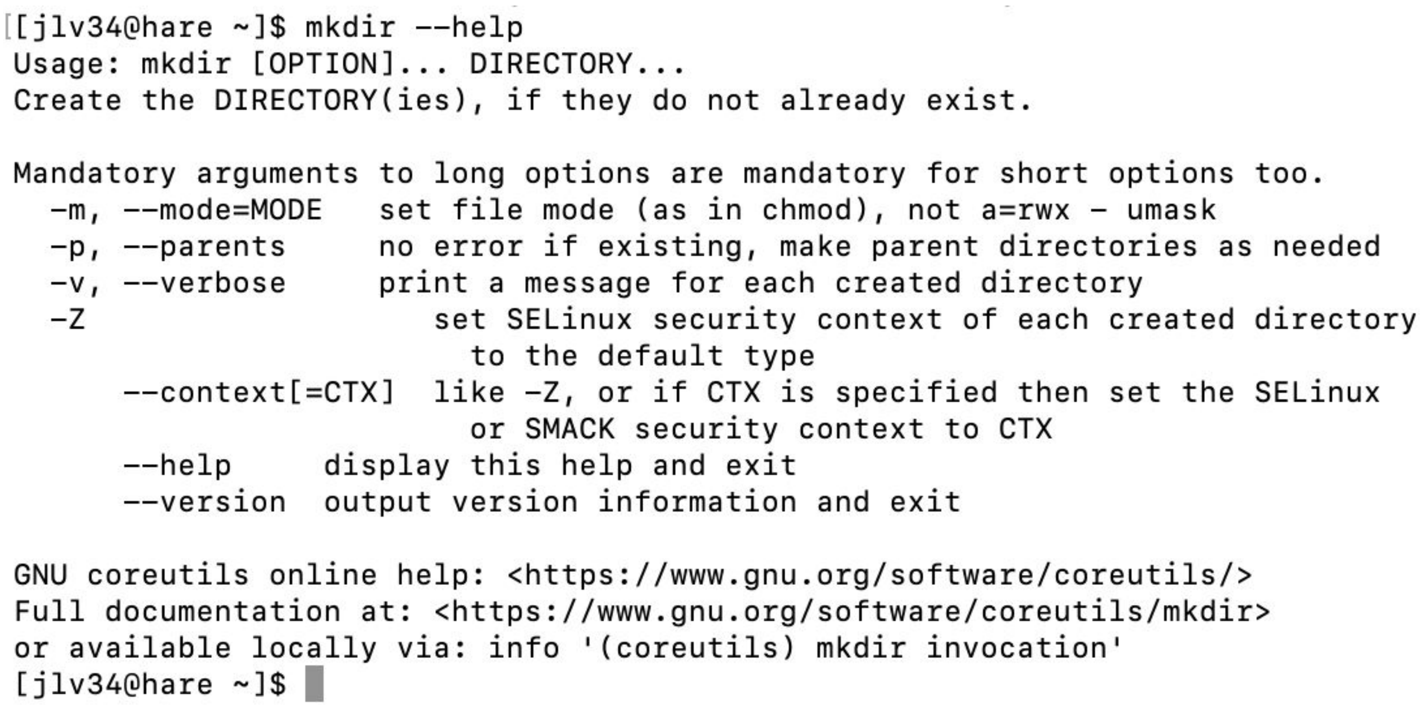
```
[Julianas-MacBook-Pro:test jules$ ls -l
total 248736
-rw-r--r--@ 1 jules  staff  127305352 Oct 24  2017 calculus_textbook.pdf
-rw-r--r--@ 1 jules  staff      23772 Jan 28 21:50 essay.docx
-rw-r--r--@ 1 jules  staff      16858 Feb 24 19:51 hw1.rkt
[Julianas-MacBook-Pro:test jules$ file essay.docx
essay.docx: Microsoft Word 2007+
Julianas-MacBook-Pro:test jules$ 
```

scenario 6



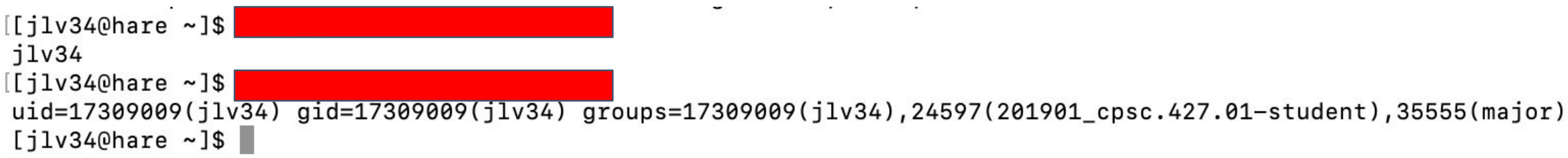
```
[[jlv34@hare ~]$ [redacted]
Usage: mkdir [OPTION]... DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short options too.
  -m, --mode=MODE   set file mode (as in chmod), not a=rwx - umask
  -p, --parents     no error if existing, make parent directories as needed
  -v, --verbose     print a message for each created directory
  -Z                   set SELinux security context of each created directory
                         to the default type
      --context[=CTX]  like -Z, or if CTX is specified then set the SELinux
                         or SMACK security context to CTX
      --help     display this help and exit
      --version  output version information and exit

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation at: <https://www.gnu.org/software/coreutils/mkdir>
or available locally via: info '(coreutils) mkdir invocation'
[jlv34@hare ~]$ █
```

scenario 6



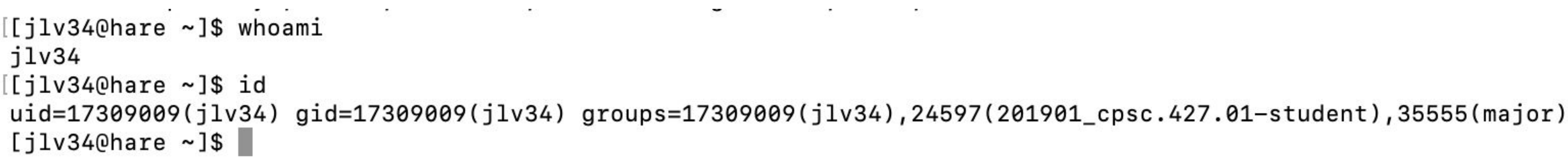
```
[[jlv34@hare ~]$ mkdir --help
Usage: mkdir [OPTION]... DIRECTORY...
Create the DIRECTORY(ies), if they do not already exist.

Mandatory arguments to long options are mandatory for short options too.
  -m, --mode=MODE   set file mode (as in chmod), not a=rwx - umask
  -p, --parents     no error if existing, make parent directories as needed
  -v, --verbose     print a message for each created directory
  -Z                   set SELinux security context of each created directory
                         to the default type
      --context[=CTX]  like -Z, or if CTX is specified then set the SELinux
                         or SMACK security context to CTX
      --help     display this help and exit
      --version  output version information and exit

GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
Full documentation at: <https://www.gnu.org/software/coreutils/mkdir>
or available locally via: info '(coreutils) mkdir invocation'
[jlv34@hare ~]$ 
```

scenario 7



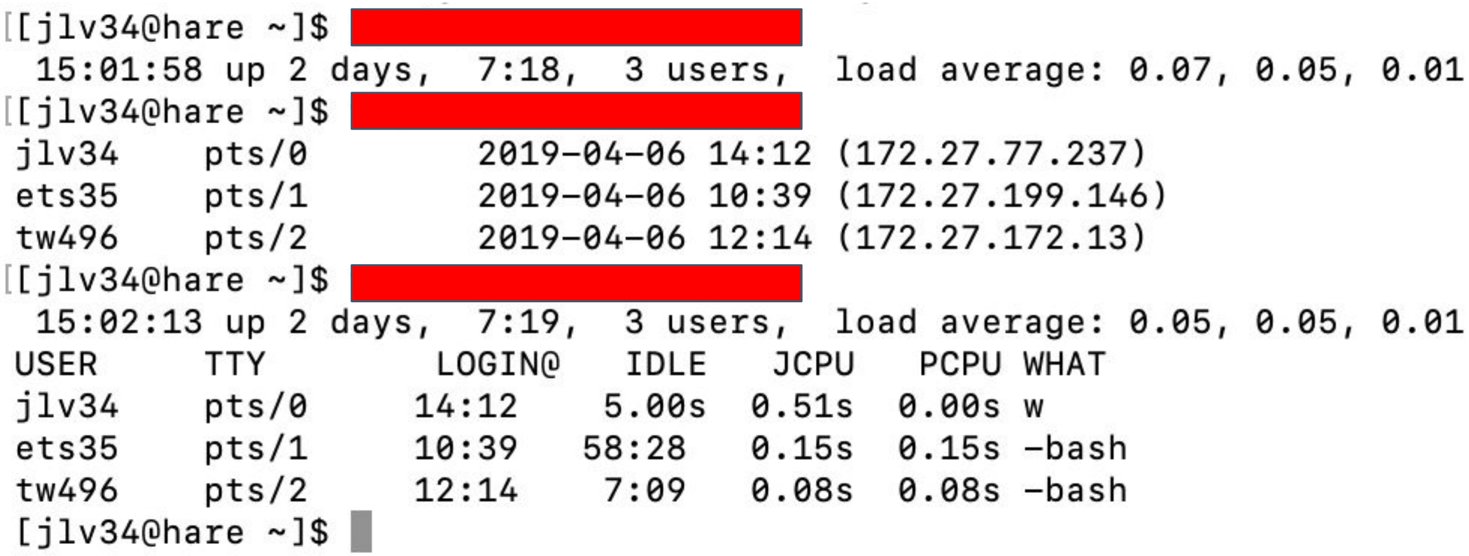
```
[[jlv34@hare ~]$ [redacted]
 jlv34
[[jlv34@hare ~]$ [redacted]
 uid=17309009(jlv34) gid=17309009(jlv34) groups=17309009(jlv34),24597(201901_cpsc.427.01-student),35555(major)
 [jlv34@hare ~]$ 
```

scenario 7



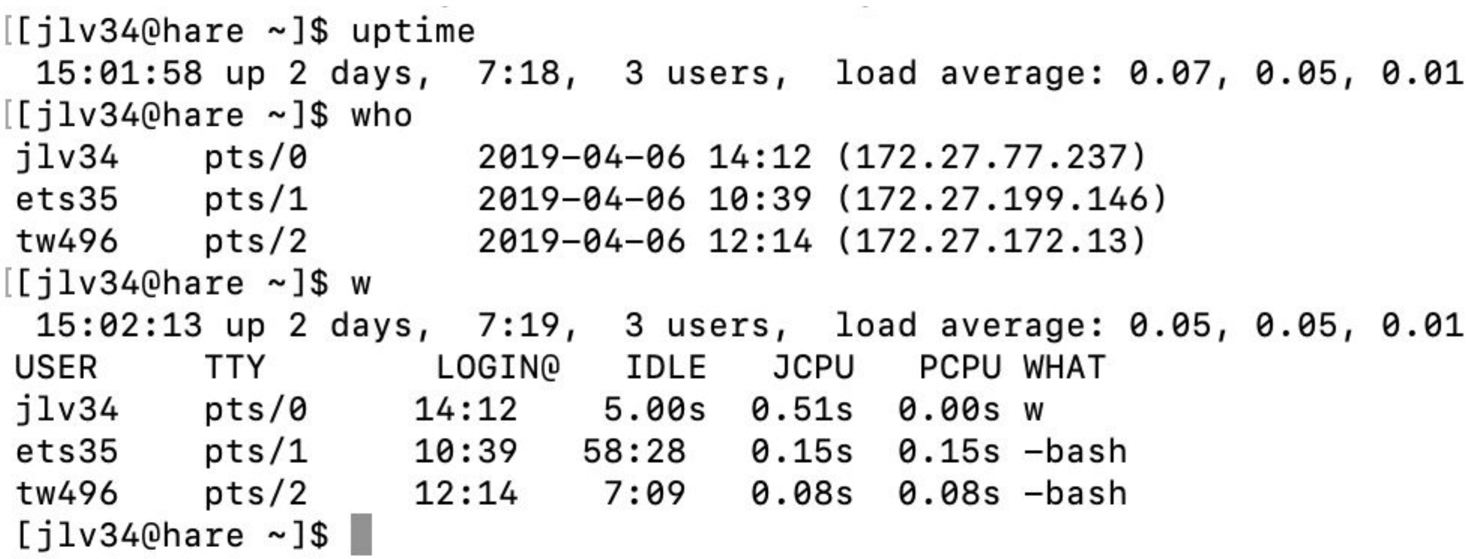
```
[[jlv34@hare ~]$ whoami
 jlv34
[[jlv34@hare ~]$ id
 uid=17309009(jlv34) gid=17309009(jlv34) groups=17309009(jlv34),24597(201901_cpsc.427.01-student),35555(major)
 [jlv34@hare ~]$ 
```

scenario 8



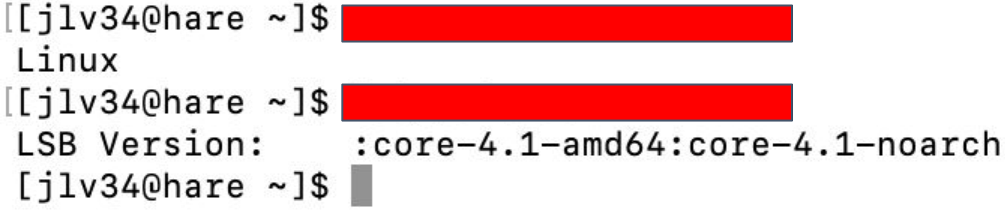
```
[[jlv34@hare ~]$ [redacted]
 15:01:58 up 2 days,  7:18,  3 users,  load average: 0.07, 0.05, 0.01
[[jlv34@hare ~]$ [redacted]
jlv34    pts/0        2019-04-06 14:12 (172.27.77.237)
ets35    pts/1        2019-04-06 10:39 (172.27.199.146)
tw496    pts/2        2019-04-06 12:14 (172.27.172.13)
[[jlv34@hare ~]$ [redacted]
 15:02:13 up 2 days,  7:19,  3 users,  load average: 0.05, 0.05, 0.01
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
jlv34    pts/0     14:12    5.00s  0.51s  0.00s w
ets35    pts/1     10:39   58:28   0.15s  0.15s -bash
tw496    pts/2     12:14    7:09   0.08s  0.08s -bash
[jlv34@hare ~]$
```

scenario 8



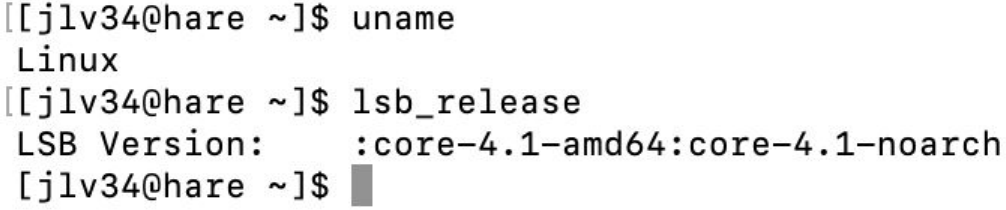
```
[[jlv34@hare ~]$ uptime
 15:01:58 up 2 days,  7:18,  3 users,  load average: 0.07, 0.05, 0.01
[[jlv34@hare ~]$ who
jlv34    pts/0        2019-04-06 14:12 (172.27.77.237)
ets35    pts/1        2019-04-06 10:39 (172.27.199.146)
tw496    pts/2        2019-04-06 12:14 (172.27.172.13)
[[jlv34@hare ~]$ w
 15:02:13 up 2 days,  7:19,  3 users,  load average: 0.05, 0.05, 0.01
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
jlv34    pts/0     14:12    5.00s  0.51s  0.00s w
ets35    pts/1     10:39   58:28   0.15s  0.15s -bash
tw496    pts/2     12:14    7:09   0.08s  0.08s -bash
[jlv34@hare ~]$
```

scenario 9



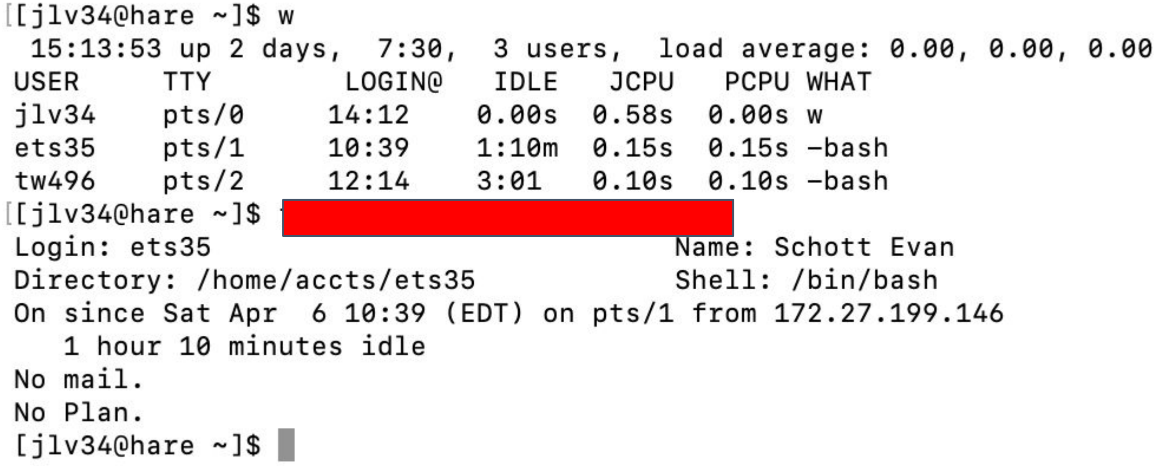
```
[[jlv34@hare ~]$ [redacted]
 Linux
[[jlv34@hare ~]$ [redacted]
 LSB Version:    :core-4.1-amd64:core-4.1-noarch
 [jlv34@hare ~]$
```

scenario 9



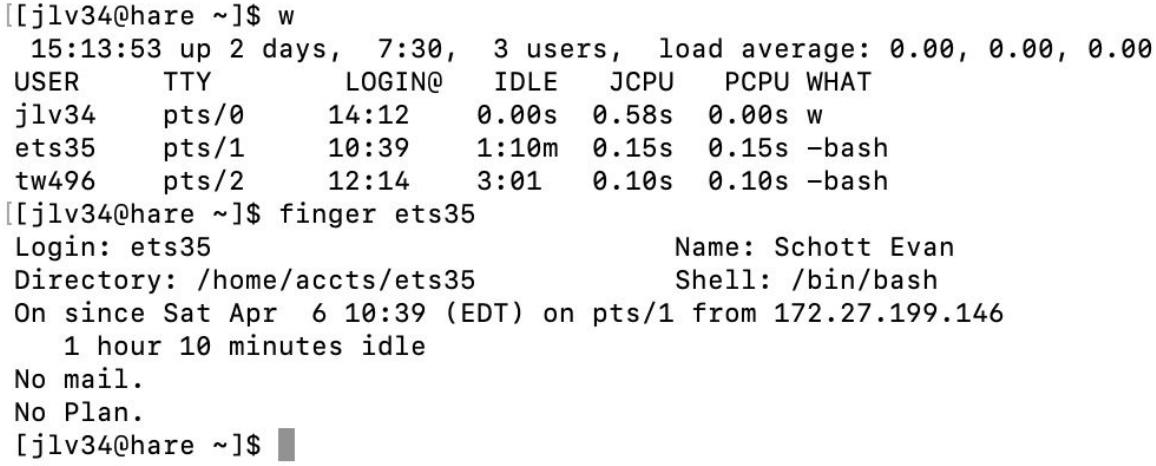
```
[[jlv34@hare ~]$ uname
 Linux
[[jlv34@hare ~]$ lsb_release
 LSB Version:    :core-4.1-amd64:core-4.1-noarch
 [jlv34@hare ~]$
```

scenario 10



```
[[jlv34@hare ~]$ w
 15:13:53 up 2 days,  7:30,  3 users,  load average: 0.00, 0.00, 0.00
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
jlv34    pts/0     14:12    0.00s  0.58s  0.00s w
ets35    pts/1     10:39    1:10m  0.15s  0.15s -bash
tw496    pts/2     12:14    3:01   0.10s  0.10s -bash
[[jlv34@hare ~]$ [redacted]
Login: ets35                            Name: Schott Evan
Directory: /home/accts/ets35            Shell: /bin/bash
On since Sat Apr  6 10:39 (EDT) on pts/1 from 172.27.199.146
   1 hour 10 minutes idle
No mail.
No Plan.
[jlv34@hare ~]$ 
```

scenario 10



```
[[jlv34@hare ~]$ w
 15:13:53 up 2 days,  7:30,  3 users,  load average: 0.00, 0.00, 0.00
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
jlv34    pts/0     14:12    0.00s  0.58s  0.00s w
ets35    pts/1     10:39    1:10m  0.15s  0.15s -bash
tw496    pts/2     12:14    3:01   0.10s  0.10s -bash
[[jlv34@hare ~]$ finger ets35
Login: ets35                            Name: Schott Evan
Directory: /home/accts/ets35            Shell: /bin/bash
On since Sat Apr  6 10:39 (EDT) on pts/1 from 172.27.199.146
   1 hour 10 minutes idle
No mail.
No Plan.
[jlv34@hare ~]$ █
```

Extra UNIX 1

```
[crb84@scorpion tmp]$ ls
Friday  test  test_01  test_02
[crb84@scorpion tmp]$ XXXX1
[crb84@scorpion tmp]$ ls
files  Friday  test  test_01  test_02
[crb84@scorpion tmp]$ ls > files_new
[crb84@scorpion tmp]$ XXXX2
0a1,3
> files
> files_new
> Friday
[crb84@scorpion tmp]$
```

Extra UNIX 1 Solution

```
[crb84@scorpion tmp]$ ls
Friday  test  test_01  test_02
[crb84@scorpion tmp]$ ls | grep test > files
[crb84@scorpion tmp]$ ls
files  Friday  test  test_01  test_02
[crb84@scorpion tmp]$ ls > files_new
[crb84@scorpion tmp]$ diff files files_new
0a1,3
> files
> files_new
> Friday
[crb84@scorpion tmp]$
```

Extra UNIX 2

```
[crb84@scorpion tmp]$ ls -l
total 0
[crb84@scorpion tmp]$ XXXX1
[crb84@scorpion tmp]$ ls -l
total 4
-rw-rw-r-- 1 crb84 crb84 204 Apr 15 21:49 1
[crb84@scorpion tmp]$ XXXX2
       total        used        free      shared  buff/cache   available
```

Extra UNIX 2 Solution

```
[crb84@scorpion tmp]$ ls -l
total 0
[crb84@scorpion tmp]$ free > 1
[crb84@scorpion tmp]$ ls -l
total 4
-rw-rw-r-- 1 crb84 crb84 204 Apr 15 21:49 1
[crb84@scorpion tmp]$ head -n1 1
       total        used        free      shared  buff/cache   available
```

Questions?

Agenda

1. Exam resources and topics
2. Boolean expressions
3. Gates and circuits
4. UNIX Principles 3 & 4
5. **Tail recursion**
6. Q&A (time permitting)

Things to know about tail recursion

You should be able to…

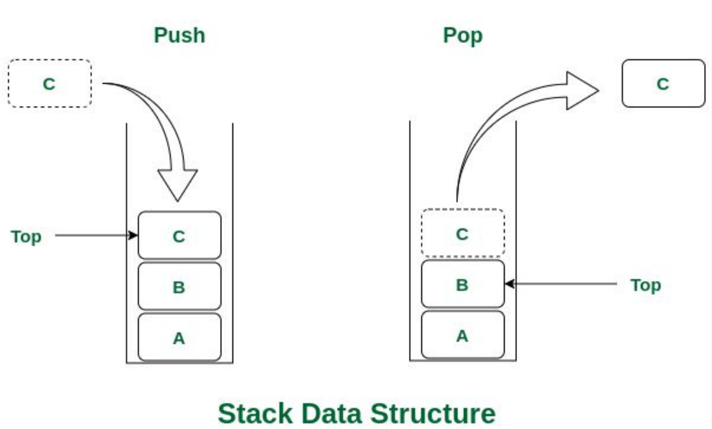
1. Describe tail recursion generally
2. Implement a tail-recursive function
3. Explain the benefits of tail recursion

What is Tail Recursion?

- A style of writing a recursive function to save memory and increase efficiency
  - Uses less memory since recursive calls don’t build up on the stack
  - Faster because you don’t have to push and pop extra calls
- A function is tail recursive if it executes the recursive call last in its definition

An Aside on Stacks

- Standard type of linear data structure
- Last in, First out (LIFO) method for adding/removing data
- Adding to a stack is called “push” and removing from a stack is called “pop”
  - Think of a stack of plates
    - You can only add/remove from the top



What is Tail Recursion?

Key Idea: The **last call** in the function definition is the **recursive call***

In other words, the highest level function inside your recursive function must be the recursive call. The recursive call **cannot** be an argument to another function (except if/cond)

There are some clues you can look for to determine if a function fulfills these criteria

*Definition taken from lecture notes

Identifying Tail Recursion

2 things to check for: **top level function** and **base case return value**

1. Top-level function (required)
    a. Is the highest level function in the definition the recursive call? (i.e. are all functions you're using nested under the recursive call as arguments?)*
2. Base case (more informal)
    a. Are you returning the result at the base case or a starting value to build on? (i.e. are you returning the final output you want or something like a list that will be cons'd onto?)

*some minor exceptions to this like if/cond statements

Writing Reverse Function – 2 Styles

Writing Reverse Function – 2 Styles

Basic Recursion

```
(define (rev1 lst)
    (if (empty? lst)
         ‘()
         (append
             (rev1 (rest lst))
             (list (first lst))
         )
    )
)
```

Writing Reverse Function – 2 Styles

Basic Recursion

```
(define (rev1 lst)
    (if (empty? lst)
        ‘()
        (append
            (rev1 (rest lst))
            (list (first lst))
        )
    )
)
```

Tail Recursion

```
(define (rev2 lst [result ‘()])
    (if (empty? lst)
        result
        (rev2
            (rest lst)
            (cons (first lst) result)
        )
    )
)
```

Writing Reverse Function – 2 Styles

Basic Recursion

```
(define (rev1 lst)
    (if (empty? lst)
        ‘()
        (append
            (rev1 (rest lst))
            (list (first lst))
        )
    )
)
```

Notice what gets returned at base case

Tail Recursion

```
(define (rev2 lst [result ‘()])
    (if (empty? lst)
        result
        (rev2
            (rest lst)
            (cons (first lst) result)
        )
    )
)
```

Writing Reverse Function – 2 Styles

Basic Recursion

```
(define (rev1 lst)
    (if (empty? lst)
        ‘()
        (append
            (rev1 (rest lst))
            (list (first lst))
        )
    )
)
```

Notice what function is at the highest level

Tail Recursion

```
(define (rev2 lst [result ‘()])
    (if (empty? lst)
        result
        (rev2
            (rest lst)
            (cons (first lst) result)
        )
    )
)
```

Agenda

1. Exam resources and topics
2. Boolean expressions
3. Gates and circuits
4. UNIX Principles 3 & 4
5. Tail recursion
6. **Q&A (time permitting)**