Makefile

- gmake on Linux/OSX, cmake = cross platform
- "directed acyclic graph" → instructions for make
- target : ... : dependencies...
  <TAB> command
  " "
  " "
- target = name of rule (all, clean, format)
- phony target doesn't produce file w/ same name when executed
  ↳ won't check for files since real & phony targets are diff
  ↳ all, clean, and debug (CFLAG += -g)

- Flag options
  - -c <directory name>, --directory=<directory name>
  - -d (debugs)
  - -f <file name>, --file=<file name>, makefile=<file name>
    (specifies which to be read as Makefile)

--warn-undefined variables

- assignment: = lazy ?= conditional
  := immediate += concatenation

- = waits until variable is used
- := reference expanded before assignment
- ?= assignment if & only if value not assigned to variable
- += adds text to defined variable separated by space

- dependency = target or filename (ex: build: clean hello.o
  build depends on clean & hello.o targets)
- Makefile has topological ordering
- command = action to be executed

- cc = gcc or cc or clang
- CFLAGS = Wall, Wextra, Wpedantic, Werror, g
- OBJ = object files list to build & link
- SRC = source files (.c) list

- shell, wild card, patsubst functions
  - shell = command expansion
  - wildcard used in rules as * (expanded by shell)
  - patsubst formatted as $, finds white space

Ex) %.c matches all text leading up to ".c"
- use make as command
- Can use Makefile to include other Makefiles
- including Makefile overrides variable & targets w/ same name
  ↳ better to use recursive approach

represent by O

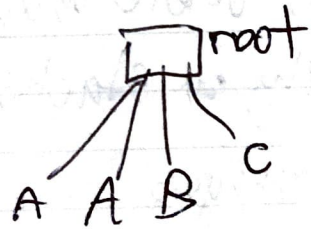
Files

- computer has millions, data centers billions, Internet trillions
- types: bak hlp mp3 pdf txt
  c html mpg ps zip
  gif jpg o tex
- access
  - sequential = read bytes/records from beginning
    can't jump but may rewind/back up
  - random access = bytes/records read in any order
    essential for data base systems
- operations: create close append get or set attributes
  delete read seek
  open write rename

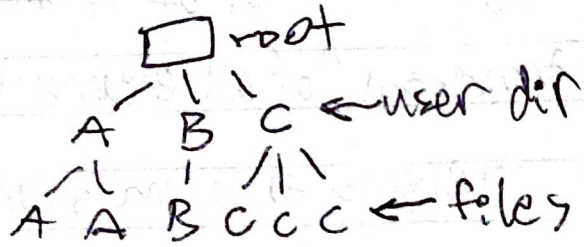
represented by □

Directories (dir)

- basically folders of files grouped
- single level system
  - 4 files
  - owned by 3 ppl A, B, C



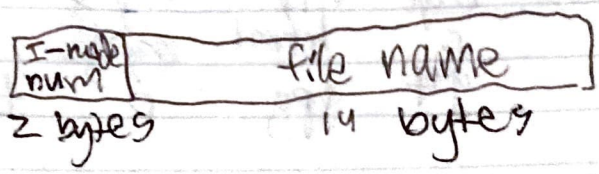
- two level system
  - letters = owners of dir & files



- hierarchical system has sub directories

- operations: create, delete, opendir, closedir, readdir, rename, link, unlink

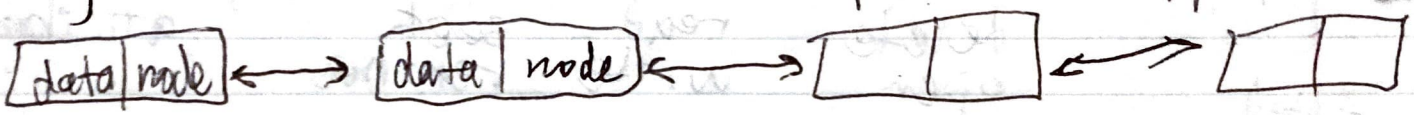
- Unix V7 file system



- cat.c ex) copies std in & out
  sequentially copies many files to std output

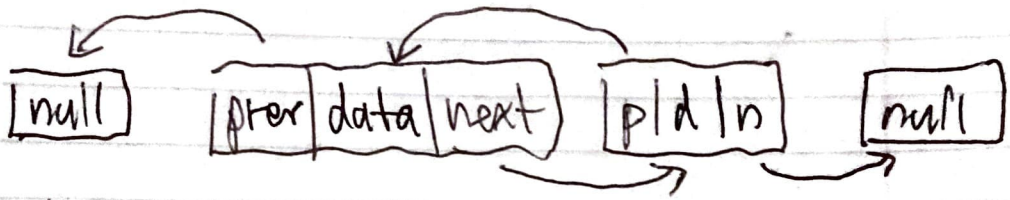
Linked Lists (LL)

- singly LL: node has data field & ptr to next node in list
- doubly LL: each node has data field & ptrs to next & prev node

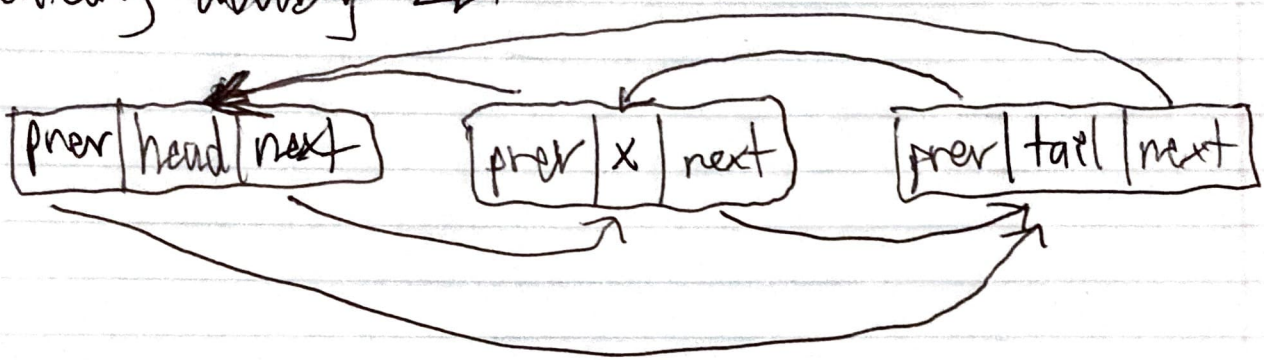


- LL advantages
  - no fixed memory allocated
  - insert & delete w/out shifting
  - use easily in stacks & queues
- LL disadvantages
  - requires a lot of memory, not efficient
  - traversal of all elements required to access specific
    - reverse easy in doubly but hard in singly
- circular singly LL = last node points to tail

- doubly LL



- sentinel nodes mark ends of LL
  placed at head and tail (1st & last data) of doubly LL
- circularly doubly LL



- LL Stacks
  - stack size limited by available memory
  - pushes/pops at head
- LL Queue
  - add at tail, remove at head

(11/05) [good diagrams on representing doubly LL insertion in slides]
↳ same for ~~removing~~ moving to front & removing node