Working with multiple C files

- Why put your code in separate files?
  - Easier to read
  - Easier to change
  - Faster to compile changes
  - Easier to re-use code in other contexts

- Exactly one file must have the `main` function
- Other functions can be put in separate files
- Appropriate headers must be included so the functions are declared for each part of the code, and the code knows where the functions are defined
- Compile each `.c` file into an object file `.o` with `gcc -c`
- Link the object files with `gcc` to create an executable

`randnum.c, random.c, mean.c`

Makefiles

- A makefile is a specially-formatted file with information for compiling a set of `.c` files
- It contains a set of dependencies and rules
  - Dependencies are what other files the object needs in order to be created
  - Rules are the commands for compiling an object
- The format, especially the tabs, are very important

`Makefile` example for `randnum.exe`

Re-directing Output

- Normally `printf()` sends output to the screen. What if you want the screen output (including error messages) to be captured and sent to a file?
- Use a redirect `>`, e.g., `a.out > outfile`
- What if you want the output of a command to be saved to a file?
- `tail -20 random.dat > sample.dat`

Foreground and Background

- Normally when you run something, it runs in the foreground, i.e., it is the only active process from that shell, and you can't do anything else in that shell.
- You can run multiple processes in the background by putting an ampersand (&) at the end of the command.
- Background processes continue even after you log off.
- Long jobs should be run in the background, so you don't tie up the terminal.
- Make sure your long jobs really do finish, and don't become run-away processes.
- Programs requiring input may need to be run in the foreground

- A foreground process can be moved to the background with Ctrl-Z (which suspends it) and `bg`
- Foreground processes can be killed with Ctrl-C. Background processes must be killed with `kill` (or kill -9), with the pid as the argument, e.g., `kill 12814`.

Top

- Displays:
  - current processes
  - pid's
  - priorities
  - memory usage
  - CPU usage
- Use to see which computers are already in use
- Use to check on your processes

Nice

- The *nice* level controls the amount of CPU time a process gets when there is competition (i.e., other processes are running)
- The levels range from 0 (not nice) to 19 (very nice), the default is 4
- If you are running a long process on a public machine, you should `nice` it, e.g., `nice a.out > output.dat &`
- There are two slightly different syntaxes:
  - Built-in shell command `nice +10 a.out &`
  - Standard command `/usr/bin/nice -n 10 a.out &`