C / RECAP

ARGUMENTS TO MAIN

- So far we've run all the programs as `./executable_name` without any arguments
- Can also accept command line arguments in C
- Currently, main accepts `void`

```
int main(void) {
    return 0;
}
```

ARGUMENTS TO MAIN

- Need to replace void with arguments
  - `argc` = number of arguments passed
  - `argv` = array of arguments (each arg is `char *`)

```
int main(int argc, char *argv[]) {
    return  0;
}
```

ARGUMENTS

- First arg is always name of program
- Other args may need to be converted to a different type
  - Example `./average 100 110`
    - `argv[1]` is `char *` storing `100`
    - need to convert to int
    - `int atoi(char *str)` in `stdlib.h`
    - other conversion functions also in `stdlib.h`

MISC C

- Variables can be preceded by `const` to be read-only
  - We used #define for the same purpose
  - Using `const` qualifier will have normal variable scope
- `static` in front of variable
  - variable maintains value even after block exits
  - Ex: `static int n` in function could be used to identify if it is first call to function or not

MISC C - LIBRARIES

- Compiling our own mutli-file programs
  - Compile all together with gcc
  - Compile into separate object files, then compile those together
- Standard libraries - don't need to compile them
  - already exist in compiled form (a library)
  - are "linked to" - in default spot, no need to specify
  - have header files that are included - in default spot
- Can do this for custom libraries
  - Need to specify where and link when

compiling (-L and -l options)

- Likely need to specify where headers live (-I option)

RECAP:

- Course split into 2 parts
  - Linux
  - C

RECAP: LINUX

- Understanding directory structure
  - General layout
  - Relative vs absolute paths
  - What is `/`, `..`, `~`, etc.
- Basic commands to navigate via terminal
  - Examples: `cd`, `mkdir`, `ls`, etc.
- Basic commands to view files
  - Examples: `more`, `cat`, `uniq`, etc.
- Remote access and file transfer

RECAP: LINUX

- Compressing/Uncompressing files
- Archiving files
- Recording shell sessions (`script`)
- `history` command
- Input/output/error redirection
- Piping
- File manipulation (ex: `cut`, `tr`, etc.)
- Basic utilities (ex: `diff`, `wc`, `grep`, etc.)

RECAP: LINUX

- git: operations, general idea, local and remote, branches, etc.
- file permissions
  - what they mean
  - viewing them, changing them
- processes:
  - viewing, killing, etc.
  - running in foreground vs background
- aliases, environment variables

RECAP: LINUX

- **Be able to use the manpages**
- Bash scripting
- sed
- gawk
- you should be able to use bash scripting, sed, awk, grep (and know what should be used when)
- regular expressions (special characters, character classes)

RECAP: C

- Harder to break down
- Understand compiled vs interpreted (what compiling means, compiling options)
- preprocessor
- header files

RECAP: C

- Be able to write, compile, and run C programs
- Know datatypes, operators, bitwise operators
- Know I/0 (stdin/stdout and file)
- loops, conditionals, functions, switch
- arrays
- pass-by-reference vs pass-by-value

RECAP: C

- pointers + pointer arithmetic
- purpose of and when to use "dereference" and "value of" operators
- stack vs heap
- how and when to allocate memory dynamically
- difference between the various memory allocation functions
- `gdb` and `valgrind`

RECAP: C

- Makefiles
- enums, unions, structs
- 2D arrays and how they are handled
- string and memory functions