MAKEFILES

RECAP: LAST CLASS

- Talked about breaking bigger projects up into multiple files
- Using separate header files
- Compiling multi-file projects

WHAT GOES IN HEADER FILES?

- Function prototpyes
- Macros that are shared across files (or even projects)
- Struct defintions
  - If needed by other code files
  - Or if needed for function prototype

COMPILING MULTI-FILE PROJECTS

- Remembering commands gets old
- Remember file dependencies even worse
- What if you are distributing your code to others?
  - How do they know dependences
  - How do they know how to compile

SOLUTION: MAKE

- `make` = utility that determines and recompiles necessary pieces of large program
- We'll use GNU `make` (the one on EOS)
- To use `make`:
  - Create a file called `Makefile`
  - run `make` at command line
  - can also run `make some_name` - `some_name` is defined in Makefile

MAKEFILES

- Made up of rules:

```
target ... : prerequisites ...
    recipe
    ...
```

- `target` = name of file being generated by rule or name of action to perform
- `prerequisites` = files needed to create/perform target
- `recipe` = sequence of commands that make performs for that rule

SIMPLE EXAMPLE

```
all : program.c sum.c sum.h
    gcc program.c sum.c sum.h
```

- like last time, recompiles everything even if `sum.c` and `sum.h` don't change

BREAKING IT UP

- Specify rules for each object file separately

```
program : program.o sum.o
    gcc -o program program.o sum.o

program.o : program.c sum.h
    gcc -c program.c

sum.o : sum.c sum.h
    gcc -c sum.c
```

MAKING IT MORE GENERAL

- What if we wanted to change compiler?
- What if we wanted to easily specify different compilation flags?
- Answer -> Use variables in Makefile
  - Expand variables with $(varname)

```
CC = gcc
CFLAGS = -Wall

program : program.o sum.o
    $(CC) $(CFLAGS) -o program.o sum.o
```

REMOVING REPETITION

- `$@` = name of target of the rule
- `$<` = name of first prerequisite
- `$^` = space separated list of all prerequsites
- `$(basename something.extension)` - removes extension

PATTERN RULES

- Same general form as other rules
- Use '%' to express pattern to match filenames
- Example '%.o% matches any file ending it .o
- When used in prerequisite - matches the same stem in target

```
%.o : %c
    gcc -o $@ -c $<
```

MISC

- Can break up long lines with \
- Add comment with #

OTHER RULES

- Typically have a `clean` rule

```
clean:
    rm program.o sum.o program
```

- Some files have an `install` rule if they are for distributing software

MINILAB 13