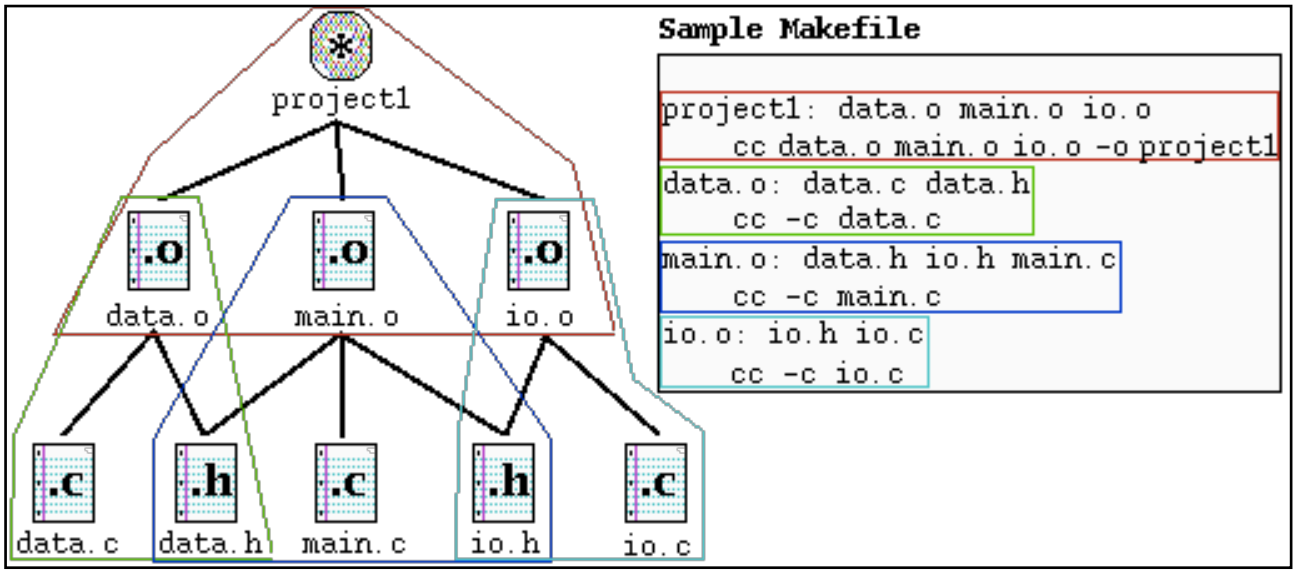
MAKEFILE NOTES:

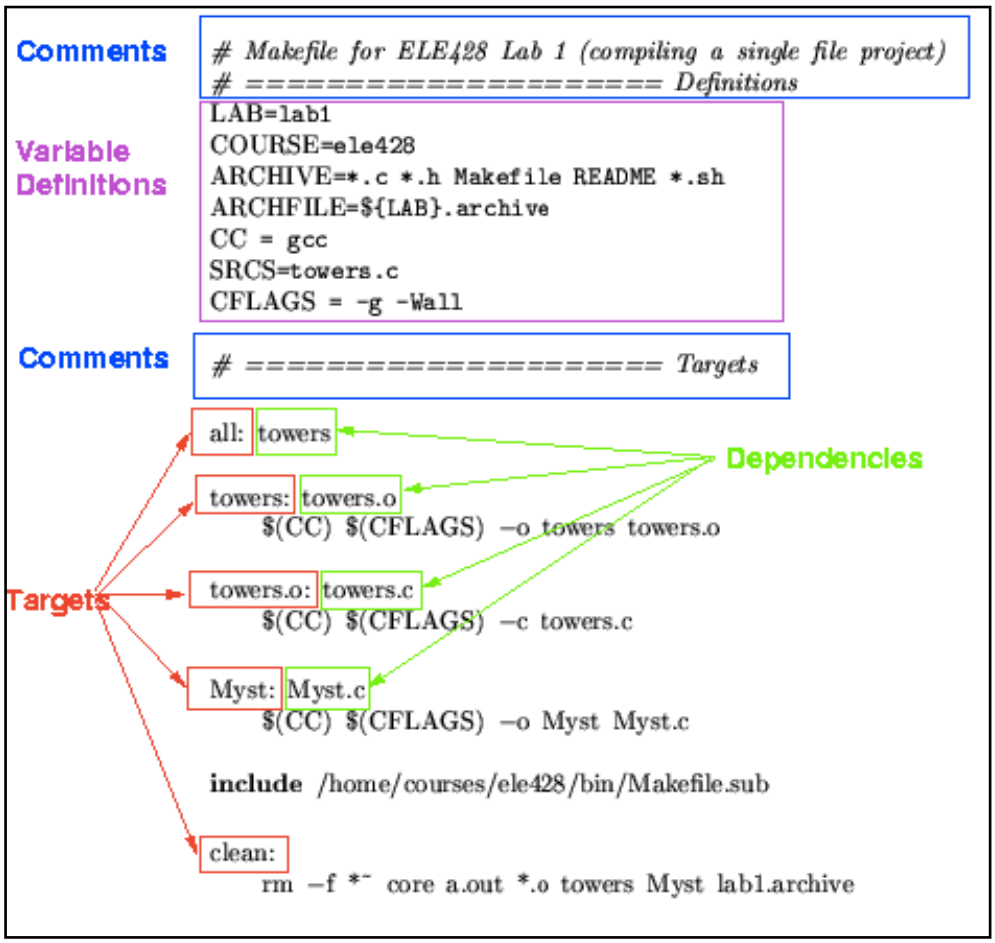
- Author: Jude Thaddeau Data
- GitHub: Zero-Luminance
- Documentation: GNU make
- Sources:
  - Barry Brown (Youtube)
  - opensource.com

MAKEFILES INTRODUCTION:

- 'Makefiles' is a text (.txt) file containing a set of tasks (e.g. compilation) to be executed via the 'make' utility
- PURPOSES: efficient compilation, testing, cleanup, program building, etc
- 'Dependencies' refers to any file that requires another file in order to function
- Example (Dependency Tree):
  - project (1st layer) -> Executable
  - .o files (2nd layer) are dependent on the .c & .h files (3rd layer)



Makefile Structure:



NOTE 1:
The remaining unannotated lines are referred to as 'actions' which must be indented by a TAB before specification

NOTE 2:
The '$()' or '${}' signs indicate the use of variable definitions

Process Of C Compilation:

- 'Compiler Tag' refers to the point at which a compiler stops file conversion
  - Example 1: gcc -S (compile up to Assembly)
  - Example 2: clang -c (compile up to Object)
  - Example 3: gcc (compile all the way up to exec)
- Compiler options for C:
  - gcc (a.k.a GNU C Compiler)
  - clang

STEP:	COMPILER TAG:	FILE TYPE:	PROCESS:	DESCRIPTION:
		.c (source)		
1	-E		Preprocessor	Handles #includes, #define & strips out comments
		Processed C file		
2	-S		Compiler	Translates C to 'Assembly' code
		.s (Assembly)		
3	-c		Assembler	Translates Assembly to an 'Object' file
		.o (Object)		
4			Linker	Compilers objects files into an 'Executable'
		exec		

Advantages Of Makefiles:

- Large projects can be partitioned into INDIVIDUAL files
- Makefiles RECOMPILES individual files that have been changed which in turn saves time & memory
- Individual files are easier to READ & DEBUG

Makefile Rules:

- NAMING: 'Makefile' without extension (Capital 'M' ensures near placement alongside READEME.md files)
- EXECUTE: 'make' utility command only works on shell/terminal when in the same directory as the makefile

Common & Useful Makefile Tasks:

OPERATION:	FORMULA:
General Format	target: dependencies @echo “message” action
Compile:	compile: @echo “Compiling ‘file.c’ into the executable ‘output’” gcc [extra tags] output file.c
Generate	generate: @echo “create a ‘file’ of type ‘filetype’” touch file.filetype
Deleting Junk Outputs	clean: @echo “Removing all files with .filetype” rm *.file-type
Executing Programs	exec1: @echo “Running program-name with ‘args’” ./program-name args
	exec2: @echo “Running ‘program-name’ with ‘args’ & ‘input’” ./program-name args < input
	exec3: @echo “Running ‘program-name’ with ‘args’ into ‘output” ./program-name args > output
A Collection Of ’n’ Operations	all: target1 target2 ... target(n)

Running Makefiles:

1) Using Shell/Terminal, enter into the same working directory as the Makefile
2) TYPE one of the following in Shell/Terminal:
   - “make target” (executes the lines with ‘target’)
   - “make all” (executes the ‘target’ “all”’s dependencies)
   - “make” (executes the FIRST ‘target’ line known as ‘default’)

NOTES:

- By default the execution order of makefile ‘targets’ follows the order of ‘dependencies’
- The FIRST target line can be OVERRIDDEN using: “.DEFAULT_GOAL := target”