MAKEFILES

WHO NEEDS ‘EM?

AGENDA

Makefile Rules, Variables

Compilation VS Linking

Makefile Basics

42-ifying a Makefile (42 PDF & Norm)

Prerequisites, Objects & Directories

Organisation & Subdirectories

Tips & Tricks

MAKEFILE RULES

A rule is defined as being a command that you run in front of make

Make - runs the first rule that is defined

Make all - ‘all’ is a rule

Make clean - ‘clean’ is a rule

To define a rule you do the following:

If you typed make helloworld, you would get:

```
$> Hello World
```

```
# <word/letter>:
# <tab><insert_commands_here>

helloworld:
    @echo "Hello World"
```

LETS START WITH THE BASICS

Here is a basic example of a makefile rule

Let’s Break it down!

First Line defining the rule all

Second Line - Compiling the source file helloworld.c into the object file helloworld.o

Third Line - Linking the helloworld.o file to the final executable sayhello

```
all:
    gcc -c helloworld.c -o helloworld.o
    gcc helloworld.o -o sayhello
```

VARIABLES

To reference a variable you can use one of many ways:

$(var)

Where var is the name of a variable:

NAME = push_swap

echo $(NAME)

Some makefile variables have defaults if not already set, the most common ones are;

Variable	Default value	Description
CC	cc	Compiling C files
CXX	g++	Compiling C++ files
CFLAGS	N/A	Extra Flags to give to the C compiler
CXXFLAGS	N/A	Extra Flags to give to the C++ compiler
RM	rm -f	Command to remove a file

https://www.gnu.org/software/make/manual/html_node/Implicit-Variables.html

COMPILATION VS LINKING

COMPILING

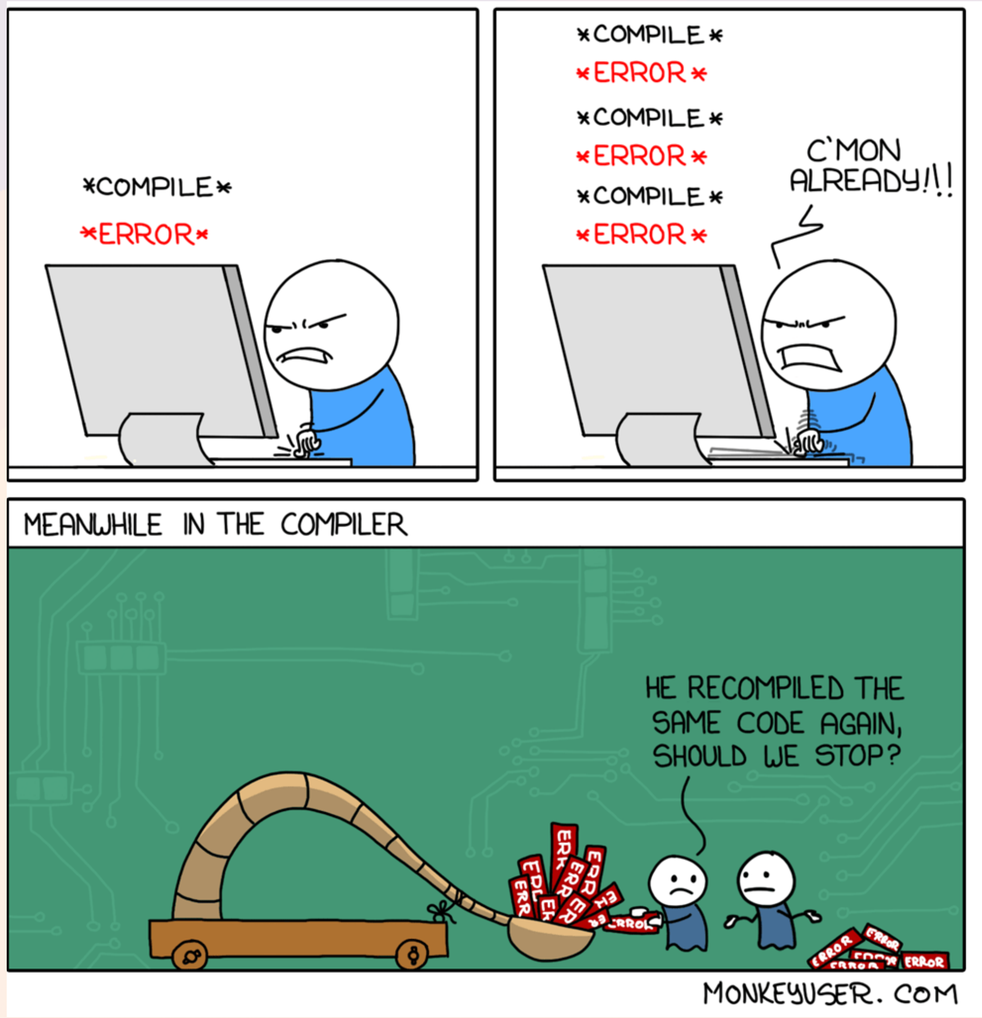
Source code is compiled into binary object code

```
gcc -c test.c -o test.o
```

LINKING

Object code is combined with other supporting code to make an executable

```
gcc test.o -o a.out
```



HTTPS://WWW.TECHONTHENET.COM/C_LANGUAGE/COMPILING_LINKING.PHP

PREREQUISITES

Rules can have prerequisites which are called first before a rule or command

```
all: $(NAME)


$(NAME)
  $(CC) $(OBJ) -o $(NAME)
```

For the all rule, $(NAME) gets called first as that is listed as a prerequisite, the generic syntax is as follows

```
rule: prerequisites
  ...
```

OBJECTS? HOW DO I USE THEM?

Have you ever been confused by this line?

```
OBJ = $(SRC:%.c=%.o)
```

The symbol % says to replace this any pattern matching .c and replace with .o

If we are given a list of files:

```
ft_strlen.c ft_putstr_fd.c main.c
```

An object file (.o) will automatically be created for each file:

```
ft_strlen.o ft_putstr_fd.o main.o
```

Example time!

HTTPS://STACKOVERFLOW.COM/QUESTIONS/40155241/HOW-TO-PROPERLY-USE-OR-ASTERISK-SYMBOLS-AS-A-PLACEHOLDER-IN-MAKEFILE

INCLUDING HEADERS

If your project requires a header/include file, you use the -I flag to specify the location:

```
-I <dir>
```

Where <dir> is the directory where your header files are located.

If you have a directory setup for the project you would usually do something like this:

```
-I includes
```

Or if your headers are in the same directory as your sources

```
-I .
```

Lets do an example!

WHAT DOES THIS MEAN? - NORMINETTE

III.11 Makefile

Makefiles aren't checked by the Norm, and must be checked during evaluation by the student.

- The $(NAME), clean, fclean, re and all rules are mandatory.
- If the makefile relinks, the project will be considered non-functional.
- In the case of a multibinary project, in addition to the above rules, you must have a rule that compiles both binaries as well as a specific rule for each binary compiled.
- In the case of a project that calls a function from a non-system library (e.g.: `libft`), your makefile must compile this library automatically.
- All source files you need to compile your project must be explicitly named in your Makefile.

42-IFYING A MAKEFILE

Your `Makefile` must at least contain the rules `$(NAME)`, `all`, `clean`, `fclean` and `re`.

	Example	Description
$(NAME)	`NAME = push_swap`	Assign the NAME variable to be push_swap
all	`all: $(NAME)`	When make is run, run the rule all which will run $(NAME) first
clean	`clean:` `  $(RM) $(NAME)`	Uses the default definition of RM (rm -f) to remove the $(NAME) executable
fclean	`fclean:` `  $(RM) *.o`	Uses the default definition of RM (rm -f) to remove all the files ending in .o

RELINKING - WHAT DOES IT MEAN?

When we link we use all the object files to make the final executable:

```
gcc ft_strlen.o main.o -o finalexe
```

The make utility knows when the files were last linked to the object file by using the last modified timestamp of each file.

If the makefile isn’t setup properly, it will re-link each file every time make is run.

If your makefile is setup properly, make will print out:

make: Nothing to be done for ‘all’

If any files have been edited, make will only compile and link it back to the program executable.

If your makefile is linking all objects back to the executable, then it is not set up properly.

WHAT ABOUT BONUSES?

Ever wanted to do bonuses for a project and been confused on how to write a makefile for it?

It can be confusing but lets see an example!

ORGANISATION

All this advice is great, but I want to organise everything? Lets do it!

You can use any variable name but for simplicity I like to append _DIR to the original variable:

```
SRC = ft_strlen.c main.c
SRC_DIR = sources
```

If you want to use the directory throughout, you can reference it using variable substitution:

```
$(SRC_DIR)
```

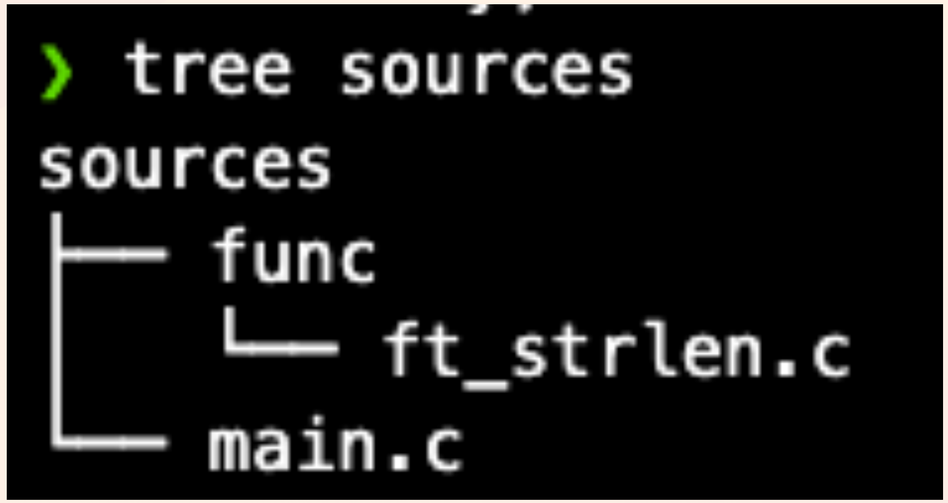
You can do this with any directory to fully organise your project too!

ORGANISATION

If you want to use this throughout the makefile, you have to make sure your variables are defined before using them:

```
SRC_DIR = sources

OBJ_DIR = objects

SRC = $(SRC_DIR)/main.c

OBJ = $(SRC:$(SRC_DIR)/%.c=$(OBJ_DIR)/%.o)

$(OBJ_DIR)/%.o : $(SRC_DIR)/%.c

  …
```

This is also useful if you would like to put your files inside subdirectories within the sources folder. For Example

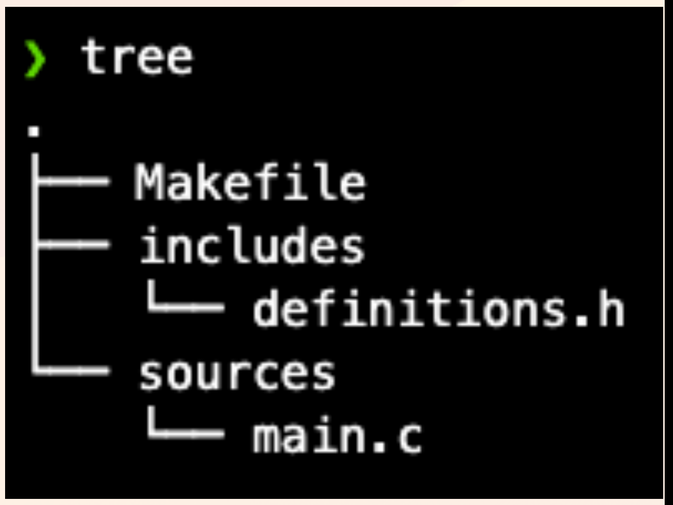


TYPICAL PROJECT LAYOUT

I like to follow is a generic project layout

Starting with the following directory structure:

sources - Contains source files
includes - Contains header files
libraries - Contains any libraries I need (if applicable)



NEAT TIPS AND TRICKS

Instead of using @ symbol to silence output of every command inside a rule:

Add .SILENT: to your makefile before the rules to silence any output from commands

You can include a makefile inside another makefile using:

-include <name>

RESOURCES & REFERENCES

Makefile Cookbook - https://makefiletutorial.com

GNU Make Manual - https://www.gnu.org/software/make/manual/

My Example Makefiles - https://github.com/Nazza01/Example-Makefile

Naming Makefiles - https://www.gnu.org/software/make/manual/html_node/Makefile-Names.html

Makefile Cheat Sheet - https://cheatography.com/bavo-van-achte/cheat-sheets/gnumake/

42 StackOverflow Makefile Good Practices - https://stackoverflow.com/c/42network/questions/1604

THANKYOU FOR ATTENDING!