Lab Report

ECPE 170 – Computer Systems and Networks – Spring 2021

Name: Kaung Khant Pyae Sone

Lab Topic: C Programming (Language, Tools, and Makefiles) (Lab #: 3)

Question #1:
Copy and paste in your functional Makefile-1
Answer:
```
all:
      gcc main.c output.c factorial.c -o factorial_program
```

Question #2:
Copy and paste in your functional Makefile-2
Answer:
```
all: factorial_program

factorial_program: main.o factorial.o output.o
      gcc main.o factorial.o output.o -o factorial_program

main.o: main.c
      gcc -c main.c

factorial.o: factorial.c
      gcc -c factorial.c

output.o: output.c
      gcc -c output.c

clean:
      rm -rf *.o factorial_program
```

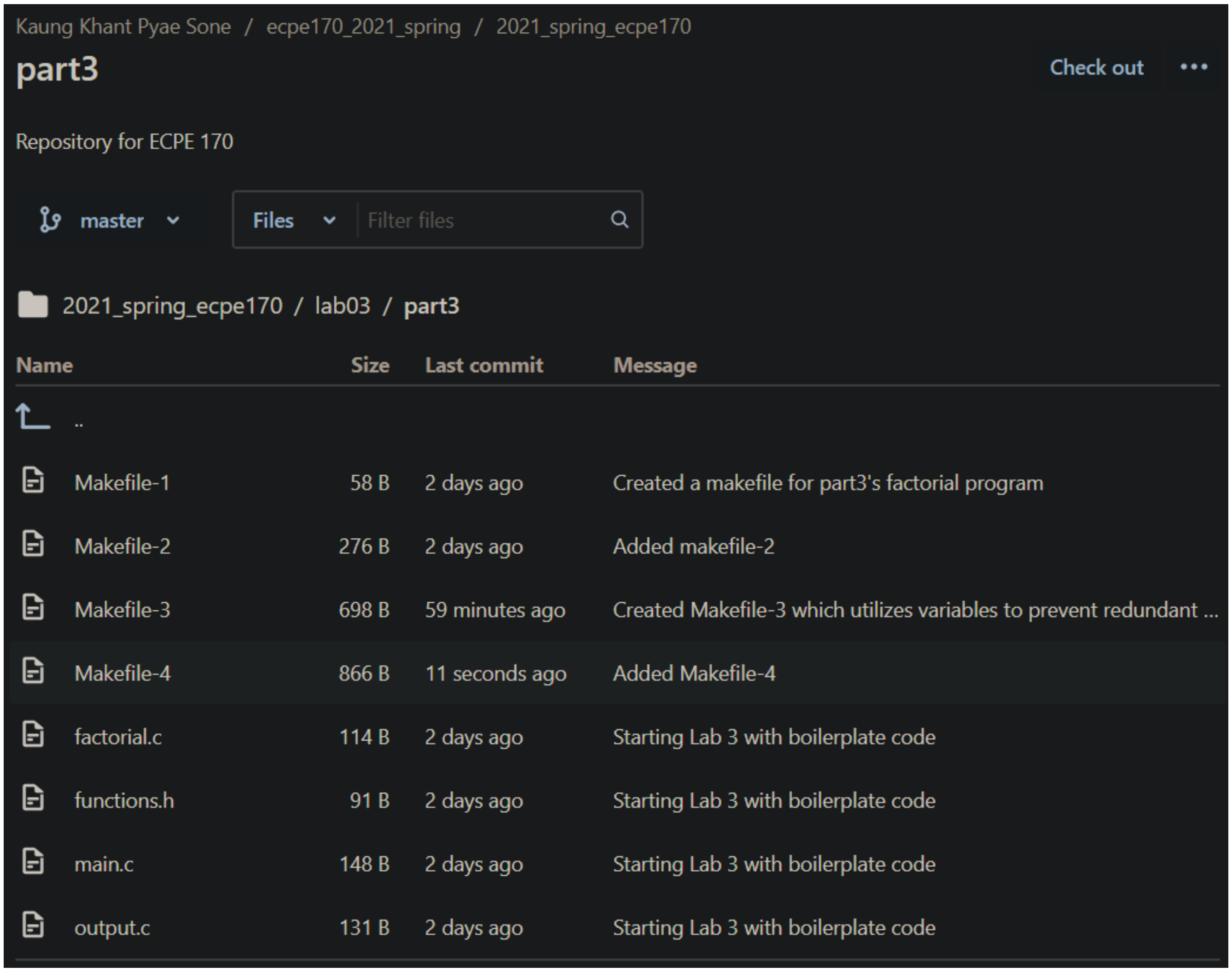
Question #3:
Describe - in detail - what happens when the command "make -f Makefile-2" is entered. **How does make step through your Makefile to eventually produce the final result?**
Answer:
First, make checks if your .c files have changed or not. If they have changed or aren't compiled yet, make first compiles all the dependencies, in our case, main.o, factorial.o and output.o. After that, make runs the main command which is under factorial_program then finally compiles the whole program and generates the factorial_program executable. After compiling, make cleans up .o files generated during the compilation of factorial_program.

Question #4:
Copy and paste in your functional Makefile-3
Answer:
```
# The variable CC specifies which compiler will be used.
# (because different unix systems may use different compilers)
CC=gcc

# The variable CFLAGS specifies compiler options
#   -c :    Only compile (don't link)
#   -Wall:  Enable all warnings about lazy / dangerous C programming
CFLAGS=-c -Wall

# The final program to build
EXECUTABLE=factorial_program

# --------------------------------------------

all: $(EXECUTABLE)

$(EXECUTABLE): main.o factorial.o output.o
	$(CC) main.o factorial.o output.o -o $(EXECUTABLE)

main.o: main.c
	$(CC) $(CFLAGS) main.c

factorial.o: factorial.c
	$(CC) $(CFLAGS) factorial.c

output.o: output.c
	$(CC) $(CFLAGS) output.c

clean:
	rm -rf *.o $(EXECUTABLE)
```

Question #5:
Copy and paste in your functional Makefile-4
Answer:
```
# The variable CC specifies which compiler will be used.
# (because different unix systems may use different compilers)
CC=gcc

# The variable CFLAGS specifies compiler options
#   -c :    Only compile (don't link)
#   -Wall:  Enable all warnings about lazy / dangerous C programming
#  You can add additional options on this same line..
#  WARNING: NEVER REMOVE THE -c FLAG, it is essential to proper operation
CFLAGS=-c -Wall

# All of the .h header files to use as dependencies
HEADERS=functions.h

# All of the object files to produce as intermediary work
OBJECTS=main.o factorial.o output.o

# The final program to build
EXECUTABLE=factorial_program

# --------------------------------------------

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) $(OBJECTS) -o $(EXECUTABLE)

%.o: %.c $(HEADERS)
	$(CC) $(CFLAGS) -o $@ $<

clean:
	rm -rf *.o $(EXECUTABLE)
```

Question #6:
Describe - in detail - what happens when the command "make -f Makefile-4" is entered. How does make step through your Makefile to eventually produce the final result?
Answer:
It goes through the variables first, in this case, CC specifies which compiler will be used, CFLAGS specify the options for the compiler, HEADERS represents the header files, OBJECTS are the object files used during compilation, EXECUTABLE is the final program to build and can be executed. After that, it replaces the commands with the information from the variables, then the commands are run. First, make checks if all the .c files have changed or not. If they have changed or aren't compiled yet, make first compiles all the dependencies, in our case, main.o, factorial.o and output.o. After that, make runs the main command which is under factorial_program then finally compiles the whole program and generates the factorial_program executable. After compiling, make cleans up .o files generated during the compilation of factorial_program.

Question #7:
To use this Makefile in a future programming project (such as Lab 4...), what specific lines would you need to change?
Answer:
HEADERS, OBJECTS and EXECUTABLE

Question #8:
Take one screen capture of the Bitbucket.org website, clearly showing the "Part 3" source folder that contains all of your Makefiles added to version control, along with the original boilerplate code.
Answer:

Kaung Khant Pyae Sone / ecpe170_2021_spring / 2021_spring_ecpe170
part3
Check out
Repository for ECPE 170
master
Files
Filter files
2021_spring_ecpe170 / lab03 / part3

Name	Size	Last commit	Message
..			
Makefile-1	58 B	2 days ago	Created a makefile for part3's factorial program
Makefile-2	276 B	2 days ago	Added makefile-2
Makefile-3	698 B	59 minutes ago	Created Makefile-3 which utilizes variables to prevent redundant ...
Makefile-4	866 B	11 seconds ago	Added Makefile-4
factorial.c	114 B	2 days ago	Starting Lab 3 with boilerplate code
functions.h	91 B	2 days ago	Starting Lab 3 with boilerplate code
main.c	148 B	2 days ago	Starting Lab 3 with boilerplate code
output.c	131 B	2 days ago	Starting Lab 3 with boilerplate code