Name: Ramon Arambula

Lab Topic: C Programming (Language, Toolchain, and Makefiles) (Lab #: 2)

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 one step through your Makefile to eventually produce the final result?
Answer:

1. When Makefile-2 is loaded, the all keyword = the target, where the dependencies follow, which means that everything will be loaded when *make* is made.
2. For every file in the directory, we listed the target (denoted by “.o” files), and its dependency (its corresponding “.c” file).
    a. Each file had a separate compilation attached
3. Finally, we included that the make file delete any residual “.o” files from the directory once it concluded compilation.

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)
```

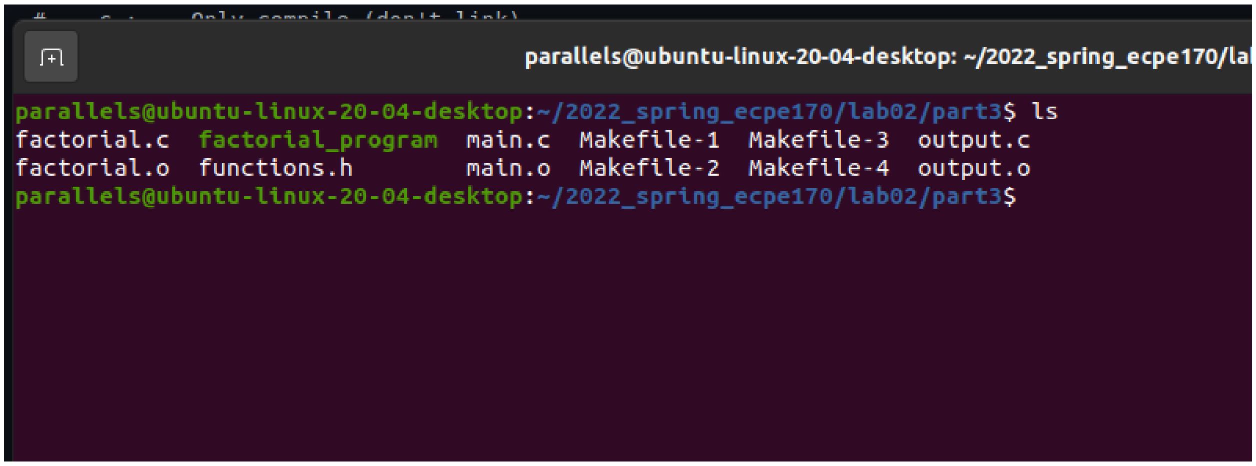
Question #6:
Describe - in detail - what happens when the command “make -f Makefile-4” is entered. How does make step through your Makefile to eventually produce the final result?
Answer:

1. When make file is called, we initialize a variable CC, which corresponds to what compiler we are going to use. In this case, CC=gcc
2. Next we define the CFLAGS, we’re using -c and -Wall
   a. -c: only compiles the file, rather than linking
   b. -Wall: warns you if you did something wrong
3. Next we initialized variables that corresponded to HEADERS, OBJECTS, and EXECUTABLES. Each variable had files that corresponded to the variable name
   a. The headers and objects work together to create the executable file.
4. All: $(EXECUTABLE) corresponds to the final build file that is to be compiled and created
   a. The $(EXECUTABLE): $(OBJECTS) line deals with all of the dependencies(OBJECTS) that are too be compiled for the final target (EXECUTABLE)
   b. The %.o: %.c $(HEADERS) line deals with compiling the header files (any file ending in .o), where its dependencies are (.c)
5. Clean removes all the created .o files when the makefile is ran

Question #7:
To use this Makefile in a future programming project (such as Lab 3…), what specific lines would you need to change?
Answer:
Lines:

- Line 3 possibly, if another compiler is used
- Line 13, since there may be multiple header files
- Line 16, since the object files will differ
- Line 19, since the executable program can have a different name

Question #8:
Take one screen capture of the your code, clearly showing the “Part 3” source folder that contains all your Makefiles, along with the original boilerplate code.
Answer:



parallels@ubuntu-linux-20-04-desktop: ~/2022_spring_ecpe170/la

```
parallels@ubuntu-linux-20-04-desktop:~/2022_spring_ecpe170/lab02/part3$ ls
factorial.c  factorial_program  main.c  Makefile-1  Makefile-3  output.c
factorial.o  functions.h        main.o  Makefile-2  Makefile-4  output.o
parallels@ubuntu-linux-20-04-desktop:~/2022_spring_ecpe170/lab02/part3$
```