Lab 2 Report

ECPE 170 – Computer Systems and Networks – Spring 2022

Name: Nicolas Ahn

Lab Topic: C Programming(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 make step through your Makefile to eventually produce the final result?**
Answer:
Make if used to automate compiling programs. It determines automatically which pieces of a large program needs to be recompiled, and issue the commands to recompile them.
When the command “make -f Makefile-2” is entered, the same output is produced as before. The difference is that it has additional dependencies that add more flexibility – if only one file

changes, make won't recompile everything. The makefile-2 file specifies what dependencies to recompile. For example, in order to generate output.o file, output.c needs to be recompiled.

Question #4a:
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:

The new Makefile-4 make file have additional variables to help in the process of automating compilation. The variable HEADERS makes sure that if a header file changes, the object files will be rebuilt. The variable OBJECTS lists all intermediary .o files in the project (the final executable program target depends on all the object files to be built).

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

Answer:

The lines to be changed are the variables on the top of the makefile, such as the HEADERS, OBJECTS, and EXECUTABLE variables.

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

Answer: