Lab Report

ECPE 170 – Computer Systems and Networks – Fall 2021

Name: Dustin Schuette

Lab Topic: Makefile (Lab #: 3)

1)all:
	gcc main.c output.c factorial.c -o factorial_program

2)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

3) When make -f Makefile-2 is entered it sees that "-f" is a modifier and thus accepts the name of the makefile which we provide. It then looks to see the all which shows that all the files are necessary in the factorial_program if we want it to compile, the next line specifies which files these are. It then goes file by file to see how each one should be compiled does so using the command provided, lastly the clean snippet is to tell the computer which files can be deleted after each run.

4)# 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)
```

5)# 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)
```

6) So, just as before we start with the -f flag which allows us to input the file name, after this we start by listing the compiler to use to ensure compatibility among different systems, we then set our c flags

which we need to stop it from linking the files and in this case we also set earnings to tell us if our code is poorly written. After this we add the header files, object files and finally name the final program we are trying to build. As we have separated each of our file types and assigned them to categories we can simply write one instruction with their generic assigned type and it will apply to all files of that type such as object or header. We then do so, we call all the object files to be compiled then the headers to be added and finally we have the clean up at the bottom which we can call to remove temporary files.

7)To use this makefile for future projects we would switch out the files listed above the line as those will change with whatever project we are working on, the stuff below the line is abstracted and will run on the given files, so we do not need to change them.

8)