Lab Report

ECPE 170 – Computer Systems and Networks – Spring 2016

Name: Dominic Lesaca

Lab Topic: C Programming (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:
The makefile goes to the all command first and looks at what program it needs to build, in this case it would be factorial_program. When building factorial_program, the makefile checks to see if all of the necessary .o files are up to date and unpdates any who need it. When they are all up to date, main.o, factorial.o, and output.o are called in 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:

The Makefile declares all of the headers to use as dependencies, the object files, and the executable to build as the final program (factorial_program). It then looks at the all command to see that it must build the executable. The makefile checks if all of the object files are up to date and updates them if necessary. Then the object files are called in order to run the executable.

Question #7:

To use this Makefile in a future programming project (such as Lab 4...), what specific lines would you need to change?

Answer:

You would need to change the headers on line 13, the objects on line 16, and the executable on line 19

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: